Entwicklerhandbuch, Version 1

AWS IoT Greengrass

AWS IoT Greengrass: Entwicklerhandbuch, Version 1

Table of Contents

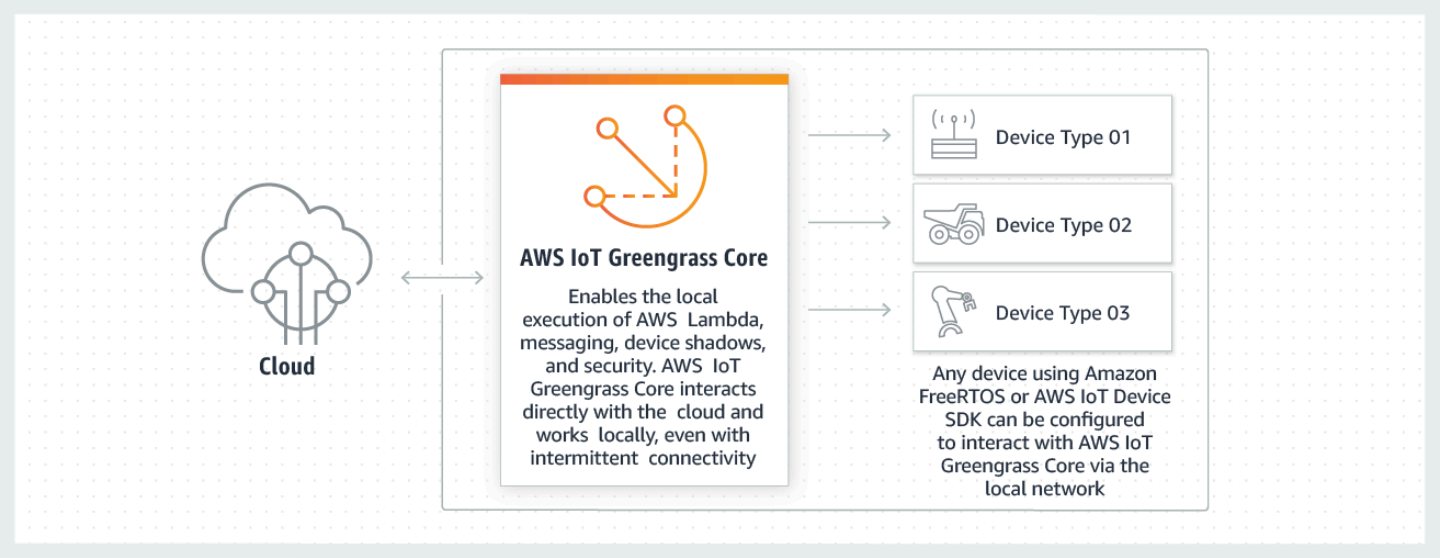
AWS IoT Greengrass Version 1 trat am 30. Juni 2023 in die erweiterte Lebensphase ein. Weitere Informationen finden Sie in der AWS IoT Greengrass V1 Wartungsrichtlinie. Nach diesem Datum AWS IoT Greengrass V1 werden keine Updates mehr veröffentlicht, die Funktionen, Verbesserungen, Bugfixes oder Sicherheitspatches bieten. Geräte, die auf laufen, werden AWS IoT Greengrass V1 nicht gestört und funktionieren weiterhin und stellen eine Verbindung zur Cloud her. Wir empfehlen Ihnen dringend, zu migrieren AWS IoT Greengrass Version 2, da dies wichtige neue Funktionen und Unterstützung für zusätzliche Plattformen bietet.

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
  - Ich ver-the-air aktualisiere den Agenten
  - Stream-Manager
  - Zugriff auf lokale Ressourcen
  - Lokale Machine Learning-Inferenz
  - Manager lokaler Secrets
  - Verbindungen mit werkseitiger Integration mit Services, Protokollen und Software

Themen

- SDKs
- Unterstützte Plattformen und Anforderungen
- AWS IoT Greengrass lädt herunter
- Bitte geben Sie uns Feedback
- Installieren Sie die AWS IoT Greengrass Core-Software.
- Konfigurieren des AWS IoT Greengrass Core

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 über AWS IoT Greengrass APIs konfiguriert, die in der Cloud gespeicherte AWS IoT Greengrass Gruppendefinitionen 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 Bereitstellung von Greengrass 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.
- O ver-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 finden 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. OpenWrt Unterstützt derzeit keine ML-Inferenz.

1.9.1

Fehlerbehebungen und Verbesserungen:

- Es wird ein in v1.9.0 eingeführter Fehler behoben, der Nachrichten aus `cloud` 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 “Festlegen 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”.
- AWS IoT Vorhersehbar benannte Client-IDs 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. Die vorhersehbare erleichtert auch die das Erstellen von Logik zu Verbindungs-IDs (z. B. zum Erstellen von Vorlagen für eine Abonnieren-Richtlinie basierend auf Zertifikatattributen). 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 “Steuern der Ausführung der Greengrass-Lambda-Funktion”.
- 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 “Ausführen von AWS IoT Greengrass in einem Docker-Container”.
- 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 Neo-Deep-Learning-Laufzeit, die Modelle für maschinelles Lernen unterstützt, die vom SageMaker 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 “Lambda-Ausführungsdateien”.
- 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 Core-Konfigurationsdatei”.
- 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:

- O ver-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.

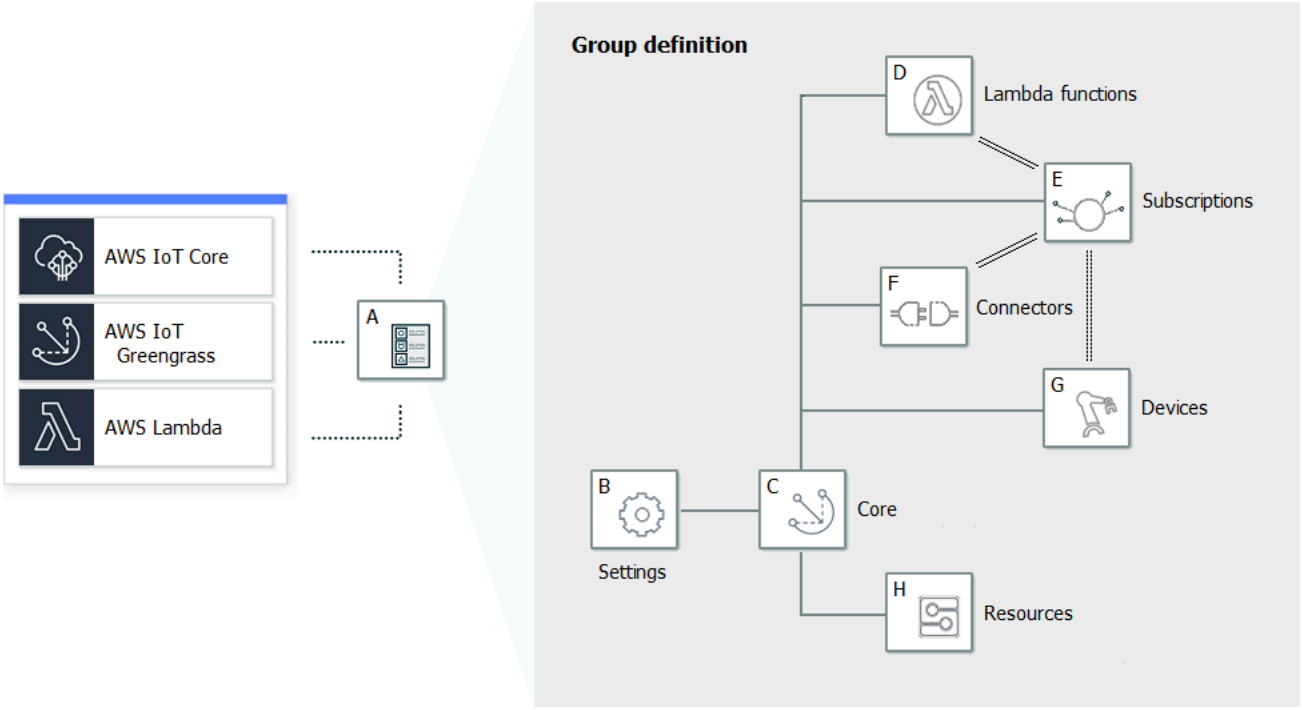
- Stellen Sie die Gruppe auf dem neuen Core bereit.

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 “Festlegen der Standard-Containerisierung für Lambda-Funktionen in einer Gruppe”.
- 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 “Konfigurieren des AWS IoT Greengrass Core”.

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 Bereitstellen von AWS IoT Greengrass-Gruppen.

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 ermitteln können. Weitere Informationen finden Sie unter the section called “Konfigurieren des AWS IoT Greengrass Core”.

> **Note**
>
> Eine Greengrass-Gruppe muss genau einen Kern enthalten.

Client-Gerät

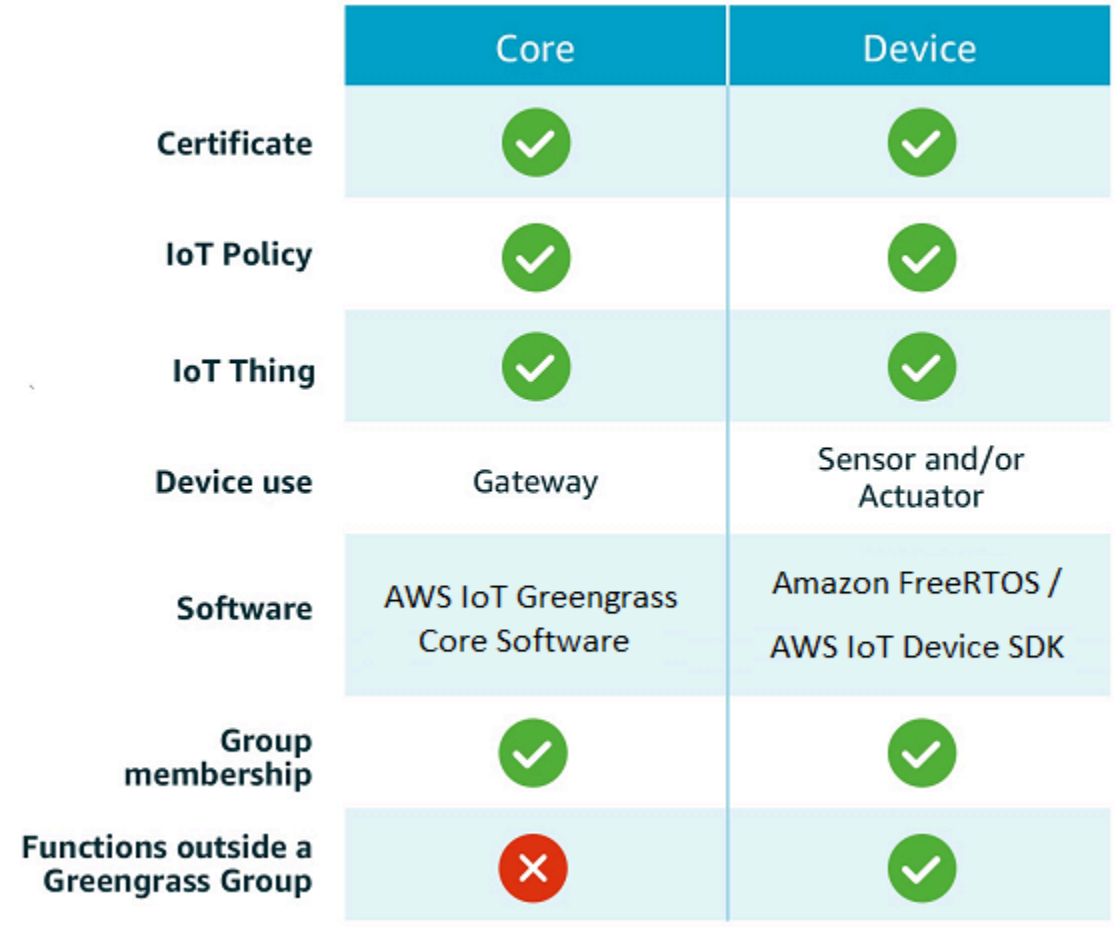
Client-Geräte (auch verbundene Geräte, Greengrass-Geräte oder Geräte genannt) sind Geräte, die über MQTT eine Verbindung zu einem Greengrass-Core herstellen. Sie verfügen über ein eigenes Gerätezertifikat für die AWS IoT Core Authentifizierung, einen Geräteshadow und einen Eintrag in der Registrierung. 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 Greengrassthe section called “Modul 4: Interagieren mit Client-Geräten in einemAWS IoT GreengrassGruppe” Beispiele, die Ihnen zeigen, wie Sie mit AWS CLI dem ein Client-Gerät erstellen und konfigurieren AWS IoT Greengrass, finden Sie create-device-definitionin 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.



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

SDKs

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

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 den AWS SDKs verfügbar sind, sind auch in der AWS IoT Greengrass API und verfügbar. AWS CLI

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 Geräte-SDKs im AWS IoT Entwicklerhandbuch.

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

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 “Ausführen von AWS IoT Greengrass in einem Docker-Container”.

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. Anweisungen finden Sie unter the section called “Festlegen der Standard-Containerisierung für Lambda-Funktionen in einer Gruppe”.
- 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
  - 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 “Ausführen von AWS IoT Greengrass in einem Docker-Container”.

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. Anweisungen finden Sie unter the section called “Festlegen der Standard-Containerisierung für Lambda-Funktionen in einer Gruppe”.
- 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 3.7 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 “Ausführen von AWS IoT Greengrass in einem Docker-Container”.

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. Anweisungen finden Sie unter the section called “Festlegen der Standard-Containerisierung für Lambda-Funktionen in einer Gruppe”.

- 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.
- 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`
  - `df`

- `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 “Ausführen von AWS IoT Greengrass in einem Docker-Container”.
  - 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. Anweisungen finden Sie unter the section called “Festlegen der Standard-Containerisierung für Lambda-Funktionen in einer Gruppe”.
- 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 Bereitstellung von Greengrass 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
Armv8 (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:

- O ver-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
Armv8 (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. OpenWrt Unterstützt derzeit keine ML-Inferenz.

- 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 finden 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
Armv8 (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 “Festlegen 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”.
- AWS IoT Vorhersehbar benannte Client-IDs 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. Die vorhersehbare erleichtert auch die das Erstellen von Logik zu Verbindungs-IDs (z. B. zum Erstellen von Vorlagen für eine Abonnieren-Richtlinie basierend auf Zertifikatattributen). 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 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
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 unterthe 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 “Ausführen von AWS IoT Greengrass in einem Snap”.

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 für die Erstellung 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 AArch64) installiert. 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 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 Version 1Wartungspolitik.

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

- Verwenden Sie für Docker Hub das *`Versions-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 unterAusführen von AWS IoT Greengrass in einem Docker-Container.

> **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. Beispielsweise:
>
> ```
> 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 Kern zu interagieren. Auf diese Weise können bereitgestellte Lambda-Funktionen:

- Tauschen Sie MQTT-Nachrichten aus mit. AWS IoT Core
- Tauschen Sie MQTT-Nachrichten mit Connectoren, 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 Core-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 Machine Learning SDKDadurch können die von Ihnen Lambda 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 GreengrassCore-Software erweitert die AWS Funktionalität auf ein AWS IoT Greengrass-Core-Gerät, sodass lokale Geräte lokal auf die von ihnen generierten Daten reagieren können.

AWS IoT Greengrass bietet verschiedene 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.

- Ausführen von AWS IoT Greengrass in einem Docker-Container.
- Ausführen von AWS IoT Greengrass in einem Snap.

Download und Extrahieren des AWS IoT Greengrass Core-Software-Pakets

Wählen Sie die AWS IoT Greengrass Core-Software für Ihre Plattform, die Sie als tar.gz-Datei herunterladen und auf Ihr Gerät extrahieren möchten. 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 die Greengrass-Geräteeinrichtung aus, um Ihr Gerät zu konfigurieren, installieren Sie die neueste AWS IoT Greengrass -Core-Softwareversion und stellen Sie in wenigen Minuten eine Hello World Lambda-Funktion bereit. Weitere Informationen finden Sie unter the section called “Schnellstart: Greengrass-Geräteeinrichtung”.

Installieren der 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 aus einem APT-Repository installieren oder aktualisieren. Auf Geräten, auf denen Sie das AWS IoT GreengrassRepository 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 Ihnen, die -AWS IoT GreengrassCore-Software mit tar-Dateien zu aktualisieren.

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

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

  Mit dem Download dieser Software stimmen Sie der Greengrass Core-Software-Lizenzvereinbarung zu.

Themen

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

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

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

```
systemctl enable greengrass.service
```

- So starten Sie den Greengras-Daemon:

```
systemctl start greengrass.service
```

- So halten Sie den Greengrass-Daemon an:

```
systemctl stop greengrass.service
```

- So überprüfen Sie den Status des Greengras-Daemons:

```
systemctl status greengrass.service
```

Deinstallieren der AWS IoT Greengrass Core-Software mithilfe des APT-Repositorys

Wenn Sie die AWS IoT Greengrass Core-Software deinstallieren, können Sie wählen, ob die Konfigurationsinformationen der AWS IoT Greengrass Core-Software beibehalten oder entfernt werden sollen, z. B. Gerätezertifikate, Gruppeninformationen und Protokolldateien.

So deinstallieren Sie die AWS IoT Greengrass Core-Software und behalten Konfigurationsinformationen bei

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

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

So deinstallieren Sie die AWS IoT Greengrass Core-Software und entfernen Konfigurationsinformationen

1. Führen Sie den folgenden Befehl aus, um die AWS IoT Greengrass Core-Softwarepakete und Konfigurationsinformationen aus der 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 der Repository-Quellen für die AWS IoT Greengrass Kernsoftware.

Entfernen der Repository-Quellen für die AWS IoT Greengrass Kernsoftware

Sie können die AWS IoT Greengrass Kernsoftware-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 beim Ausführen von zu vermeiden`apt update`.

So entfernen Sie das APT-Repository aus der Quellenliste

- 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
  ```

Ausführen von AWS IoT Greengrass in einem Docker-Container

AWS IoT Greengrass stellt ein Dockerfile und Docker-Images zur Verfügung, mit deren Hilfe Sie AWS IoT Greengrass leichter in einem Docker-Container ausführen können. 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.

Ausführen von AWS IoT Greengrass in einem Snap

AWS IoT Greengrass Mit Snap 1.11.x können Sie eine begrenzte Version von AWS IoT Greengrass über praktische Softwarepakete zusammen mit allen erforderlichen Abhängigkeiten in einer containerisierten Umgebung ausführen.

Am 31. Dezember 2023 AWS IoT Greengrass wird die Wartung für die AWS IoT Greengrass Core-Softwareversion 1.11.x Snap beenden, die auf snapcraft.io veröffentlicht wird. Geräte, auf denen derzeit der Snap ausgeführt wird, funktionieren bis auf Weiteres. Der AWS IoT Greengrass Kern-Snap erhält jedoch nach dem Ende der Wartung keine Sicherheitspatches oder Fehlerbehebungen mehr.

Snap-Konzepte

Im Folgenden finden Sie grundlegende Snap-Konzepte, die Ihnen helfen, zu verstehen, wie Sie AWS IoT Greengrass Snap verwenden:

Channel

Eine Snap-Komponente, die definiert, welche Version eines Snap installiert und für Updates verfolgt 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 verbunden sein. Beachten Sie, dass zuerst verbunden 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

Snaps werden automatisch aktualisiert. Der `snapd` Daemon ist der Snap-Paketmanager, der standardmäßig viermal täglich nach Updates sucht. Jede Aktualisierungsprüfung wird als Aktualisierung bezeichnet. Wenn eine Aktualisierung erfolgt, wird der Daemon beendet, der Snap aktualisiert und der Daemon wird dann neu gestartet.

Weitere Informationen finden Sie auf der Snapmaker-Website.

Was ist neu bei AWS IoT Greengrass Snap v1.11.x?

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

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

> **Important**
>
> Da sich nicht containerisierte Lambda-Funktionen denselben Benutzer teilen müssen (`snap_daemon`), haben die Lambda-Funktionen keine Isolierung voneinander. Weitere Informationen finden Sie unter Steuern der Ausführung von Greengrass-Lambda-Funktionen mithilfe der 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 Python-3.7-Lambda-Funktionen tatsächlich die Python-3.8-Laufzeit aus.

Erste Schritte mit AWS IoT Greengrass Snap

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

Voraussetzungen

Gehen Sie wie folgt vor, um den AWS IoT Greengrass Snap auszuführen:

- 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 unter Installing Snapd in der Snap-Dokumentation.

Installieren und Konfigurieren des AWS IoT Greengrass Snap

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-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
   ```

   Snap ist ein Basis-Snap`core18`, der eine Laufzeitumgebung mit häufig verwendeten Bibliotheken bereitstellt. Dieser Snap basiert auf Ubuntu 18.04 LTS .

2. Führen Sie ein Upgrade durch, `snapd` 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 Snap AWS IoT Greengrass installiert ist.

   Die folgende Beispielantwort zeigt, dass 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 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 zu 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 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 v1.11.5 aktualisiert.

Sie können den `snap info aws-iot-greengrass` Befehl ausführen, um die Liste der verfügbaren Kanäle für abzurufenAWS 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 Greenrgrass 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: SRDuhPJGj4XPxFNNZQKOTvURAp0wxKnd
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 vom AWS IoT Greengrass Snap unterstützten Schnittstellen abzurufen:

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

   Beispielantwort:

   ```
   Interface               Plug                                            Slot
                Notes
   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 in der Spalte Slot ein Bindestrich (-) angezeigt wird, ist die entsprechende Schnittstelle nicht verbunden.

6. Folgen Sie Installieren der AWS IoT Greengrass Core-Software, um ein -AWS IoTObjekt, eine Greengrass-Gruppe, Sicherheitsressourcen, die eine sichere Kommunikation mit ermöglichenAWS IoT, und die AWS IoT Greengrass Core-Softwarekonfigurationsdatei zu erstellen. Die Konfigurationsdatei `config.json`enthält eine für Ihren Greengrass-Kern spezifische Konfiguration, z. B. den Speicherort der Zertifikatsdateien und den AWS IoT Gerätedatenendpunkt.

> **Note**
>
> Wenn Sie die Datei auf ein anderes Gerät heruntergeladen haben, führen Sie diesen Schritt aus, um die Dateien auf das AWS IoT Greengrass Core-Gerät zu übertragen.

7. Stellen Sie für AWS IoT Greengrass Snap sicher, dass Sie die Datei config.json aktualisieren, wie im Folgenden gezeigt:

   - Ersetzen Sie jede Instance 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 Instance 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.keyy"
      },
      "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
```

Bereitstellen einer Lambda-Funktion

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

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

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

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

   Beispielantwort:

   ```
   Started.
   ```

   > **Note**
   >
   > Wenn Sie einen Fehler erhalten, können Sie den `snap run` Befehl für eine detaillierte Fehlermeldung verwenden. Weitere Informationen zur Fehlerbehebung finden Sie unter Fehler: kann die folgenden Aufgaben nicht ausführen: - Ausführen des Servicebefehls „start“ für Services ["greengrassd"] von snap „aws-iot-greengrass“ ([start snap.aws-iot-greengrass.greengrassd.service] ist mit dem Beendigungsstatus 1 fehlgeschlagen: Auftrag für snap.aws-iot-greengrass.greengrassd.service fehlgeschlagen, da der Steuerungsprozess mit dem Fehlercode beendet wurde. Weitere Informationen finden Sie unter „Systemctl status snap.aws-iot-greengrass.greengrassd.service“ und „journalctl -xe“.).

2. Führen Sie den folgenden Befehl aus, um zu bestätigen, dass 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 auf AWS IoT Greengrass , um eine Hello World Lambda-Funktion zu erstellen und bereitzustellen. Bevor Sie die Lambda-Funktion bereitstellen, führen Sie jedoch den nächsten Schritt aus.
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 in der AWS IoT Greengrass-Konsole an.
   b. Erweitern Sie im Navigationsbereich der AWS IoTKonsole 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 Seite Gruppenkonfiguration im Navigationsbereich die Registerkarte Lambda-Funktionen aus.
   e. Wählen Sie unter Standard-Lambda-Funktionslaufzeitumgebung die Option Bearbeiten aus und gehen Sie wie folgt vor:
      i. Wählen Sie für Standardsystembenutzer und -gruppe die Option Andere Benutzer-ID/Gruppen-ID aus und geben Sie dann sowohl **584788** für Systembenutzer-ID (Nummer) als auch für Systemgruppen-ID (Nummer) ein.
      ii. Wählen Sie für Standard-Lambda-Funktionscontainerisierung die Option Kein Container aus.
      iii. Wählen Sie Speichern.

Stoppen des AWS IoT Greengrass Daemon

Sie können den `snap stop` Befehl verwenden, um einen Service 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 zurückgeben`Stopped.`.

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

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

Beispielantwort:

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

Deinstallieren des AWS IoT Greengrass Snap

Um den AWS IoT Greengrass Snap zu deinstallieren, führen Sie den folgenden Befehl aus:

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

Beispielantwort:

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

Fehlerbehebung bei AWS IoT Greengrass Snap

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

Fehler aufgrund einer verweigerten Berechtigung.

Lösung : Fehler aufgrund einer Berechtigungsverweigerung sind häufig auf fehlende Schnittstellen zurückzuführen. Für die 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 ein Fehler aufgrund einer verweigerten Berechtigung 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 'plugs'
```

Dieses Beispiel zeigt, dass das Erstellen der `/home/ubuntu/my-file.txt` Datei den Berechtigungsfehler verursacht hat. Es schlägt auch vor, dass Sie home zu hinzufügen`plugs`. Diese Argumentation ist jedoch nicht anwendbar. Die - `home-for-greengrassd` und -`home-for-hooks`Plugs erhalten nur Lesezugriff.

Weitere Informationen finden Sie unter Snappy-Debug Snap in der Snap-Dokumentation.

Fehler: kann die folgenden Aufgaben nicht ausführen: - Ausführen des Servicebefehls „start“ für Services ["greengrassd"] von snap „aws-iot-greengrass“ ([start snap.aws-iot-greengrass.greengrassd.service] ist mit dem Beendigungsstatus 1 fehlgeschlagen: Auftrag für snap.aws-iot-greengrass.greengrassd.service fehlgeschlagen, da der Steuerungsprozess mit dem Fehlercode beendet wurde. Weitere Informationen finden Sie unter „Systemctl status snap.aws-iot-greengrass.greengrassd.service“ und „journalctl -xe“.)

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

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

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

Beispielantwort:

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

Diese Beispiele zeigen, dass die `config.json` Datei nicht gefunden AWS IoT Greengrass hat. Sie können die Konfigurations- und Zertifikatsdateien überprüfen.

/var/snap/aws-iot-greengrass/current/ggc-write-directory//packages/1.11.5/rootfs/Merged ist 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 bei der 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 update snapd ausgeführt haben.

Lösung : Folgen Sie den Schritten 6 bis 8 in Installieren und Konfigurieren des AWS IoT Greengrass Snap, um das AWS IoT Greengrass Zertifikat und die Konfigurationsdateien zum AWS IoT Greengrass Snap hinzuzufügen.

Archivieren einer AWS IoT Greengrass Core-Software-Installation

Wenn Sie ein Upgrade auf eine neue Version der AWS IoT Greengrass Core-Software ausführen, 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 setzt voraus, 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 Platzhalter *`os-architecture`* und *`version`* 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.

Konfigurieren des AWS IoT Greengrass Core

Ein -AWS IoT GreengrassKern ist ein -AWS IoTObjekt (Gerät), das als Hub oder Gateway in Edge-Umgebungen fungiert. Wie andere AWS IoT-Geräte ist ein Core in der Registrierung vorhanden, verfügt über einen Geräteschatten und verwendet ein Gerätezertifikat, um sich gegenüber AWS IoT Core und AWS IoT Greengrass zu autorisieren. 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 folgende Funktionalität:

- Bereitstellung und lokale Ausführung von Connectors und Lambda-Funktionen.
- Verarbeiten Sie Datenströme lokal mit automatischen Exporten in die AWS Cloud.
- MQTT-Nachrichten über das lokale Netzwerk zwischen Geräten, Connectors und Lambda-Funktionen mithilfe verwalteter Abonnements.
- MQTT-Nachrichten zwischen AWS IoT und Geräten, Connectors und Lambda-Funktionen mithilfe verwalteter Abonnements.
- Sichere Verbindungen zwischen Geräten und mithilfe der Geräteauthentifizierung und AWS Cloud-autorisierung.
- Lokale Shadow-Synchronisierung von Geräten. Schatten können so konfiguriert werden, dass sie mit dem synchronisiert werdenAWS 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 Softwareupdates over-the-air (OTA) von benutzerdefinierten Lambda-Funktionen.
- Sichere, verschlüsselte Speicherung lokaler Secrets und kontrollierter Zugriff durch Konnektoren und Lambda-Funktionen.

AWS IoT Greengrass Core-Konfigurationsdatei

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, unter 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 auf dem Core-Gerät in einem funktionierenden Zustand 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 über die AWS IoT Greengrass Konsole 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	Beschreibung	Hinweise
`caPath`	Der Pfad zum AWS IoT-Root-CA, relativ zum Verzeichn	Für Abwärtskompatibilität mit Versionen vor 1.7.0. Diese

Feld	Beschreibung	Hinweise
	is */greengrass-root /* `certs`.	Eigenschaft wird ignoriert , wenn das `crypto` Objekt vorhanden ist. **Note** Stellen Sie sicher, dass Ihre Endpunkte Ihrem Zertifikattyp entsprechen.
`certPath`	Der Pfad zum Core-Gerä tezertifikat, relativ zum Verzeichnis */greengras s-root* `/certs`.	Für 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 */greengras s-root* `/certs`.	Für Abwärtskompatibilität mit Versionen vor 1.7.0. Diese Eigenschaft wird ignoriert , wenn das `crypto` Objekt vorhanden ist.
`thingArn`	Der Amazon-Ressourcenn ame (ARN) des AWS IoT Objekts, das das AWS IoT Greengrass Core-Gerät darstellt.	Suchen Sie den ARN für Ihren Core in der AWS IoT GreengrassKonsole unter Cores oder durch Ausführen des `aws greengrass get-core-definition-version` CLI-Befehls .

Feld	Beschreibung	Hinweise
`iotHost`	Ihr AWS IoT-Endpunkt.	Suchen Sie den Endpunkt in der AWS IoTKonsole unter Einstellungen oder durch Ausführen des `aws iot describe-endpoint --endpoint-type iot:Data-ATS` CLI-Befehls . 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 Ihrem entsprechenAWS-Region.

Feld	Beschreibung	Hinweise
`ggHost`	Ihr AWS IoT Greengrass-Endpunkt.	Das ist Ihr `iotHost`-Endpunkt mit dem durch greengrass ersetzten Host-Präfix (z. B. `greengrass-ats.iot.`*`region`*`.amazonaws.com`). Verwenden Sie dasselbe AWS-Region wie `iotHost`. **Note** Stellen Sie sicher, dass Ihre Endpunkte Ihrem Zertifikatstyp entsprechen. Stellen Sie sicher, dass Ihre Endpunkte Ihrem entsprechenAWS-Region.
`iotMqttPort`	Optional. Die Portnummer für die MQTT-Kommunikation mit 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	Beschreibung	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 Proxy-Server kann HTTP oder HTTPS sein. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy.

Feld	Beschreibung	Hinweise
`mqttOperationTimeout`	Optional. Die Zeit (in Sekunden), für die es dem Greengrass-Core möglich ist, einen Veröffent lichungs-, Abonnement- oder Abmeldevorgang in MQTT-Verbindungen zu AWS IoT Core abzuschließen.	Der Standardwert ist 5. Der Mindestwert ist 5.
`ggDaemonPort`	Optional. Die Greengrass Core IPC-Portnummer.	Diese Eigenschaft ist in AWS IoT Greengrass v1.11.0 oder höher verfügbar. Gültige Werte liegen zwischen 1024 und 65535. Der Standardwert ist 8000.
`systemComponentAut hTimeout`	Optional. Die Zeit (in Millisekunden), für die die Greengrass Core IPC die Authentifizierung abschließen kann.	Diese Eigenschaft ist in AWS IoT Greengrass v1.11.0 oder höher verfügbar. Gültige Werte liegen zwischen 500 und 5000. Der Standardwert ist 5000.

runtime

Feld	Beschreibung	Hinweise
`maxWorkItemAnzahl`	Optional. Die maximale Anzahl von Arbeitsel ementen, die der Greengras s-Daemon gleichzei tig verarbeiten kann. Arbeitselemente, die diesen	Der Standardwert lautet 1024. Der Maximalwert wird durch Ihre Gerätehardware begrenzt. Wenn Sie diesen Wert erhöhen, erhöht sich der

Feld	Beschreibung	Hinweise
	Grenzwert überschreiten, werden ignoriert. Die Arbeitselementwart eschlange wird von Systemkomponenten, benutzerdefinierten Lambda-Funktionen und Connectors gemeinsam genutzt.	Speicher, den AWS IoT Greengrass verwendet. Sie können diesen Wert erhöhen, wenn Sie erwarten, dass Ihr Core einen hohen MQTT-Nachrichtenverkehr empfängt.
`maxConcurrentLimit`	Optional. Die maximale Anzahl von gleichzeitigen, nicht angehefteten Lambda-Workern, die der Greengras s-Daemon haben kann. Sie können eine andere Ganzzahl angeben, um diesen Parameter zu überschreiben.	Der Standardwert ist 25. Der Mindestwert wird durch definiert`lruSize`.
`lruSize`	Optional. Defines the minimum value for `maxConcurrentLimit` .	The default value is 25.
`mountAllBlockGeräte`	Optional. Enables AWS IoT Greengrass to use bind mounts to mount all block devices into a container after setting up the OverlayFS.	Diese Eigenschaft ist in AWS IoT Greengrass v1.11.0 oder höher verfügbar. Gültige Werte sind `yes` und no. Der Standardwert ist no. Setzen Sie diesen Wert auf `yes`, wenn sich Ihr `/usr` Verzeichnis nicht unter der `/` Hierarchie befindet.

Feld	Beschreibung	Hinweise
`postStartHealthCheckTimeout`	Optional. The time (in milliseconds) after starting that the Greengrass daemon waits for the health check to finish.	The default timeout is 30 seconds (30000 ms).
`cgroup`		
`useSystemd`	Indicates whether your device uses `systemd`.	Valid values are `Ja` or `Nein`. Run the `check_ggc_dependencies` script in Modul 1 to see if your device uses `systemd`.

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	Beschreibung	Hinweise
`caPath`	Der absolute Pfad zur AWS IoT Root CA.	Muss ein Datei-URI im folgenden Format sein: `file:///absolute/path/to/file` . **Note** Stellen Sie sicher, dass Ihre Endpunkte

Feld	Beschreibung	Hinweise
		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-Agenten mit Hardwaresicherheit verwenden. Weitere Informationen finden Sie unter the section called “OTAUpdates konfigurieren”.
`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 für das Modul verwendet wird.	Muss über ausreichende Berechtigungen verfügen, um C_Sign mit den konfigurierten privaten Schlüsseln auszuführen.
`principals`		
`IoTCertificate`	The certificate and private key that the core uses to make requests to AWS IoT.	

Feld	Beschreibung	Hinweise
`IoTCertificate .privateKeyPath`	Der Pfad zum privaten Core-Schlüssel.	Muss für den Dateisystemspeicher ein Datei-URI im folgenden Format sein: `file:///absolute/path/to/file` . 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: `file:///absolute/path/to/file` .
`MQTTServerCertificate`	Optional. Der private Schlüssel, den der Kern in Kombination mit dem Zertifikat verwendet, um als MQTT-Server oder Gateway zu fungieren.	

Feld	Beschreibung	Hinweise
`MQTT ServerCertificate .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: `file:///`*`absolute/path/to/file`* . Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt. Wenn diese Eigenschaft weggelassen wird, rotiert AWS IoT Greengrass den Schlüssel basierend auf Ihren Rotationseinstellungen. Sofern angegeben, ist der Kunde für das Rotieren des Schlüssels verantwortlich.
`SecretsManager`	The private key that secures the data key used for encryption. For more information, see Bereitstellen von Secrets für den Core.	

Feld	Beschreibung	Hinweise
`SecretsManager .privateKeyPath`	Der Pfad zum privaten Schlüssel des lokalen Secrets Managers.	Nur ein RSA-Schlüssel wird unterstützt. Muss für den Dateisyst emspeicher ein Datei-URI im folgenden Format sein: `file:///`*`absolute/path/to/file`* . Muss für einen HSM-Speic her ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbez eichnung angibt. Der private Schlüssel muss erstellt werden, indem Sie den PKCS#1 v1.5-Padding-Mechanismus verwenden.

Die folgenden Konfigurationseigenschaften werden ebenfalls unterstützt:

Feld	Beschreibung	Hinweise
`mqttMaxConnectionR etryInterval`	Optional. Das Maximalin tervall (in Sekunden) zwischen MQTT-Verb indungsversuchen, 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 `true` oder `false`. Weitere Informati onen finden Sie unter OTA-Updates der AWS IoT Greengrass Core-Software.

Feld	Beschreibung	Hinweise
`writeDirectory`	Optional. Das Schreibverzeichnis, in dem alle Lese-/Schreibressourcen AWS IoT Greengrass erstellt.	Weitere Informationen finden Sie unter Konfigurieren eines Schreibverzeichnisses für AWS IoT Greengrass.
pidFileDirectory	Optional. 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 GreengrassCore-Software befinden sich in der Phase mit verlängerter Lebensdauer. Diese Informationen dienen lediglich Referenzzwecken.

CCPC 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	Beschreibung	Hinweise
`caPath`	Der Pfad zum AWS IoT-Root-CA, relativ zum Verzeichnis */greengrass-root* `/certs`.	Für 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 Zertifikattyp entsprechen.
`certPath`	Der Pfad zum Core-Gerätezertifikat, relativ zum Verzeichnis */greengrass-root* `/certs`.	Für 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	Für Abwärtskompatibilität mit Versionen vor 1.7.0.

Feld	Beschreibung	Hinweise
	Verzeichnis */greengrass-root* /certs.	Diese Eigenschaft wird ignoriert, wenn das `crypto` Objekt vorhanden ist.
`thingArn`	Der Amazon-Ressourcenname (ARN) des AWS IoT Objekts, das das AWS IoT Greengrass Core-Gerät darstellt.	Suchen Sie den ARN für Ihren Core in der AWS IoT GreengrassKonsole unter Cores oder durch Ausführen des `aws greengrass get-core-definition-version` CLI-Befehls .

Feld	Beschreibung	Hinweise
`iotHost`	Ihr AWS IoT-Endpunkt.	Suchen Sie den Endpunkt in der AWS IoTKonsole unter Einstellungen oder durch Ausführen des `aws iot describe-endpoint --endpoint-type iot:Data-ATS` CLI-Befehls . 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 Ihrem entsprechenAWS-Region.

Feld	Beschreibung	Hinweise
`ggHost`	Ihr AWS IoT Greengrass-Endpunkt.	Das ist Ihr `iotHost`-Endpunkt mit dem durch greengrass ersetzten Host-Präfix (z. B. `greengrass-ats.iot.`*`region`*`.amazonaws.com`). Verwenden Sie dasselbe AWS-Region wie `iotHost`. **Note** Stellen Sie sicher, dass Ihre Endpunkte Ihrem Zertifikatstyp entsprechen. Stellen Sie sicher, dass Ihre Endpunkte Ihrem entsprechenAWS-Region.
`iotMqttPort`	Optional. Die Portnummer für die MQTT-Kommunikation mit 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	Beschreibung	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 Proxy-Server kann HTTP oder HTTPS sein. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy.

Feld	Beschreibung	Hinweise
`mqttOperationTimeout`	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.

runtime

Feld	Beschreibung	Hinweise
`maxWorkItemAnzahl`	Optional. Die maximale Anzahl von Arbeitselementen, die der Greengrass-Daemon gleichzeitig verarbeiten kann. Arbeitselemente, die diesen Grenzwert überschreiten, werden ignoriert. Die Arbeitselementwartschlange wird von Systemkomponenten, benutzerdefinierten Lambda-Funktionen und Connectors gemeinsam genutzt.	Der Standardwert lautet 1024. Der Maximalwert wird durch Ihre Gerätehardware begrenzt. Wenn Sie diesen Wert erhöhen, erhöht sich der Speicher, den AWS IoT Greengrass verwendet. Sie können diesen Wert erhöhen, wenn Sie erwarten, dass Ihr Core einen hohen MQTT-Nachrichtenverkehr empfängt.
`maxConcurrentLimit`	Optional. Die maximale Anzahl von gleichzeitigen, nicht angehefteten Lambda-Workern, die der Greengras	Der Standardwert ist 25. Der Mindestwert wird durch definiert`lruSize`.

Feld	Beschreibung	Hinweise
	s-Daemon haben kann. Sie können eine andere Ganzzahl angeben, um diesen Parameter zu überschreiben.	
`lruSize`	Optional. Defines the minimum value for `maxConcurrentLimit` .	The default value is 25.
`postStartHealthCheckTimeout`	Optional. The time (in milliseconds) after starting that the Greengrass daemon waits for the health check to finish.	The default timeout is 30 seconds (30000 ms).
`cgroup`		
`useSystemd`	Indicates whether your device uses `systemd`.	Valid values are `Ja` or `Nein`. Run the `check_ggc_dependencies` script in Modul 1 to see if your device uses `systemd`.

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	Beschreibung	Hinweise
`caPath`	Der absolute Pfad zur AWS IoT Root CA.	Muss ein Datei-URI im folgenden Format sein: `file:///absolute/path/to/file` . **Note** Stellen Sie sicher, dass Ihre Endpunkte Ihrem Zertifikattyp 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-Agenten mit Hardwaresicherheit verwenden. Weitere Informationen finden Sie unter the section called “OTAUpdates konfigurieren”.
`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	Beschreibung	Hinweise
`slotUserPin`	Die Benutzer-PIN, die zur Authentifizierung des Greengrass-Kerns für das Modul verwendet wird.	Muss über ausreichende Berechtigungen verfügen, um C_Sign mit den konfiguri erten privaten Schlüsseln auszuführen.
`principals`		
`IoTCertificate`	The certificate and private key that the core uses to make requests to AWS IoT.	
`IoTCertif icate .privateK eyPath`	Der Pfad zum privaten Core-Schlüssel.	Muss für den Dateisyst emspeicher ein Datei-URI im folgenden Format sein: `file:///`*`absolute/path/to/file`* . Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt.
`IoTCertif icate .certific atePath`	Der absolute Pfad zum Core-Gerätezertifikat.	Muss ein Datei-URI im folgenden Format sein: `file:///`*`absolute/path/to/file`* .
`MQTTServerCertific ate`	Optional. Der private Schlüssel, den der Kern in Kombinati on mit dem Zertifikat verwendet, um als MQTT-Server oder Gateway zu fungieren.	

Feld	Beschreibung	Hinweise
`MQTT ServerCertificate .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 Dateisyst emspeicher ein Datei-URI im folgenden Format sein: `file:///`*`absolute/path/to/file`* . Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt. Wenn diese Eigenschaft weggelassen wird, rotiert AWS IoT Greengrass den Schlüssel basierend auf Ihren Rotationseinstellu ngen. Sofern angegeben, ist der Kunde für das Rotieren des Schlüssels verantwor tlich.
`SecretsManager`	The private key that secures the data key used for encryptio n. For more information, see Bereitstellen von Secrets für den Core.	

Feld	Beschreibung	Hinweise
`SecretsManager .privateKeyPath`	Der Pfad zum privaten Schlüssel des lokalen Secrets Managers.	Nur ein RSA-Schlüssel wird unterstützt. Muss für den Dateisyst emspeicher ein Datei-URI im folgenden Format sein: `file:///`*`absolute/path/to/file`* . 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	Beschreibung	Hinweise
`mqttMaxConnectionR etryInterval`	Optional. Das Maximalin tervall (in Sekunden) zwischen MQTT-Verb indungsversuchen, 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 benutzerd efinierten Code ausführen muss.	Gültige Werte sind `true` oder `false`. Weitere Informationen finden Sie unter OTA-Updates der AWS IoT Greengrass Core-Software.

Feld	Beschreibung	Hinweise
`writeDirectory`	Optional. Das Schreibverzeichnis, in dem alle Lese-/Schreibressourcen AWS IoT Greengrass erstellt.	Weitere Informationen finden Sie unter Konfigurieren eines Schreibverzeichnisses für AWS IoT Greengrass.

CCPC 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	Beschreibung	Hinweise
`caPath`	Der Pfad zum AWS IoT-Root-CA, relativ zum Verzeichnis */greengrass-root* `/certs`.	Für 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 */greengrass-root* `/certs`.	Für 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 */greengrass-root* `/certs`.	Für 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 Objekts, das das AWS IoT Greengrass Core-Gerät darstellt.	Suchen Sie den ARN für Ihren Core in der AWS IoT GreengrassKonsole unter Cores oder durch Ausführen des `aws greengrass get-core-definition-version` CLI-Befehls .

Feld	Beschreibung	Hinweise
`iotHost`	Ihr AWS IoT-Endpunkt.	Suchen Sie den Endpunkt in der AWS IoTKonsole unter Einstellungen oder durch Ausführen des `aws iot describe-endpoint --endpoint-type iot:Data-ATS` CLI-Befehls . 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 Ihrem entsprechenAWS-Region.

Feld	Beschreibung	Hinweise
`ggHost`	Ihr AWS IoT Greengrass-Endpunkt.	Das ist Ihr `iotHost`-Endpunkt mit dem durch greengrass ersetzten Host-Präfix (z. B. `greengrass-ats.iot.` *`region`*`.amazonaws.com`). Verwenden Sie dasselbe AWS-Region wie `iotHost`. **Note** Stellen Sie sicher, dass Ihre Endpunkte Ihrem Zertifikatstyp entsprechen. Stellen Sie sicher, dass Ihre Endpunkte Ihrem entsprechenAWS-Region.
`iotMqttPort`	Optional. Die Portnummer für die MQTT-Kommunikation mit 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	Beschreibung	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 Proxy-Server kann HTTP oder HTTPS sein. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy.

runtime

Feld	Beschreibung	Hinweise
`maxConcurrentLimit`	Optional. Die maximale Anzahl von gleichzeitigen, nicht angehefteten Lambda-Workern, die der Greengrass-Daemon haben kann. Sie können eine andere Ganzzahl angeben, um diesen Parameter zu überschreiben.	Der Standardwert ist 25. Der Mindestwert wird durch definiert`lruSize`.
`lruSize`	Optional. Defines the minimum value for `maxConcurrentLimit` .	The default value is 25.
`postStartHealthCheckTimeout`	Optional. The time (in milliseconds) after starting that the Greengrass daemon waits for the health check to finish.	The default timeout is 30 seconds (30000 ms).
`cgroup`		
`useSystemd`	Indicates whether your device uses `systemd`.	Valid values are `Ja` or `Nein`. Run the `check_ggc_dependencies` script in Modul 1 to see if your device uses `systemd`.

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	Beschreibung	Hinweise
`caPath`	Der absolute Pfad zur AWS IoT Root CA.	Muss ein Datei-URI im folgenden Format sein: `file:///absolute/path/to/file` . **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-Agenten mit Hardwaresicherheit verwenden. Weitere Informationen finden Sie unter the section called “OTAUpdates konfigurieren”.
`P11Provider`	Der absolute Pfad zur libdl-ladbaren Bibliothek der PKCS#11-Implementierung.	Muss ein Pfad zu einer Datei im Dateisystem sein.

Feld	Beschreibung	Hinweise
`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 für das Modul verwendet wird.	Muss über ausreichende Berechtigungen verfügen, um C_Sign mit den konfigurierten privaten Schlüsseln auszuführen.
`principals`		
`IoTCertificate`	The certificate and private key that the core uses to make requests to AWS IoT.	
`IoTCertificate .privateKeyPath`	Der Pfad zum privaten Core-Schlüssel.	Muss für den Dateisystemspeicher ein Datei-URI im folgenden Format sein: `file:///`*`absolute/path/to/file`* . 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: `file:///`*`absolute/path/to/file`* .
`MQTTServerCertificate`	Optional. Der private Schlüssel, den der Kern in Kombination mit dem Zertifikat verwendet, um als MQTT-Server oder Gateway zu fungieren.	

Feld	Beschreibung	Hinweise
`MQTT ServerCertificate .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: `file:///`*`absolute/path/to/file`* . Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt. Wenn diese Eigenschaft weggelassen wird, rotiert AWS IoT Greengrass den Schlüssel basierend auf Ihren Rotationseinstellungen. Sofern angegeben, ist der Kunde für das Rotieren des Schlüssels verantwortlich.
`SecretsManager`	The private key that secures the data key used for encryption. For more information, see Bereitstellen von Secrets für den Core.	

Feld	Beschreibung	Hinweise
`SecretsManager .privateKeyPath`	Der Pfad zum privaten Schlüssel des lokalen Secrets Managers.	Nur ein RSA-Schlüssel wird unterstützt. Muss für den Dateisyst emspeicher ein Datei-URI im folgenden Format sein: `file:///`*`absolute/path/to/file`* . 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	Beschreibung	Hinweise
`mqttMaxConnectionR etryInterval`	Optional. Das Maximalin tervall (in Sekunden) zwischen MQTT-Verb indungsversuchen, 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 benutzerd efinierten Code ausführen muss.	Gültige Werte sind `true` oder `false`. Weitere Informationen finden Sie unter OTA-Updates der AWS IoT Greengrass Core-Software.

Feld	Beschreibung	Hinweise
`writeDirectory`	Optional. Das Schreibverzeichnis, in dem alle Lese-/Schreibressourcen AWS IoT Greengrass erstellt.	Weitere Informationen finden Sie unter Konfigurieren eines Schreibverzeichnisses für AWS IoT Greengrass.

CCPC 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	Beschreibung	Hinweise
`caPath`	Der Pfad zum AWS IoT-Root-CA, relativ zum Verzeichnis */greengrass-root* `/certs`.	Für 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 */greengrass-root* `/certs`.	Für 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 */greengrass-root* `/certs`.	Für 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 Objekts, das das AWS IoT Greengrass Core-Gerät darstellt.	Suchen Sie den ARN für Ihren Core in der AWS IoT GreengrassKonsole unter Cores oder durch Ausführen des `aws greengrass get-core-definition-version` CLI-Befehls .

Feld	Beschreibung	Hinweise
`iotHost`	Ihr AWS IoT-Endpunkt.	Suchen Sie den Endpunkt in der AWS IoTKonsole unter Einstellungen oder durch Ausführen des `aws iot describe-endpoint --endpoint-type iot:Data-ATS` CLI-Befehls . 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 Ihrem entsprechenAWS-Region.

Feld	Beschreibung	Hinweise
`ggHost`	Ihr AWS IoT Greengrass-Endpunkt.	Das ist Ihr `iotHost`-Endpunkt mit dem durch greengrass ersetzten Host-Präfix (z. B. `greengrass-ats.iot.`*`region`*`.amazonaws.com`). Verwenden Sie dasselbe AWS-Region wie `iotHost`. **Note** Stellen Sie sicher, dass Ihre Endpunkte Ihrem Zertifikatstyp entsprechen. Stellen Sie sicher, dass Ihre Endpunkte Ihrem entsprechenAWS-Region.
`iotMqttPort`	Optional. Die Portnummer für die MQTT-Kommunikation mit 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	Beschreibung	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 Proxy-Server kann HTTP oder HTTPS sein. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy.

runtime

Feld	Beschreibung	Hinweise
`cgroup`		

Feld	Beschreibung	Hinweise
`useSystemd`	Indicates whether your device uses `systemd`.	Valid values are `Ja` or `Nein`. Run the `check_ggc_dependencies` script in Modul 1 to see if your device uses `systemd`.

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	Beschreibung	Hinweise
`caPath`	Der absolute Pfad zur AWS IoT Root CA.	Muss ein Datei-URI im folgenden Format sein: `file:///absolute/path/to/file` . **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	Beschreibung	Hinweise
	nterstützung unter OpenSSL zu aktivieren.	Diese Eigenschaft ist erforderlich, wenn Sie den Greengrass-OTA-Update-Agenten mit Hardwares icherheit verwenden. Weitere Informationen finden Sie unter the section called “OTAUpdates konfigurieren”.
`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 entsprech en.
`slotUserPin`	Die Benutzer-PIN, die zur Authentifizierung des Greengrass-Kerns für das Modul verwendet wird.	Muss über ausreichende Berechtigungen verfügen, um C_Sign mit den konfiguri erten privaten Schlüsseln auszuführen.
`principals`		
`IoTCertificate`	The certificate and private key that the core uses to make requests to AWS IoT.	

Feld	Beschreibung	Hinweise
`IoTCertificate .privateKeyPath`	Der Pfad zum privaten Core-Schlüssel.	Muss für den Dateisyst emspeicher ein Datei-URI im folgenden Format sein: `file:///`*`absolute/path/to/file`* . 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: `file:///`*`absolute/path/to/file`* .
`MQTTServerCertificate`	Optional. Der private Schlüssel, den der Kern in Kombination mit dem Zertifikat verwendet, um als MQTT-Server oder Gateway zu fungieren.	

Feld	Beschreibung	Hinweise
`MQTT ServerCertificate .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: `file:///`*`absolute/path/to/file`* . Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt. Wenn diese Eigenschaft weggelassen wird, rotiert AWS IoT Greengrass den Schlüssel basierend auf Ihren Rotationseinstellungen. Sofern angegeben, ist der Kunde für das Rotieren des Schlüssels verantwortlich.
`SecretsManager`	The private key that secures the data key used for encryption. For more information, see Bereitstellen von Secrets für den Core.	

Feld	Beschreibung	Hinweise
`SecretsManager .privateKeyPath`	Der Pfad zum privaten Schlüssel des lokalen Secrets Managers.	Nur ein RSA-Schlüssel wird unterstützt. Muss für den Dateisyst emspeicher ein Datei-URI im folgenden Format sein: `file:///`*`absolute/path/to/file`* . 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	Beschreibung	Hinweise
`mqttMaxConnectionR etryInterval`	Optional. Das Maximalin tervall (in Sekunden) zwischen MQTT-Verb indungsversuchen, 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 benutzerd efinierten Code ausführen muss.	Gültige Werte sind `true` oder `false`. Weitere Informationen finden Sie unter OTA-Updates der AWS IoT Greengrass Core-Software.

Feld	Beschreibung	Hinweise
`writeDirectory`	Optional. Das Schreibverzeichnis, in dem alle Lese-/Schreibressourcen AWS IoT Greengrass erstellt.	Weitere Informationen finden Sie unter Konfigurieren eines Schreibverzeichnisses 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	Beschreibung	Hinweise
`caPath`	Der Pfad zum AWS IoT-Root-CA, relativ zum Verzeichnis */greengrass-root* `/certs`.	Für 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 */greengrass-root* `/certs`.	Für 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 */greengrass-root* `/certs`.	Für 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 Objekts, das das AWS IoT Greengrass Core-Gerät darstellt.	Suchen Sie den ARN für Ihren Core in der AWS IoT GreengrassKonsole unter Cores oder durch Ausführen des `aws greengrass get-core-definition-version` CLI-Befehls .

Feld	Beschreibung	Hinweise
`iotHost`	Ihr AWS IoT-Endpunkt.	Suchen Sie den Endpunkt in der AWS IoTKonsole unter Einstellungen oder durch Ausführen des `aws iot describe-endpoint --endpoint-type iot:Data-ATS` CLI-Befehls . 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 Zertifikattyp entsprechen. Stellen Sie sicher, dass Ihre Endpunkte Ihrem entsprechenAWS-Region.

Feld	Beschreibung	Hinweise
`ggHost`	Ihr AWS IoT Greengrass-Endpunkt.	Das ist Ihr `iotHost`-Endpunkt mit dem durch greengrass ersetzten Host-Präfix (z. B. `greengrass-ats.iot.`*`region`*`.amazonaws.com`). Verwenden Sie dasselbe AWS-Region wie `iotHost`. **Note** Stellen Sie sicher, dass Ihre Endpunkte Ihrem Zertifikatstyp entsprechen. Stellen Sie sicher, dass Ihre Endpunkte Ihrem entsprechenAWS-Region.
`iotMqttPort`	Optional. Die Portnummer für die MQTT-Kommunikation mit 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 `KeepAlive` in Sekunden.	Der gültige Bereich liegt zwischen 30 und 1200 Sekunden. Der Standardwert ist `600`.

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

runtime

Feld	Beschreibung	Hinweise
`cgroup`		
`useSystemd`	Indicates whether your device uses `systemd`.	Valid values are `Ja` or `Nein`. Run the `check_ggc_dependencies` script in Modul 1 to see if your device uses `systemd`.

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 finden 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	Beschreibung	Hinweise
`caPath`	Der absolute Pfad zur AWS IoT Root CA.	Muss ein Datei-URI im folgenden Format sein:

Feld	Beschreibung	Hinweise
		`file:///absolute/path/to/file` . **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-Agenten mit Hardwaresicherheit verwenden. Weitere Informationen finden Sie unter the section called “OTAUpdates konfigurieren”.
`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	Beschreibung	Hinweise
`slotUserPin`	Die Benutzer-PIN, die zur Authentifizierung des Greengrass-Kerns für das Modul verwendet wird.	Muss über ausreichende Berechtigungen verfügen, um C_Sign mit den konfiguri erten privaten Schlüsseln auszuführen.
`principals`		
`IoTCertificate`	The certificate and private key that the core uses to make requests to AWS IoT.	
`IoTCertif icate .privateK eyPath`	Der Pfad zum privaten Core-Schlüssel.	Muss für den Dateisyst emspeicher ein Datei-URI im folgenden Format sein: `file:///`*`absolute/path/to/file`* . Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt.
`IoTCertif icate .certific atePath`	Der absolute Pfad zum Core-Gerätezertifikat.	Muss ein Datei-URI im folgenden Format sein: `file:///`*`absolute/path/to/file`* .
`MQTTServerCertific ate`	Optional. Der private Schlüssel, den der Kern in Kombinati on mit dem Zertifikat verwendet, um als MQTT-Server oder Gateway zu fungieren.	

Feld	Beschreibung	Hinweise
`MQTT ServerCertificate .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 Dateisyst emspeicher ein Datei-URI im folgenden Format sein: `file:///`*`absolute/path/to/file`* . Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt. Wenn diese Eigenschaft weggelassen wird, rotiert AWS IoT Greengrass den Schlüssel basierend auf Ihren Rotationseinstellu ngen. Sofern angegeben, ist der Kunde für das Rotieren des Schlüssels verantwor tlich.
`SecretsManager`	The private key that secures the data key used for encryptio n. For more information, see Bereitstellen von Secrets für den Core.	

Feld	Beschreibung	Hinweise
`SecretsManager .privateKeyPath`	Der Pfad zum privaten Schlüssel des lokalen Secrets Managers.	Nur ein RSA-Schlüssel wird unterstützt. Muss für den Dateisyst emspeicher ein Datei-URI im folgenden Format sein: `file:///`*`absolute/path/to/file`* . 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	Beschreibung	Hinweise
`mqttMaxConnectionR etryInterval`	Optional. Das Maximalin tervall (in Sekunden) zwischen MQTT-Verb indungsversuchen, 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 benutzerd efinierten Code ausführen muss.	Gültige Werte sind `true` oder `false`. Weitere Informationen finden Sie unter OTA-Updates der AWS IoT Greengrass Core-Software.

Feld	Beschreibung	Hinweise
`writeDirectory`	Optional. Das Schreibverzeichnis, in dem alle Lese-/Schreibressourcen AWS IoT Greengrass erstellt.	Weitere Informationen finden Sie unter Konfigurieren eines Schreibverzeichnisses 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 Standardgruppenerstellung von der AWS IoT Greengrass Konsole aus verwenden, wird die `config.json` Datei auf dem Core-Gerät in einem funktionierenden Zustand bereitgestellt, der die Standardkonfiguration angibt.

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

Feld	Beschreibung	Hinweise
`caPath`	Der Pfad zum AWS IoT Root CA, relativ zum Verzeichnis */greengrass-root* `/certs`.	Speichern Sie die Datei unter */greengrass-root*`/certs`.
`certPath`	Der Pfad zum AWS IoT Greengrass Kernzertifikat relativ zum */greengrass-root* `/certs` Verzeichnis.	Speichern Sie die Datei unter */greengrass-root*`/certs`.
`keyPath`	Der Pfad zum privaten AWS IoT Greengrass Kernschlüssel relativ zum */greengrass-root* `/certs` Verzeichnis.	Speichern Sie die Datei unter */greengrass-root*`/certs`.
`thingArn`	Der Amazon-Ressourcenname (ARN) des AWS IoT Objekts, das das AWS IoT Greengrass Core-Gerät darstellt.	Suchen Sie den ARN für Ihren Core in der AWS IoT GreengrassKonsole unter Cores oder durch Ausführen des `aws greengrass get-core-definition-version` CLI-Befehls .
`iotHost`	Ihr AWS IoT-Endpunkt.	Suchen Sie dies in der AWS IoTKonsole unter Einstellungen oder durch Ausführen des `aws iot describe-endpoint` CLI-Befehls .
`ggHost`	Ihr AWS IoT Greengrass-Endpunkt.	Für diesen Wert wird das `greengrass.iot.`*region*`.amazonaws.com` -Format verwendet

Feld	Beschreibung	Hinweise
		. Verwenden Sie dieselbe Region wie `iotHost`.
`keepAlive`	Der MQTT-Zeitraum `KeepAlive` in Sekunden.	Dies ist ein optionaler Wert. Der Standardwert ist `600`.
`mqttMaxConnectionRetryInterval`	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 `60`.
`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.
`managedRespawn`	Ein optionales Feature für over-the-air (OTA)-Aktualisierungen, dies bedeutet, dass der OTA-Agent vor einem Update benutzerdefinierten Code ausführen muss.	Gültige Werte sind `true` oder `false`. Weitere Informationen finden Sie unter OTA-Updates der AWS IoT Greengrass Core-Software.
`writeDirectory`	Das Schreibverzeichnis, in dem alle Lese-/Schreibressourcen AWS IoT Greengrass erstellt.	Dies ist ein optionaler Wert. Weitere Informationen finden Sie unter Konfigurieren eines Schreibverzeichnisses 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	Beschreibung	Hinweise
`caPath`	Der Pfad zum AWS IoT Root CA, relativ zum Ordner */greengrass-root* `/certs`.	Speichern Sie die Datei im Ordner */greengrass-root*`/certs`.
`certPath`	Der Pfad zum AWS IoT Greengrass Kernzertifikat relativ zum */greengrass-root* `/certs` Ordner.	Speichern Sie die Datei im Ordner */greengrass-root*`/certs`.
`keyPath`	Der Pfad zum privaten AWS IoT Greengrass Kernschlüssel relativ zum */greengrass-root* `/certs` Ordner.	Speichern Sie die Datei im Ordner */greengrass-root*`/certs`.

Feld	Beschreibung	Hinweise
`thingArn`	Der Amazon-Ressourcenname (ARN) des AWS IoT Objekts, das das AWS IoT Greengrass Core-Gerät darstellt.	Suchen Sie den ARN für Ihren Core in der AWS IoT GreengrassKonsole unter Cores oder durch Ausführen des `aws greengrass get-core-definition-version` CLI-Befehls .
`iotHost`	Ihr AWS IoT-Endpunkt.	Suchen Sie dies in der AWS IoT Konsole unter Einstellungen oder durch Ausführen des Befehls `aws iot describe-endpoint` .
`ggHost`	Ihr AWS IoT Greengrass-Endpunkt.	Für diesen Wert wird das `greengrass.iot.` *`region`*`.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	Beschreibung	Hinweise
`managedRespawn`	Ein optionales Feature für over-the-air (OTA)-Aktualisierungen, dies bedeutet, 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	Beschreibung	Hinweise
`caPath`	Der Pfad zum AWS IoT Root CA, relativ zum Ordner	Speichern Sie die Datei im Ordner */greengrass-root*`/certs`.

Feld	Beschreibung	Hinweise
	/greengrass-root /certs.	
certPath	Der Pfad zum AWS IoT Greengrass Kernzertifikat relativ zum */greengrass-root* /certs Ordner.	Speichern Sie die Datei im Ordner */greengrass-root*/certs.
keyPath	Der Pfad zum privaten AWS IoT Greengrass Kernschlüssel relativ zum */greengrass-root* /certs Ordner.	Speichern Sie die Datei im Ordner */greengrass-root*/certs.
thingArn	Der Amazon-Ressourcenname (ARN) des AWS IoT Objekts, das den AWS IoT Greengrass Kern darstellt.	Sie finden diesen Wert in der AWS IoT Greengrass Konsole unter der Definition für Ihr AWS IoT Objekt.
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 greengrass. vorangestelltem .
keepAlive	Der MQTT-Zeitraum KeepAlive in Sekunden.	Dies ist ein optionaler Wert. Der Standardwert ist 600 Sekunden.

Feld	Beschreibung	Hinweise
`useSystemd`	Ein Binär-Marker, wenn Ihr Gerät `systemd` verwendet.	Die Werte sind `yes` oder `no`. Verwenden Sie das Abhängigkeitenskript in Modul 1, um festzustellen, ob Ihr Gerät `systemd` verwendet.
`managedRespawn`	Ein optionales Feature für over-the-air (OTA)-Aktualisierungen, dies bedeutet, 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	Beschreibung	Hinweise
`caPath`	Der Pfad zum AWS IoT Root CA, relativ zum Ordner */greengrass-root* `/certs`.	Speichern Sie die Datei im Ordner */greengrass-root*`/certs`.
`certPath`	Der Pfad zum AWS IoT Greengrass Kernzertifikat relativ zum */greengrass-root* `/certs` Ordner.	Speichern Sie die Datei im Ordner */greengrass-root*`/certs`.
`keyPath`	Der Pfad zum privaten AWS IoT Greengrass Kernschlüssel relativ zum */greengrass-root* `/certs` Ordner.	Speichern Sie die Datei im Ordner */greengrass-root*`/certs`.
`thingArn`	Der Amazon-Ressourcenname (ARN) des AWS IoT Objekts, das den AWS IoT Greengrass Kern darstellt.	Sie finden diesen Wert in der AWS IoT Greengrass Konsole unter der Definition für Ihr AWS IoT Objekt.
`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 `greengrass.` vorangestelltem .
`keepAlive`	Der MQTT-Zeitraum `KeepAlive` in Sekunden.	Dies ist ein optionaler Wert. Der Standardwert ist 600 Sekunden.

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

GGC v1.0

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

```
{
   "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	Beschreibung	Hinweise
`caPath`	Der Pfad zum AWS IoT Root CA, relativ zum Ordner */greengrass-root* `/`	Speichern Sie die Datei im Ordner */greengrass-*

Feld	Beschreibung	Hinweise
	`configuration/certs` .	*`root`*`/configuration/certs` .
`certPath`	Der Pfad zum AWS IoT Greengrass Kernzertifikat relativ zum */greengrass-root* `/configuration/certs` Ordner.	Speichern Sie die Datei im Ordner */greengrass-root*`/configuration/certs` .
`keyPath`	Der Pfad zum privaten AWS IoT Greengrass Kernschlüssel relativ zum */greengrass-root* `/configuration/certs` Ordner.	Speichern Sie die Datei im Ordner */greengrass-root*`/configuration/certs` .
`thingArn`	Der Amazon-Ressourcenname (ARN) des AWS IoT Objekts, das den AWS IoT Greengrass Kern darstellt.	Sie finden diesen Wert in der AWS IoT Greengrass Konsole unter der Definition für Ihr AWS IoT -Objekt.
`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 `greengrass.` vorangestelltem .
`keepAlive`	Der MQTT-Zeitraum `KeepAlive` in Sekunden.	Dies ist ein optionaler Wert. Der Standardwert ist 600 Sekunden.

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

Service-Endpunkte müssen mit dem Zertifikatstyp der Stammzertifizierungsstelle übereinstimmen

Ihre AWS IoT Core- und AWS IoT Greengrass-Endpunkte müssen dem Zertifikattyp des Stamm-CA-Zertifikats auf Ihrem Gerät entsprechen. Wenn die Endpunkte und der Zertifikatstyp nicht übereinstimmen, schlagen die Authentifizierungsversuche zwischen AWS IoT Core und AWS IoT Greengrass fehl. Weitere Informationen finden Sie unter Serverauthentifizierung im AWS IoT-Entwicklerhandbuch.

Wenn Ihr Gerät ein Amazon-Trust-Services-(ATS)-Stammzertifizierungsstellenzertifikat verwendet, was die bevorzugte Methode ist, muss es auch ATS-Endpunkte für Operationen auf Geräteverwaltung und Datenerkennungsebene verwenden. ATS-Endpunkte beinhalten das `ats`-Segment, wie in der folgenden Syntax für den AWS IoT Core-Endpunkt angezeigt.

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

> **Note**
>
> Aus Gründen der Abwärtskompatibilität unterstützt AWS IoT Greengrass derzeit ältere VeriSign Stammzertifizierungsstellenzertifikate und Endpunkte in einigen AWS-Regionen. Wenn Sie ein VeriSign älteres Stammzertifizierungsstellenzertifikat verwenden, empfehlen wir Ihnen, stattdessen einen ATS-Endpunkt zu erstellen und ein ATS-Stammzertifizierungsstellenzertifikat 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 Eigenschaft `iotHost` steht für den AWS IoT Core-Endpunkt. Die Eigenschaft `ggHost` steht für den AWS IoT Greengrass-Endpunkt. Im folgenden Beispiel für einen Ausschnitt legen diese Eigenschaften ATS-Endpunkte fest.

```
{
  "coreThing" : {
    ...
    "iotHost" : "abcde1234uwxyz-ats.iot.us-west-2.amazonaws.com",
    "ggHost" : "greengrass-ats.iot.us-west-2.amazonaws.com",
    ...
  },
```

AWS IoT Core-Endpunkt

Sie erhalten Ihren AWS IoT Core-Endpunkt, indem Sie den CLI-Befehl aws iot describe-endpoint 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
  ```

- Um einen Legacy- VeriSign signierten Endpunkt zurückzugeben, führen Sie Folgendes aus:

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

AWS IoT Greengrass-Endpunkt

Für Ihren AWS IoT Greengrass-Endpunkt wird das Host-Präfix Ihres `iotHost`-Endpunkts durch greengrass ersetzt. Zum Beispiel lautet der signierte ATS-Endpunkt `greengrass-ats.iot.`*`region`*`.amazonaws.com`. Dieser verwendet die selbe 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-Cores kommunizieren mit AWS IoT Core unter Verwendung des MQTT-Messaging-Protokolls mit TLS-Client-Authentifizierung. Standardmäßig verwendet MQTT über TLS den Port

8883. Doch als Sicherheitsmaßnahme 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 solchen Fällen zu ermöglichen, erlaubt AWS IoT Greengrass die folgenden Konfigurationen:

- 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 ermöglicht diese Verbindung mittels der Application Layer Protocol Network (ALPN) TLS-Erweiterung. Wie bei der Standardkonfiguration verwendet MQTT über TLS auf Port 443 eine zertifikatbasierte Client-Authentifizierung.

  Wenn für die Verwendung einer direkten Verbindung mit Port 443 konfiguriert, unterstützt der Core over-the-air (OTA)-Updates für AWS IoT Greengrass Software. Diese Unterstützung erfordert AWS IoT Greengrass Core v1.9.3 oder höher.
- HTTPS-Kommunikation über Port 443. AWS IoT Greengrass sendet HTTPS-Datenverkehr standardmäßig über Port 8443. Sie können ihn jedoch so konfigurieren, dass er Port 443 verwendet.
- 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 Proxy-Konfiguration 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. Häufige Bibliotheken, die von Lambda-Funktionen zum Herstellen von Verbindungen verwendet werden (wie boto3- oder cURL- und Python-`requests`Pakete), verwenden diese Umgebungsvariablen in der Regel standardmäßig. Wenn eine Lambda-Funktion auch dieselben Umgebungsvariablen angibt, überschreibt 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 stoppen:

   ```
   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 stoppen:

   ```
   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 AWS IoT Greengrass, eine Internetverbindung über einen HTTP- oder HTTPS-Netzwerk-Proxy herzustellen.

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

   ```
   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	Beschreibung
`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. • `url`. Die URL des Proxy-Servers im Format `scheme://userinfo@host:port` . • `scheme`. Das Schema. Es muss sich entweder um `http` oder `https` handeln. • `userinfo`. Optional. Der Benutzername und die Passwortinformationen. Die Felder `username` und `password` werden ignoriert, sofern angegeben. • `host`. Der Hostname oder die IP-Adresse des Proxy-Servers. • `port`. Optional. Die Port-Nummer. Wenn nicht angegeben, werden die folgenden Standardwerte verwendet: • `http`: 80 • `https`: 443 • `username`. Optional. Der Benutzername, mit dem die Authentifizierung gegenüber dem Proxy-Server erfolgt. • `password`. Optional. Das Passwort, das zur Authentifizierung am Proxy-Server verwendet wird.

Zulassen von Endpunkten

Die Kommunikation zwischen Greengrass-Geräten und AWS IoT Core 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.

Endpunkt	Port	Beschreibung
`greengrass. region.amazonaws.com`	443	Für Operationen auf Steuerebene für die Gruppenverwaltung
`prefix-ats.iot. region.amazonaws.com` or `prefix.iot.region.amazonaws.com`	MQTT: 8883 oder 443 HTTPS: 8443 oder 443	Für Operationen auf Datenebene für das Geräteman agement, wie z. B. Schattens ynchronisierung Erlauben Sie die Verwendung eines oder beider Endpunkte, je nachdem, ob Ihre Core- und Client-Geräte Amazon Trust

Endpunkt	Port	Beschreibung
		Services (bevorzugte) Root-CA-Zertifikate, Legacy-Root-CA-Zertifikate oder beides verwenden. Weitere Informationen finden Sie unter the section called “Service-Endpunkte müssen mit dem Zertifikattyp übereinstimmen”.

Endpunkt	Port	Beschreibung
`greengrass-ats.iot. region.amazonaws.com` or `greengrass.iot. region.amazonaws.com`	8443 oder 443	Für Geräteerkennungsoperationen Erlauben Sie die Verwendung eines oder beider Endpunkte, je nachdem, ob Ihre Core- und Client-Geräte Amazon Trust Services (bevorzugte) Root-CA-Zertifikate, Legacy-Root-CA-Zertifikate oder beides verwenden. Weitere Informationen finden Sie unter the section called “Service-Endpunkte müssen mit dem Zertifikattyp übereinstimmen”.

Endpunkt	Port	Beschreibung
		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 `ProtocolName`

Endpunkt	Port	Beschreibung
		in der übergeben `ProtocolNameList` . Weitere Informationen finden Sie unter Protokolle im AWS IoT - Entwicklerhandbuch.

Endpunkt	Port	Beschreibung
`*.s3.amazonaws.com`	443	Wird für Bereitstellungsvorgänge und over-the-air Updates verwendet. Dieses Format enthält das Zeichen *, da Endpunktpräfixe intern kontrolliert werden und sich jederzeit ändern können.
`logs.`*`region`*`.amazonaws.com`	443	Erforderlich, wenn die Greengrass-Gruppe so konfiguriert ist, dass Protokolle in CloudWatch geschrieben werden.

Konfigurieren eines Schreibverzeichnisses 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 unter einem einzigen Stammverzeichnis bereitgestellt, in dem AWS IoT Greengrass alle Lese- und Schreiboperationen ausführt. Sie können AWS IoT Greengrass allerdings so konfigurieren, dass ein separater Ordner für alle Schreiboperationen, einschließlich der Erstellung von Verzeichnissen und Dateien, verwendet wird. In diesem Fall verwendet AWS IoT Greengrass zwei Top-Level-Verzeichnisse:

- Das *greengrass-root*-Verzeichnis, das Sie für Lese-/Schreibvorgänge oder optional schreibgeschützt verwenden können. Darin sind die AWS IoT Greengrass Core-Software und andere kritische Komponenten enthalten, die während der Laufzeit unveränderlich bleiben sollen, beispielsweise Zertifikate und `config.json`.
- Das angegebene Schreibverzeichnis. Dies enthält beschreibbare Inhalte 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 AWS IoT 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 `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. Nachdem die Einstellung aktualisiert wurde, verwendet AWS IoT Greengrass das neu angegebene Schreibverzeichnis beim nächsten Start. Allerdings wird kein Inhalt aus dem vorherigen Schreibverzeichnis migriert.

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 System-Lambda-Funktionen 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. Stellen Sie mit Ihrem bevorzugten Mechanismus den Schreibschutz für das *greengrass-root*-Verzeichnis ein.

> **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 gemounteten Verzeichnis anzuwenden, muss das Verzeichnis zuerst entmountet und nach dem Update erneut gemountet 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 kommunizierenAWS IoT Core. Die gesamte Kommunikation erfolgt durch den Core, der die Abonnements verwaltet, die die MQTT-Kommunikation zwischen Entitäten autorisieren.

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

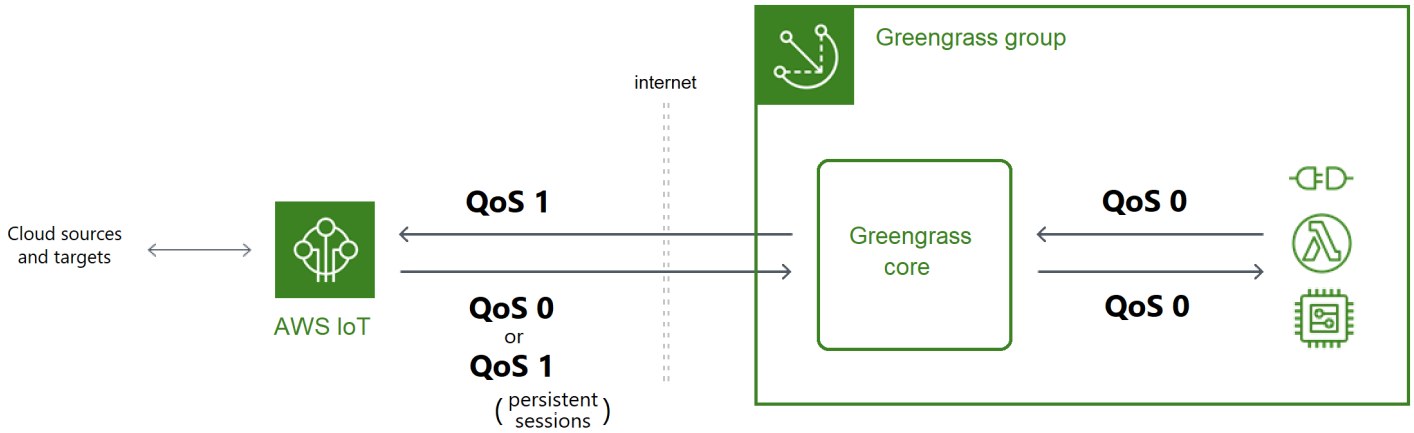
- the section called “Nachrichtenqualität des Service”
- the section called “MQTT-Nachrichtenwarteschlange”

- 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-, Abonnement- und Abmeldevorgänge in MQTT-Verbindungen mit der AWS Cloud”

> **Note**
>
> OPC-UA ist ein Standard für den Informationsaustausch für die industrielle Kommunikation. Um die Unterstützung für OPC-UA auf dem Greengrass-Kern zu implementieren, können Sie den IoT SiteWise -Konnektor verwenden. Der Konnektor sendet industrielle Gerätedaten von OPC-UA-Servern an Objekteigenschaften in AWS IoT SiteWise.

Nachrichtenqualität des Service

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



Weitere Informationen zu MQTT und QoS finden Sie unter Erste Schritte auf der MQTT-Website.

Kommunikation mit der AWS Cloud

- Ausgehende Nachrichten verwenden QoS 1

Der Kern sendet Nachrichten, die für AWS Cloud Ziele bestimmt sind, mit QoS 1. AWS IoT Greengrass verwendet eine MQTT-Nachrichtenwarteschlange, um diese Nachrichten zu verarbeiten. Wenn die Nachrichtenzustellung nicht von bestätigt wirdAWS IoT, wird die Nachricht gespoolt und später erneut versucht. Die Nachricht kann nicht erneut versucht werden, wenn die Warteschlange voll ist. Die Bestätigung der Nachrichtenzustellung kann dazu beitragen, Datenverluste durch intermittierende Konnektivität zu minimieren.

Da ausgehende Nachrichten QoS 1 AWS IoT verwenden, hängt die maximale Rate, mit der der Greengrass-Kern Nachrichten senden kann, von der Latenz zwischen dem Kern und abAWS IoT. Jedes Mal, wenn der Kern eine Nachricht sendet, wartet er, bis die Nachricht AWS IoT bestätigt, bevor er die nächste Nachricht sendet. Wenn die Round-Trip-Zeit zwischen dem Kern und seinem beispielsweise 50 Millisekunden AWS-Region beträgt, kann der Kern bis zu 20 Nachrichten pro Sekunde senden. Berücksichtigen Sie dieses Verhalten, wenn Sie die auswählen, AWS-Region in der Ihr Kern eine Verbindung herstellt. Um IoT-Daten mit hohem Volumen in die aufzunehmenAWS Cloud, können Sie den Stream-Manager verwenden.

Weitere Informationen zur MQTT-Nachrichtenwarteschlange, einschließlich der Konfiguration eines lokalen Speicher-Caches, der Nachrichten für AWS Cloud Ziele beibehalten kann, finden Sie unter the section called “MQTT-Nachrichtenwarteschlange”.

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

  Standardmäßig abonniert der Core Nachrichten aus AWS Cloud Quellen mit QoS 0. 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, einschließlich der Aktivierung des Kerns zum Einrichten einer persistenten Sitzung mit AWS Cloud Zielen, 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 Kern versucht einen Versuch, eine Nachricht an ein lokales Ziel zu senden, bei dem es sich um eine Greengrass-Lambda-Funktion, einen Konnektor 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 gleich.

MQTT-Nachrichtenwarteschlange für Cloud-Ziele

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

Standardmäßig speichert der Greengrass-Kern unverarbeitete Nachrichten, die für AWS Cloud Ziele bestimmt sind, im Speicher. Sie können den Core so konfigurieren, dass nicht verarbeitete Nachrichten stattdessen in einem lokalen Speichercache gespeichert werden. Im Gegensatz zu speicherinternem Speicher überdauert der lokale Speicher-Cache einen Neustart (z. B. nach der Bereitstellung einer Gruppe oder ein Geräteneustart), sodass AWS IoT Greengrass die Nachrichten weiter verarbeiten kann. Sie können auch die Speichergröße konfigurieren.

> **Warning**
>
> Der Greengrass-Kern stellt möglicherweise doppelte MQTT-Nachrichten in die Warteschlange, wenn er die Verbindung verliert, da er einen Veröffentlichungsvorgang wiederholt, bevor der MQTT-Client feststellt, 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 `mqttOperationTimeout` Werts. Weitere Informationen finden Sie unter AWS IoT Greengrass Core-Konfigurationsdatei.

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 Core-Geräts. 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. Dieser Wert kann eine beliebige nicht negative Ganzzahl größer oder gleich 262144 (256 KB) sein. 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 AWS IoT Greengrass so konfigurieren, dass Nachrichten im Dateisystem zwischengespeichert werden und einen Core-Neustart überdauern. Dazu stellen Sie eine Funktionsdefinitionsversion bereit, in der die `GGCloudSpooler`-Funktion den Speichertyp auf `FileSystem` festlegt. Sie müssen den lokalen Speicher-Cache mit der AWS IoT Greengrass-API konfigurieren. Sie können dies von der Konsole aus erledigen.

Im folgenden Verfahren wird der `create-function-definition-version` CLI-Befehl verwendet, 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 die IDs der Greengrass-Zielgruppen und die Gruppenversion ab. Bei diesem Verfahren wird davon ausgegangen, dass es sich um die neueste Gruppe 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 IoTKonsole. Die Gruppen-ID wird auf der Seite Einstellungen der Gruppe angezeigt. Gruppenversions-IDs werden auf der Registerkarte Bereitstellungen der Gruppe angezeigt.

2. Kopieren Sie die `LatestVersion`Werte und `Id` aus der Zielgruppe in die Ausgabe.
3. Rufen Sie die neueste Gruppenversion ab.

   - Ersetzen Sie *`group-id`* durch die kopierte `Id`.
   - 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 Objekt `Definition` in der Ausgabe den `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.

   - Ersetzen Sie durch *`function-definition-id`* die `Id`, die Sie für die Funktionsdefinition kopiert haben.
   - Ersetzen Sie *`arbitrary-function-id`* durch einen Namen für die Funktion, z. B. **`spooler-function`**.
   - 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 kleinere 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 zuvor die `GG_CONFIG_SUBSCRIPTION_QUALITY`-Umgebungsvariable so einstellen, dass sie persistente Sitzungen mit AWS IoT Core unterstützt, 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 System-Lambda-Funktion enthält.

   - Ersetzen Sie *`group-id`* durch die `Id` für die Gruppe.
   - Ersetzen Sie durch *`core-definition-version-arn`* die `CoreDefinitionVersionArn`, die Sie aus der neuesten Gruppenversion kopiert haben.
   - Ersetzen Sie durch *`function-definition-version-arn`* die `Arn`, die Sie für die neue Version der Funktionsdefinition kopiert haben.
   - Ersetzen Sie die ARNs für andere Gruppenkomponenten (zum Beispiel `SubscriptionDefinitionVersionArn` oder `DeviceDefinitionVersionArn`), die Sie aus der neuesten Gruppe 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.

    - Ersetzen Sie *`group-id`* durch die kopierte `Id` für die Gruppe.
    - Ersetzen Sie durch *`group-version-id`* die `Version`, die Sie für die neue Gruppenversion kopiert haben.

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

Sie können die Speichereinstellungen mit der AWS IoT Greengrass API aktualisieren, um eine neue Version der Funktionsdefinition zu erstellen, in der die `GGCloudSpooler`-Funktion mit der aktualisierten Konfiguration enthalten ist. 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-Nachrichtenbroker einrichten. 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-Nachrichtenbroker alle Abonnements, die der Core während der Verbindung erstellt. Wenn der Kern getrennt wird, speichert der AWS IoT Message Broker unbestätigte und neue Nachrichten, die als QoS 1 veröffentlicht und für lokale Ziele wie Lambda-Funktionen und Client-Geräte bestimmt sind. Wenn sich der Core erneut verbindet, wird die persistente Sitzung fortgesetzt, und der AWS IoT-Nachrichtenbroker 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 der Abonnements, die AWS IoT als Quelle haben. 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, Konnektor oder Gerät) wird als QoS 0 gesendet. garantiert AWS IoT Greengrass keine Übermittlung an lokale Ziele.

    > **Note**
    >
    > Sie können die Count-maxWorkItemKonfigurationseigenschaft verwenden, um die Größe der Arbeitselementwarteschlange 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 AWS IoT Greengrass so konfigurieren, 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.

Im folgenden Verfahren wird der `create-function-definition-version` CLI-Befehl verwendet, 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 die IDs der Greengrass-Zielgruppen und die Gruppenversion ab. Bei diesem Verfahren wird davon ausgegangen, dass es sich um die neueste Gruppe 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 IoTKonsole. Die Gruppen-ID wird auf der Seite Einstellungen der Gruppe angezeigt. Gruppenversions-IDs werden auf der Registerkarte Bereitstellungen der Gruppe angezeigt.

2. Kopieren Sie die `LatestVersion`Werte und `Id` aus der Zielgruppe in die Ausgabe.
3. Rufen Sie die neueste Gruppenversion ab.

   - Ersetzen Sie *`group-id`* durch die kopierte `Id`.
   - 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 Objekt `Definition` in der Ausgabe den `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.

   - Ersetzen Sie durch *`function-definition-id`* die `Id`, die Sie für die Funktionsdefinition kopiert haben.
   - Ersetzen Sie *`arbitrary-function-id`* durch einen Namen für die Funktion, z. B. **`spooler-function`**.
   - 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 System-Lambda-Funktion enthält.

   - Ersetzen Sie *`group-id`* durch die `Id` für die Gruppe.
   - Ersetzen Sie durch *`core-definition-version-arn`* die `CoreDefinitionVersionArn`, die Sie aus der neuesten Gruppenversion kopiert haben.
   - Ersetzen Sie durch *`function-definition-version-arn`* die `Arn`, die Sie für die neue Version der Funktionsdefinition kopiert haben.
   - Ersetzen Sie die ARNs für andere Gruppenkomponenten (zum Beispiel `SubscriptionDefinitionVersionArn` oder `DeviceDefinitionVersionArn`), die Sie aus der neuesten Gruppe 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.

    - Ersetzen Sie *`group-id`* durch die kopierte `Id` für die Gruppe.
    - Ersetzen Sie durch *`group-version-id`* die `Version`, die 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 maxWorkItemCount-Eigenschaft in der Core-Konfigurationsdatei. Dies kann dem Core helfen, erhöhten MQTT-Datenverkehr und die Kommunikation mit lokalen Zielen zu bewältigen.

Zum Aktualisieren des Cores mit diesen Konfigurationsänderungen verwenden Sie die AWS IoT Greengrass-API, um eine neue Funktionsdefinitionsversion 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-Core öffnet MQTT-Verbindungen mit AWS IoT Core für Operationen, wie etwa die Schattensynchronisierung und die Zertifikatsverwaltung. Für diese Verbindungen generiert der Core vorhersagbare Client-IDs auf Grundlage des Core-Namens. Vorhersehbare Client-IDs können mit Überwachungs-, Überwachungs- und Preisfunktionen verwendet werden, einschließlich AWS IoT Device Defender und AWS IoT Lebenszyklusereignissen. Sie können auch eine Logik im Zusammenhang mit vorhersagbaren Client-IDs erstellen (Beispiel: Aborichtlinien-Vorlagen auf Grundlage von Zertifikatsattributen).

GGC v1.9 and later

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

Operation	Client-ID Muster
Bereitstellungen	*`core-thing-name`* Beispiel: `MyCoreThing`

Operation	Client-ID Muster
	Verwenden Sie diese Client-ID für Benachrichtigungen bei folgenden Lebenszyklusereignissen: Verbinden, Verbindung trennen, Abonnieren und Abonnement kündigen.
Subscriptions (Abonnements)	*core-thing-name* `-c`*n* Beispiel: `MyCoreThing-c01` *n* 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, die ihren Schattenstatus mit synchronisieren AWS IoT Core (maximal 2 500 pro Gruppe) und die Anzahl der Abonnements mit `cloud` als Quelle in der Gruppe (maximal 10 000 pro Gruppe). Die Spooler-Systemkomponente stellt eine Verbindung mit herAWS IoT Core, um Nachrichten gegen Abonnements mit einer Cloud-Quelle oder einem Ziel auszutauschen. Der Spooler fungiert außerdem als Proxy für den lokalen Nachrichtenaustausch zwischen AWS IoT Core und dem lokalen Schattenservice und dem Device Certificate Manager.

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 subscriptions for supporting certificate generation + number of MQTT topics in AWS IoT Core + number of device shadows synced) / 50`.
  - Wobei `50` = die maximale Anzahl von Abonnements pro Verbindung ist, die unterstützen AWS IoT Core kann.

> **Note**
>
> Wenn Sie persistente Sitzungen für das Abonnement mit AWS IoT Core aktivieren, 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. Anschließend senden Sie die aggregierten Daten an AWS Cloud. Daher verwenden Sie weniger MQTT-Themen in AWS IoT Core. Weitere Informationen finden Sie unter AWS IoT Greengrass-Preisgestaltung.

GGC v1.8

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

Operation	Client-ID Muster
Bereitstellungen	*`core-thing-name`* Beispiel: `MyCoreThing` Verwenden Sie diese Client-ID für Benachrichtigungen bei folgenden Lebenszyklusereignissen: Verbinden, Verbindung trennen, Abonnieren und Abonnement kündigen.

Operation	Client-ID Muster
MQTT-Nachrichtenaustausch mit AWS IoT Core	*core-thing-name* `-spr` Beispiel: `MyCoreThing-spr`
Schattensynchronisierung	*core-thing-name* `-s`*nn* Beispiel: `MyCoreThing-s01` *nn* 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, die ihren Schattenstatus mit AWS IoT Core (maximal 50 Abonnements pro Verbindung) synchronisieren.
Gerätezertifikatsverwaltung	*core-thing-name* `-dcm` Beispiel: `MyCoreThing-dcm`

> **Note**
>
> Doppelte Client-IDs in gleichzeitigen Verbindungen können zu einer unendlichen Verbinden-Trennen-Schleife führen. Dies kann der Fall sein, wenn für ein anderes Gerät festgelegt ist, dass der Core-Gerätename 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 Flottenindizierungsservice im AWS IoT Entwicklerhandbuch für .

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 Message Broker 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 stoppen:

```
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 Kernkonnektivitätsinformationen zu erhalten. Sie können die AWS IoT Konsole oder AWS IoT Greengrass API verwenden, um die Kernkonnektivitätsinformationen 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 Seite Gruppenkonfiguration den Greengrass-Kern aus.
   2. Wählen Sie auf der Seite mit den Kerndetails die Registerkarte MQTT-Broker-Endpunkte aus.
   3. Wählen Sie Endpunkte verwalten und dann Endpunkt 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 das MQTT-Messaging mit AWS IoT Core verwendet. Weitere Informationen finden Sie unter the section called “Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy”.

Timeout für Veröffentlichungs-, Abonnement- und Abmeldevorgänge in MQTT-Verbindungen mit der 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. Beispielsweise:

```
{
  "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 konfigurierenAWS IoT Greengrass, dass Client-Geräte in einer Greengrass-Gruppe den Greengrass-Kern automatisch erkennen können. Wenn diese Option aktiviert ist, sucht der Core nach Änderungen an seinen IP-Adressen. Wenn sich eine Adresse ändert, veröffentlicht der Core

eine aktualisierte Liste von Adressen. Diese Adressen werden Client-Geräten zur Verfügung gestellt, die sich in derselben Greengrass-Gruppe wie der -Kern befinden.

> **Note**
>
> Die AWS IoT Richtlinie für Client-Geräte muss die -`greengrass:Discover`Berechtigung erteilen, damit Geräte Konnektivitätsinformationen für den Kern abrufen können. Weitere Informationen zur Richtlinienanweisung finden Sie unter the section called “Berechtigung zum Discovery”.

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

Um die automatische Erkennung mit der AWS IoT Greengrass API zu verwalten, müssen Sie die `IPDetector` System-Lambda-Funktion 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 die IDs der Greengrass-Zielgruppen und die Gruppenversion ab. Bei diesem Verfahren wird davon ausgegangen, dass es sich um die neueste Gruppe 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 IoTKonsole. Die Gruppen-ID wird auf der Seite Einstellungen der Gruppe angezeigt. Gruppenversions-IDs werden auf der Registerkarte Bereitstellungen der Gruppe angezeigt.

2. Kopieren Sie die `LatestVersion`Werte und `Id` aus der Zielgruppe in die Ausgabe.
3. Rufen Sie die neueste Gruppenversion ab.

   - Ersetzen Sie *`group-id`* durch die kopierte `Id`.
   - 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 Objekt `Definition` in der Ausgabe den `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 Sie die , die *`function-definition-id`* Sie für die Funktionsdefinition kopiert haben. Zum 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.

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

```
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 System-Lambda-Funktion enthält.

   - Ersetzen Sie *group-id* durch die `Id` für die Gruppe.
   - Ersetzen Sie durch *core-definition-version-arn* die `CoreDefinitionVersionArn`, die Sie aus der neuesten Gruppenversion kopiert haben.
   - Ersetzen Sie durch *function-definition-version-arn* die `Arn`, die Sie für die neue Version der Funktionsdefinition kopiert haben.
   - Ersetzen Sie die ARNs für andere Gruppenkomponenten (zum Beispiel `SubscriptionDefinitionVersionArn` oder `DeviceDefinitionVersionArn`), die Sie aus der neuesten Gruppe 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.

    - Ersetzen Sie *`group-id`* durch die kopierte `Id` für die Gruppe.
    - Ersetzen Sie durch *`group-version-id`* die `Version`, die 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 findet 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 die AWS IoT Greengrass Core-Software mit `apt` installiert haben, können Sie die systemd-Skripte verwenden, um „Start beim Booten“ 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 1Wartungspolitik

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 GreengrassVersionsschema

AWS IoT Greengrassverwendet 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 Core-Software AWS IoT Greengrass

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 Greengrassveröffentlicht 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 Bugfixes bieten. Die AWS Cloud Endgeräte und API-Operationen bleiben jedoch weiterhin verfügbar und werden gemäß dem AWS IoT GreengrassService 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 Endpunkte 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 Greengrassfügt 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 AWS IoT Greengrassveröffentlicht 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 AWSre: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 Greengrassermöglicht es Ihnen, AWS Lambda Funktionen auf IoT-Geräten auszuführen. AWS Lambdabietet 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 LambdaEntwicklerhandbuch.

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 IoTDevice Tester (IDT) für AWS IoT Greengrass V1 ermöglicht es Ihnen, Ihre AWS IoT Greengrass Geräte für die Aufnahme in den AWS PartnerGerätekatalog 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 PartnerAufnahme in den AWS Gerätekatalog 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 PartnerGerätekatalog 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 AWSSupport.

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-Konnektor	31. März 2022
Anschluss 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 GreengrassKernsoftware v1.10.x	31. März 2022
AWS IoT GreengrassKernsoftware v1.x Docker-Images	30. Juni 2022
AWS IoT GreengrassKernsoftware v1.11.x	30. Juni 2023
AWS IoT GreengrassKernsoftware 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 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 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

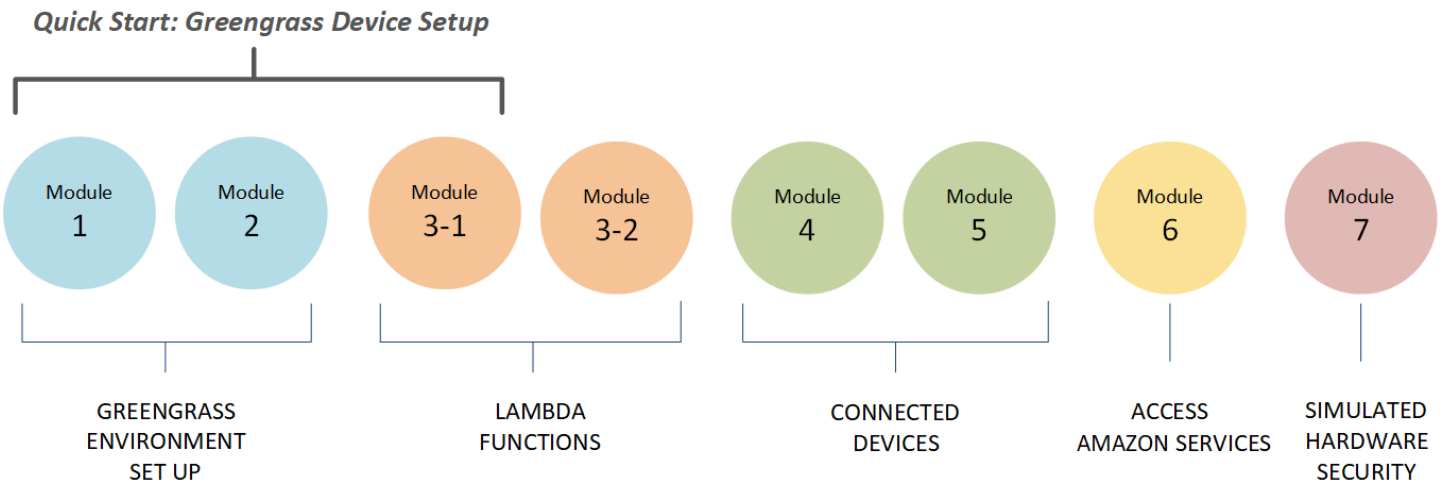
Wählen Sie aus, wie Sie beginnen möchten AWS IoT Greengrass

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.



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 EC2 EC2-Instance. 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/signup.
2. Folgen Sie den Online-Anweisungen.

   Bei der Anmeldung müssen Sie auch einen Telefonanruf entgegennehmen und einen Verifizierungscode über die Telefontasten eingeben.

   Wenn Sie sich für eine anmelden AWS-Konto, Root-Benutzer des AWS-Kontoswird eine erstellt. Der Root-Benutzer hat Zugriff auf alle AWS -Services und Ressourcen des Kontos. Aus Sicherheitsgründen sollten Sie einem Benutzer Administratorzugriff zuweisen und nur den Root-Benutzer verwenden, um Aufgaben auszuführen, für die Root-Benutzerzugriff erforderlich ist.

AWS sendet Ihnen nach Abschluss des Anmeldevorgangs eine Bestätigungs-E-Mail. Sie können jederzeit Ihre aktuelle Kontoaktivität anzeigen und Ihr Konto verwalten. Rufen Sie dazu https://aws.amazon.com/ auf und klicken Sie auf Mein Konto.

Erstellen Sie einen Benutzer 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 Management Consoleals 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 Sie einen Benutzer 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 Benutzer in IAM Identity Center Administratorzugriff.

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

Melden Sie sich als Benutzer mit Administratorzugriff an

- 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 Zugriffsportal.

Weisen Sie weiteren Benutzern Zugriff zu

1. Erstellen Sie in IAM Identity Center einen Berechtigungssatz, der der bewährten Methode zur Anwendung von Berechtigungen mit den geringsten Rechten folgt.

   Anweisungen finden Sie im Benutzerhandbuch unter Einen Berechtigungssatz erstellen.AWS IAM Identity Center

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

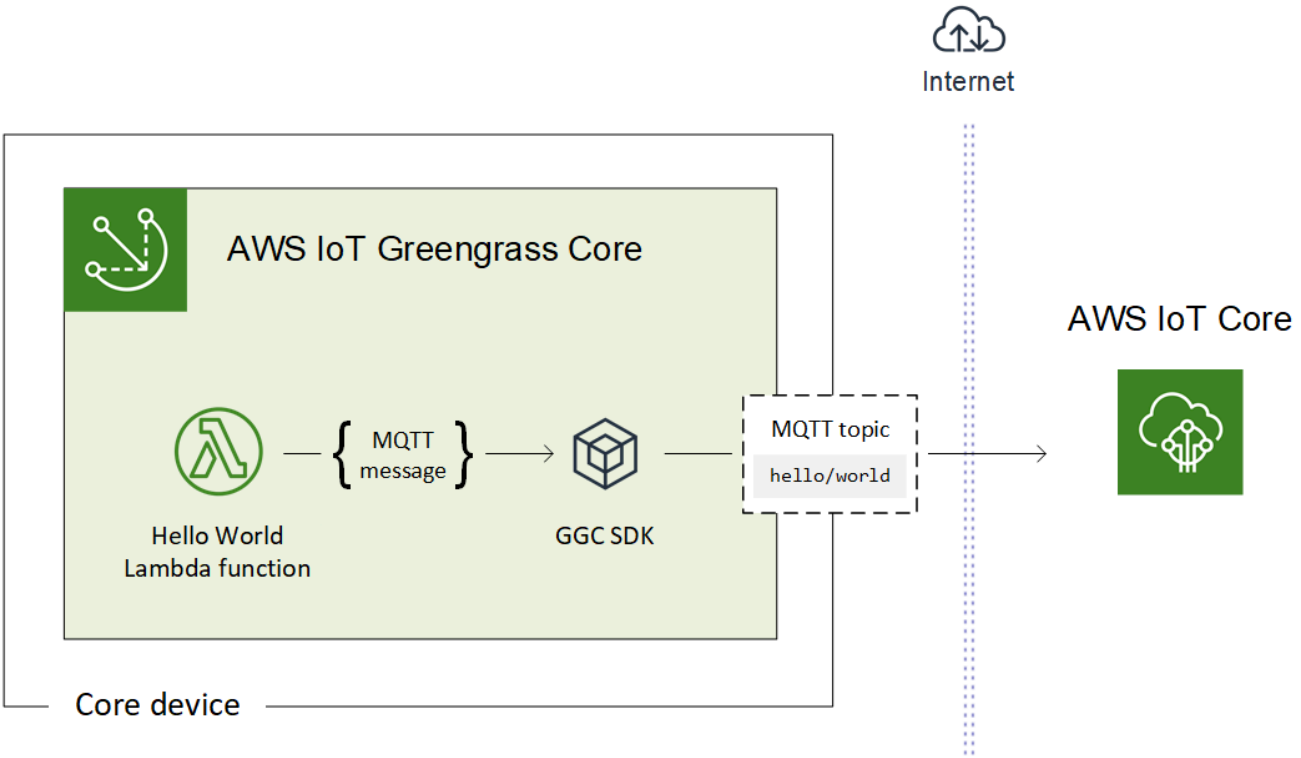
Anweisungen finden Sie im AWS IAM Identity Center Benutzerhandbuch unter Gruppen hinzufügen.

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

Schnellstart: Greengrass-Geräteeinrichtung

Greengrass Device Setup ist ein Skript, das Ihr Kerngerät in wenigen Minuten einrichtet, sodass Sie mit der Verwendung beginnen könnenAWS IoT Greengrass. Verwenden Sie dieses Skript für:

1. Konfigurieren Sie Ihr Gerät und installieren Sie dieAWS IoT Greengrass Core-Software.
2. Konfigurieren Sie Ihre cloudbasierten Ressourcen.
3. Stellen Sie optional eine Greengrass-Gruppe mit einer Hello World Lambda-Funktion bereit, die MQTT-NachrichtenAWS IoT vomAWS IoT Greengrass Core aus sendet. Dadurch wird die im folgenden Diagramm 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 IhreAWS-Konto Anmeldeinformationen angeben. Weitere Informationen finden Sie unter the section called “AWS-KontoAnmeldeinformationen angeben”.

> **Note**
>
> Greengrass Device Setup installiert die neueste Version der AWS IoT Greengrass-Core-Software auf dem Gerät. Indem Sie die AWS IoT Greengrass-Software installieren, stimmen Sie der Lizenzvereinbarung für die Greengrass Core-Software zu.

Ausführen von Greengrass Device Setup

Sie können Greengrass Device Setup in nur wenigen Schritten ausführen. Nachdem Sie IhreAWS-Konto Anmeldeinformationen eingegeben haben, stellt das Skript Ihr Greengrass-Core-Gerät bereit und stellt innerhalb weniger 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...o5OytwEXAMPLE=
```

> **Note**
>
> Wenn Sie das Greengrass-Gerätesetup 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 Eingabewertefort. Sie können die Eingabetaste drücken, um den Standardwert zu verwenden, oder einen benutzerdefinierten Wert eingeben und dann die Eingabetastedrücken.

   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 das Skript erneut.

   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...o5OytwEXAMPLE=
      ```

   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/versi
Greengrass service role: arn:aws:iam::012345678910:role/GreengrassServiceRole_mui1v
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. Überprüfen Sie die neue Greengrass-Gruppe, die das Skript anhand der von Ihnen angegebenen Eingabewerte konfiguriert.

   a. Melden Sie sich AWS Management Consoleauf Ihrem Computer bei der an und öffnen Sie dieAWS IoT Konsole.

> **Note**
>
> Stellen Sie sicher, dass die in der KonsoleAWS-Region ausgewählte Option dieselbe ist, die Sie für die 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 finden.

6. Wenn Sie die Hello World Lambda-Funktion integriert haben, stellt das Greengrass-Gerätesetup die Greengrass-Gruppe auf Ihrem Kerngerät bereit. Um die Lambda-Funktion zu testen oder Informationen zum Entfernen der Lambda-Funktion aus der Gruppe zuthe section called “Überprüfen, ob die Lambda-Funktion auf dem Core-Gerät ausgeführt wird” erhalten, fahren Sie mit Modul 3-1 des Tutorials Erste Schritte fort.

   > **Note**
   >
   > Stellen Sie sicher, dass die in der KonsoleAWS-Region ausgewählte Option dieselbe ist, die Sie für die Konfiguration Ihrer Greengrass-Umgebung verwendet haben. Standardmäßig ist die Region USA West (Oregon).

   Wenn Sie die Hello World Lambda-Funktion nicht integriert 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 derAWS IoT Greengrass Geräteeinrichtung zu beheben.

Fehler: Python (Python3.7) wurde nicht gefunden. Ich versuche es zu installieren...

Lösung: Dieser Fehler wird möglicherweise angezeigt, wenn Sie mit einer Amazon EC2 EC2-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
```

Weitere Fehlerbehebung

Um weitere Probleme mit derAWS IoT Greengrass Geräteeinrichtung 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 unterFehlerbehebung oder überprüfe das AWS IoT GreengrassTag aufAWS Re:Post.

Konfigurationsoptionen für Greengrass Device Setup

Sie konfigurieren das Greengrass-Gerätesetup, um auf IhreAWS Ressourcen zuzugreifen und Ihre Greengrass-Umgebung einzurichten.

AWS-KontoAnmeldeinformationen angeben

Das Greengrass-Gerätesetup verwendet IhreAWS-Konto Anmeldeinformationen, um auf IhreAWS Ressourcen zuzugreifen. Es unterstützt langfristige Anmeldeinformationen für einen IAM-Benutzer oder temporäre Sicherheits-Anmeldeinformationen aus einer 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. Weitere Informationen zum Erstellen von Zugriffsschlüsseln für langfristige Anmeldeinformationen finden Sie unter Verwalten von Zugriffsschlüsseln für IAM-Benutzer.

- Um temporäre Sicherheits-Anmeldeinformationen (empfohlen). Geben Sie die Zugriffsschlüssel-ID, den geheimen Zugriffsschlüssel und das Sitzungstoken an. Informationen zum Extrahieren temporärer Sicherheitsanmeldeinformationen aus demAWS STS`assume-role` Befehl finden SieAWS CLI im IAM-Benutzerhandbuch unter Verwenden temporärer Sicherheitsanmeldeinformationen mit dem.

> **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 Eingabetastedrücken. Im Hintergrundmodus geben Sie nach dem Start des Skripts Eingabewerte ein.

Eingabewerte

AWSZugangsschlüssel-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 “AWS-KontoAnmeldeinformationen angeben”.

Optionsname für den Hintergrundmodus: `--aws-access-key-id`

AWSGeheimer -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 “AWS-KontoAnmeldeinformationen angeben”.

Optionsname für den Hintergrundmodus: `--aws-secret-access-key`

AWSSitzungstoken

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 “AWS-KontoAnmeldeinformationen angeben”.

Optionsname für den Hintergrundmodus: `--aws-session-token`

AWS-Region

DerAWS-Region Ort, an dem Sie die Greengrass-Gruppe erstellen möchten Eine Liste der unterstütztenAWS-Region s finden Sie AWS IoT Greengrassin 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 Core ist ein AWS IoT-Gerät (Objekt), das die AWS IoT Greengrass-Core-Software ausführt. Der Core wird zur AWS IoT-Registry und zur Greengrass-Gruppe hinzugefügt. Wenn Sie einen Namen angeben, muss dieser imAWS-Konto und eindeutig seinAWS-Region.

Standardwert: `GreengrassDeviceSetup_Core_`*`guid`*

Optionsname für den Hintergrundmodus: `--core-name`

Installationspfad für die AWS IoT Greengrass-Core-Software

Der Speicherort im Geräte-Dateisystem, an dem die AWS IoT Greengrass-Core-Software installiert werden soll.

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 inAWS 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 Python 3.7 Lambda-Funktion. 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 dazu 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 IhreAWS-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...o5OytwEXAMPLE=
   ```

   > **Note**
   >
   > Wenn Sie das Greengrass-Gerätesetup 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. 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 das Skript erneut.

   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...o5OytwEXAMPLE=
      ```

   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/versi
Greengrass service role: arn:aws:iam::012345678910:role/GreengrassServiceRole_mui1v
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 integriert haben, stellt das Greengrass-Gerätesetup die Greengrass-Gruppe auf Ihrem Kerngerät bereit. Um die Lambda-Funktion zu testen oder Informationen zum Entfernen der Lambda-Funktion aus der Gruppe zuthe section called “Überprüfen, ob die Lambda-Funktion auf dem Core-Gerät ausgeführt wird” erhalten, fahren Sie mit Modul 3-1 des Tutorials Erste Schritte fort.

> **Note**
>
> Stellen Sie sicher, dass die in der KonsoleAWS-Region ausgewählte Option dieselbe ist, die Sie für die Konfiguration Ihrer Greengrass-Umgebung verwendet haben. Standardmäßig ist die Region USA West (Oregon).

Wenn Sie die Hello World Lambda-Funktion nicht integriert 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...o5OytwEXAMPLE=
     ```

   - 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...o5OytwEXAMPLE=
     ```

```
--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-Gerätesetup 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 das Skript erneut.

   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. 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...o5OytwEXAMPLE=
```

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/versi
Greengrass service role: arn:aws:iam::012345678910:role/GreengrassServiceRole_mui1v
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 integriert haben, stellt das Greengrass-Gerätesetup die Greengrass-Gruppe auf Ihrem Kerngerät bereit. Um die Lambda-Funktion zu testen oder Informationen zum Entfernen der Lambda-Funktion aus der Gruppe zuthe section called “Überprüfen, ob die Lambda-Funktion auf dem Core-Gerät ausgeführt wird” erhalten, fahren Sie mit Modul 3-1 des Tutorials Erste Schritte fort.

> **Note**
>
> Stellen Sie sicher, dass die in der KonsoleAWS-Region ausgewählte Option dieselbe ist, die Sie für die Konfiguration Ihrer Greengrass-Umgebung verwendet haben. Standardmäßig ist die Region USA West (Oregon).

Wenn Sie die Hello World Lambda-Funktion nicht integriert 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: Einrichten der Umgebung für Greengrass

Dieses Modul führt Sie durch die Einrichtung einer out-of-the-box Raspberry Pi, Amazon EC2 EC2-Instance oder eines anderen Geräts, das von verwendet werden kannAWS IoT Greengrassals IhreAWS IoT Greengrass-Core-Gerät.

> **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
- Einrichtung einer Amazon EC2 EC2-Instance
- Einrichten anderer Geräte

> **Note**
>
> Weitere Informationen, wie Sie AWS IoT Greengrass in einem vorkonfigurierten Docker-Container verwenden, finden Sie unter the section called “Ausführen von AWS IoT Greengrass in einem Docker-Container”.

Einrichten eines Raspberry Pi

Befolgen Sie die Schritte in diesem Thema, um einen Raspberry Pi alsAWS IoT GreengrassCores.

> **Tip**
>
> AWS IoT Greengrass bietet auch andere Optionen für die Installation der AWS IoT Greengrass Core-Software. Sie können beispielsweise das Greengras-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.

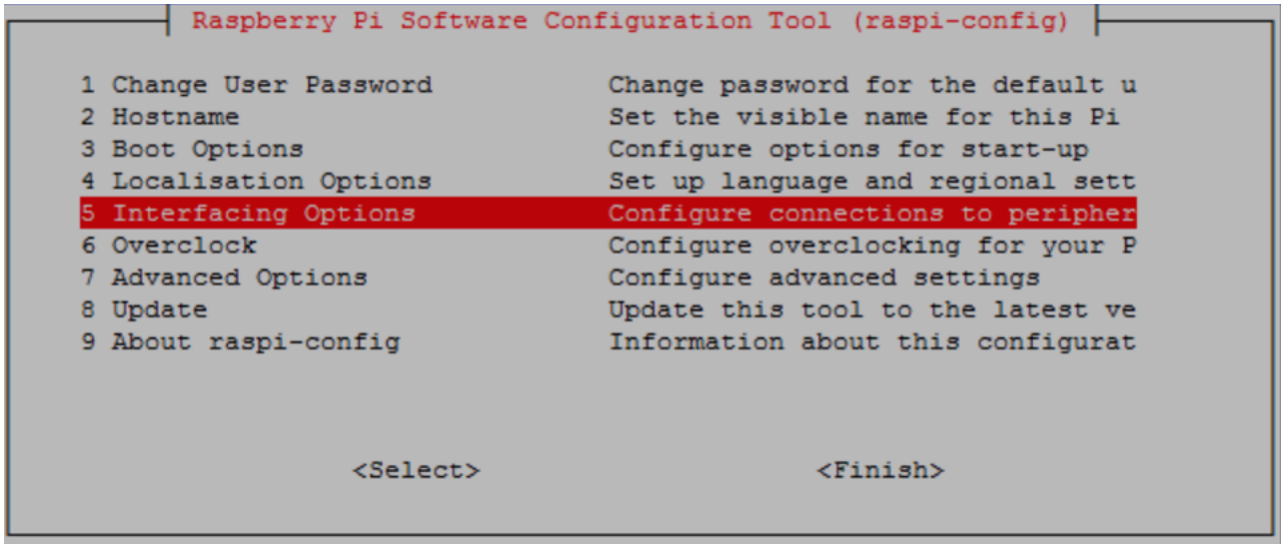
1. Laden Sie einen SD-Karten-Formatierer herunter und installieren Sie ihn, z.SD-Speicherkarten-Formatierer. Legen Sie die SD-Karte in Ihren Computer. Starten Sie das Programm und wählen Sie das Laufwerk aus, in das Sie die SD-Karte eingelegt haben. Sie können eine Schnellformatierung der SD-Karte durchführen.
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. 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.

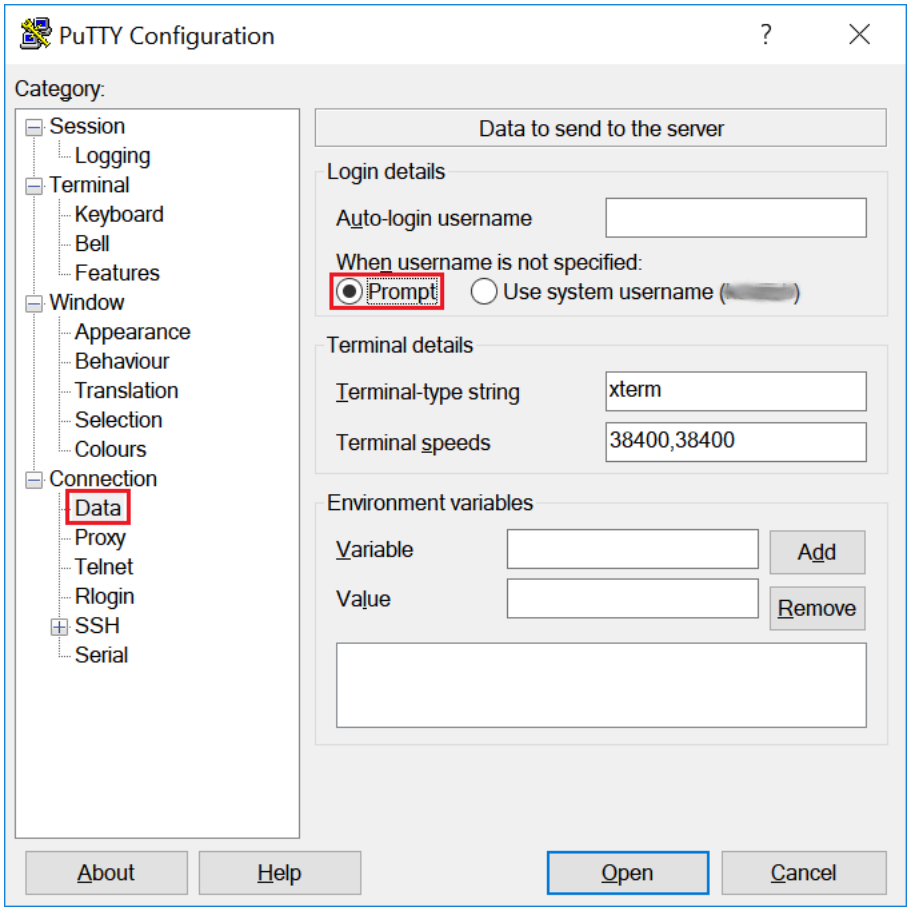
> **Note**
>
> Wenn Sie nachfolgend eine Meldung hinsichtlich des Fingerabdrucks für den ECDSA-Schlüssel (`Are you sure you want to continue connecting (yes/no)?`) erhalten, geben Sie `yes` ein. Das Standardpasswort für den Raspberry Pi lautet **`raspberry`**.

Wenn Sie macOS verwenden, öffnen Sie ein Terminalfenster und geben Sie Folgendes ein:

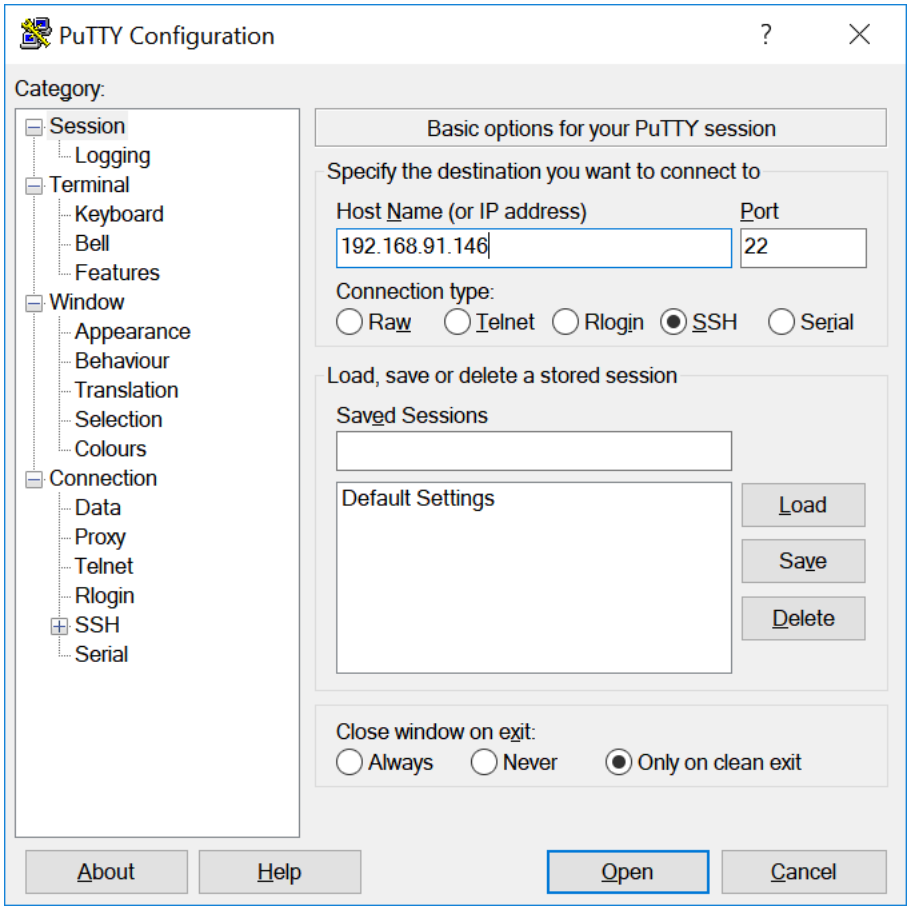
```
ssh pi@IP-address
```

Hier entspricht *IP-address* der IP-Adresse Ihres Raspberry Pi, die Sie mit dem Befehl `hostname -I` 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:

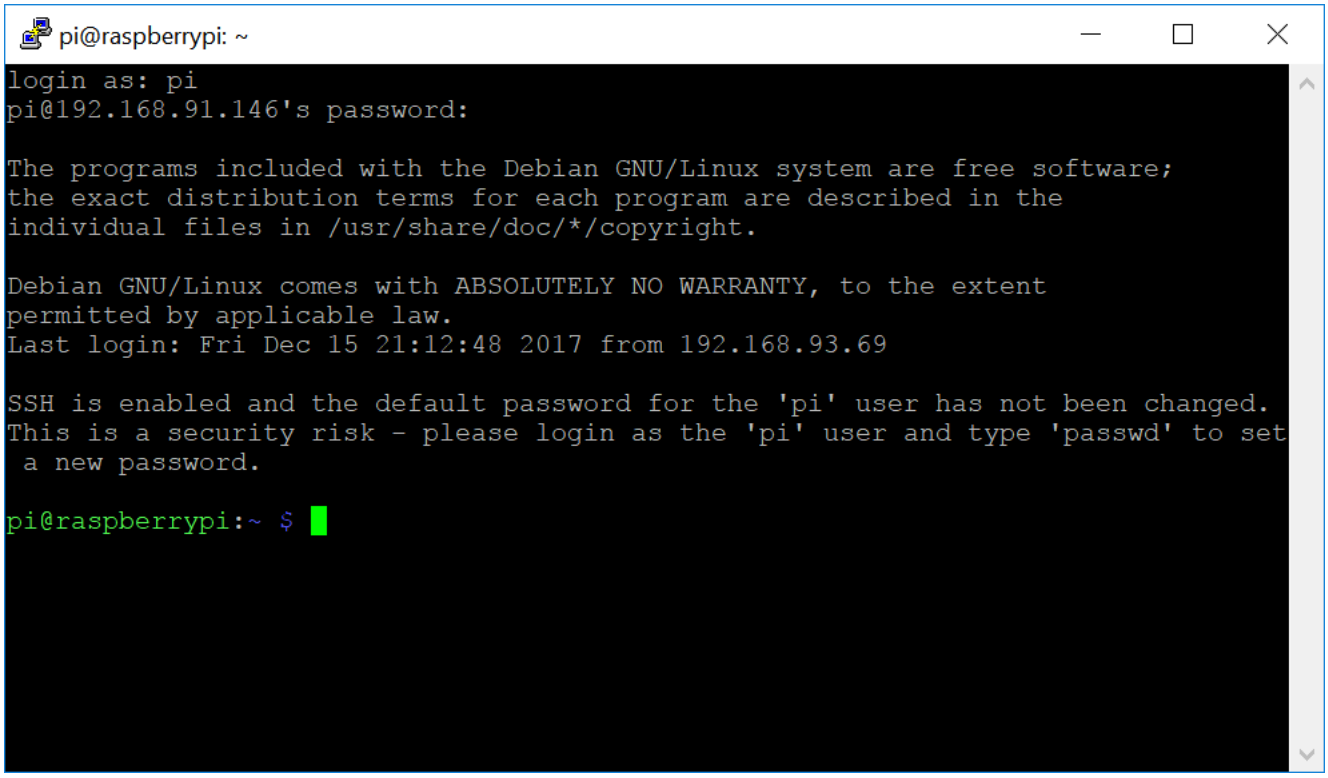


Wählen Sie Session (Sitzung), geben Sie die IP-Adresse des Raspberry Pi ein und wählen Sie dann Open (Öffnen) unter Verwendung der Standardeinstellungen.



Wenn eine PuTTY-Sicherheitswarnung angezeigt wird, klicken Sie auf Yes (Ja).

Die Standardanmeldedaten und das Passwort für den Raspberry Pi lauten **`pi`** und **`raspberry`**.



> **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. Jetzt können Sie den Raspberry Pi für AWS IoT Greengrass einrichten. 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 Greengras-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. Auf diese Weise kann AWS IoT Greengrass das Speicherlimit für Lambda-

Funktionen festlegen. Cgroups müssen auch ausgeführt werdenAWS IoT Greengrassin der Standard-Containerisierungmode.

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 von derAWS IoT GreengrassBeispieleRepository auf GitHub. Diese Befehle entpacken das Abhängigkeitsprüftools im`Downloads`-Verzeichnis.

> **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 im Containermodus ausgeführte Lambda-Funktionen fehlschlagen.

Weitere Informationen zum Aktualisieren Ihres Kernels finden Sie in derCgroups wurden nach dem Kernel-Upgrade nicht geladenin den Raspberry Pi-Foren.

```
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**
>
> Für dieses Tutorial ist die Python 3.7-Laufzeitumgebung erforderlich, um lokale Lambda-- 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: Installieren vonAWS IoT GreengrassCore-Software”.

Einrichtung einer Amazon EC2 EC2-Instance

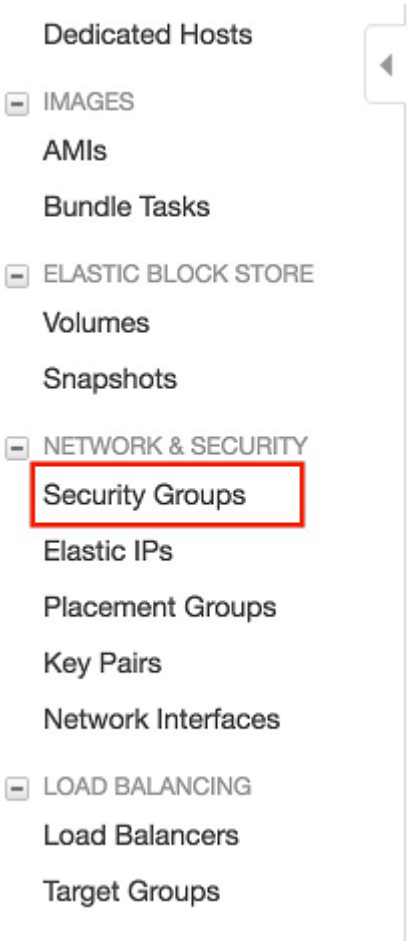
Folgen Sie den Schritten in diesem Thema, um eine Amazon EC2 EC2-Instance einzurichten, die Sie als Ihren AWS IoT Greengrass Kern verwenden können.

> **Tip**
>
> Oder, wie Sie ein Skript verwenden, das Ihre Umgebung einrichtet und die AWS IoT Greengrass Core-Software für Sie installiert, finden Sie unterthe section called “Schnellstart: Greengrass-Geräteeinrichtung”.

Sie können dieses Tutorial zwar mit einer Amazon EC2 EC2-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 Amazon EC2 EC2-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 Management Consoleund starten Sie eine Amazon EC2 EC2-Instance mit einem Amazon Linux AMI. Informationen zu Amazon EC2-Instances finden Sie im Amazon EC2 Getting Started Guide.
2. Nachdem Ihre Amazon EC2 EC2-Instance ausgeführt wurde, aktivieren Sie Port 8883, um eingehende MQTT-Kommunikation zuzulassen, sodass sich andere Geräte mit dem Core verbinden können. AWS IoT Greengrass

   a. Wählen Sie im Navigationsbereich der Amazon EC2 EC2-Konsole Security Groups 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 Hinzufügen von Regeln zu einer Sicherheitsgruppe im Amazon EC2 EC2-Benutzerhandbuch.

d. Wählen Sie auf der Seite Regeln für eingehenden Datenverkehr bearbeiten die Option Regel hinzufügen aus, geben Sie die folgenden Einstellungen ein und klicken Sie dann auf 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. Stellen Sie eine Verbindung zu Ihrer Amazon-EC2-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 Amazon EC2 EC2-Benutzerhandbuch.

   Sie sind jetzt bereit, Ihre Amazon EC2 EC2-Instance für AWS IoT Greengrass einzurichten.

4. Nachdem Sie mit Ihrer Amazon EC2 EC2-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 Amazon EC2 EC2-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 Amazon EC2 EC2-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 Amazon EC2 EC2-Instance sollte jetzt bereit für AWS IoT Greengrass sein.

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 Amazon EC2 EC2-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 Amazon EC2 EC2-Instance-Konfiguration ist abgeschlossen. Fahren Sie fort mit the section called “Modul 2: Installieren vonAWS IoT GreengrassCore-Software”.

Einrichten anderer Geräte

Befolgen Sie die Schritte in diesem Thema, um ein anderes Gerät als einen Raspberry Pi als Ihre einzurichtenAWS IoT GreengrassCore-Wert.

> **Tip**
>
> Weitere Informationen zum Verwenden eines Skripts, das Ihre Umgebung einrichtet und die AWS IoT Greengrass-Core-Software für Sie installiert, finden Sie unter the section called “Schnellstart: Greengrass-Geräteeinrichtung”.

Wenn für Sie neu istAWS IoT Greengrassempfehlen wir, dass Sie einen Raspberry Pi oder eine Amazon EC2 EC2--Instance als Ihr Core-Gerät verwenden undSetup-Schrittepassend für Ihr Gerät.

Wenn Sie planen, mit dem Yocto-Projekt ein benutzerdefiniertes Linux-basiertes System zu erstellen, können Sie dieAWS IoT GreengrassBitbake-Rezept aus dem`meta-aws`-Projekt. Dieses Rezept hilft Ihnen auch bei der Entwicklung einer Softwareplattform, die unterstütztAWSEdge-Software für eingebettete Anwendungen. Der Bitbake-Build installiert, konfiguriert und führt automatischAWS IoT GreengrassCore-Software auf Ihrem Gerät.

Yocto-Projekt

Ein Open-Source-Kollaborationsprojekt, das Ihnen hilft, benutzerdefinierte Linux-basierte Systeme für eingebettete Anwendungen unabhängig von der Hardwarearchitektur Weitere Informationen finden Sie im .Yocto-Projektaus.

`meta-aws`

Importieren in &S3;AWSverwaltetes Projekt, das Yocto-Rezepte bereitstellt. Sie können die Rezepte verwenden, um sich zu entwickelnAWSEdge-Sofware in Linux-basierten Systemen gebaut mitOpenEmbeddedund Yocto Project. Weitere Informationen zu dieser Community unterstützten Funktion finden Sie unter`meta-aws`-Projekt auf GitHub.

`meta-aws-demos`

Importieren in &S3;AWSverwaltetes Projekt, das Demonstrationen für die`meta-aws`-Projekt. Weitere Beispiele für den Integrationsprozess finden Sie unter`meta-aws-demos`-Projekt auf 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 Core-Gerät um ein NVIDIA Jetson-Gerät handelt, müssen Sie zuerst die Firmware mit dem JetPack 4.3 -Installationsprogramm. Wenn Sie ein anderes Gerät konfigurieren, fahren Sie mit Schritt 2 fort.

   > **Note**
   >
   > Die JetPack Die Version des -Installationsprogramms, die Sie verwenden, basiert auf Ihrer Ziel-CUDA-Toolkit-Version. Mit den folgenden Anweisungen 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. Auf einem physischen Desktop, auf dem Ubuntu 16.04 oder höher ausgeführt wird, aktualisieren Sie die Firmware mit dem JetPack 4.3 Installationsprogramm, wie unterHerunterladen und Installieren von JetPack(4.3) In der NVIDIA-Dokumentation.

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 OpenWrt ausführt, 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 die Greengrass-Abhängigkeitsprüfung von derAWS IoT GreengrassBeispiele-Repository auf GitHub. Diese Befehle entpacken das Abhängigkeitsprüftool-Skript und führen es aus.

```
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 wird auf von AWS IoT Greengrass unterstützten Plattformen ausgeführt und erfordert spezifische 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 cgroupfs-mount ausführen. Auf diese Weise kann AWS IoT Greengrass das Speicherlimit für Lambda-Funktionen festlegen. Für die Ausführung sind auch Cgroups erforderlichAWS IoT GreengrassIn der Standard-Containerisierung-Modus.

   Wenn keine Fehler in der Ausgabe angezeigt werden, sollte AWS IoT Greengrass jetzt erfolgreich auf Ihrem Gerät ausgeführt werden können.

> **Important**
>
> Für dieses Tutorial ist die Python 3.7-Laufzeitumgebung erforderlich, um lokale Lambda-Funktionen ausführen zu können. 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.

Die Liste der AWS IoT Greengrass-Anforderungen und -Abhängigkeiten finden Sie unter the section called “Unterstützte Plattformen und Anforderungen”.

Modul 2: Installieren vonAWS IoT GreengrassCore-Software

In diesem Modul erfahren Sie, 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 “Konfigurieren des AWS IoT Greengrass Core”.

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 Greengras-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

BereiteinesAWS IoTals Greengrass-Kern zu verwenden

GreengrassKernesind Geräte, auf denen dasAWS IoT GreengrassKernsoftware zur Verwaltung lokaler IoT-Prozesse. Um einen Greengrass-Kern einzurichten, erstellen Sie einAWS IoT Sache, die ein Gerät oder eine logische Entität darstellt, die eine Verbindung mitAWS IoTaus. Wenn Sie ein Gerät alsAWS IoT-Element, dieses Gerät kann ein digitales Zertifikat und Schlüssel verwenden, die es zum Zugriff aufAWS IoTaus. Du benutzt einAWS IoTPolitikum dem Gerät die Kommunikation mit demAWS IoTundAWS IoT Greengrass-Services.

In diesem Abschnitt registrieren Sie Ihr Gerät alsAWS IoTvon -Elementen.

So erstellen Sie einenAWS IoTSache

1. Navigieren Sie zur AWS IoT-Konsole.
2. UnderVerwalten, erweiternAlle Geräteund wählen Sie dannElementeaus.
3. Auf derElemente-Seite, wählen SieErstellen von Objektenaus.
4. Auf derErstellen von Objekten-Seite, wählen SieEinzelnes Objektund wählen Sie dannWeiteraus.
5. Auf derGeben Sie Ding-Eigenschaften-Seite, machen Sie das Folgende:

   a. FürThing-Nameeinen Namen ein, der für Ihr Gerät steht, z. B.**`MyGreengrassV1Core`**aus.

   b. Wählen Sie Next (Weiter).
6. Auf derGerätezertifikat.-Seite, wählen SieWeiteraus.
7. Auf derRichtlinien an-Seite einen der folgenden Schritte aus:
   - Wählen Sie eine vorhandene Richtlinie aus, die Berechtigungen gewährt, die Kerne benötigen, und wählen Sie dannErstellen eines -Objektsaus.

     Ein Modal wird geöffnet, in dem Sie die Zertifikate und Schlüssel herunterladen können, die das Gerät für die Verbindung mit demAWS Cloudaus.
   - Erstellen und fügen Sie eine neue Richtlinie hinzu, die Kerngeräteberechtigungen gewährt. Gehen Sie wie folgt vor:

a. Wählen Sie Create Policy (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. FürRichtliniennameeinen beschreibenden Namen für die Richtlinie ein, z. B.**`GreengrassV1CorePolicy`**aus.

   ii. Auf derRichtlinienananunterRichtliniendokument, wählenJSONaus.

   iii. Geben Sie das folgende -Richtliniendokument ein. Diese Richtlinie ermöglicht es dem Core, mit demAWS IoT Core-Dienst, interagieren Sie mit Device Shadows und kommunizieren Sie mit demAWS IoT GreengrassService. Weitere Informationen, wie Sie den Zugriff auf diese Richtlinie auf der Grundlage Ihres Anwendungsfalls einschränken, finden Sie unterMinimale AWS IoT-Richtlinie für das AWS IoT Greengrass Core-Gerätaus.

```
{
  "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 dem Browser-Registerkarte zurückRichtlinien an-Seite offen. Gehen Sie wie folgt vor:

i. In derRichtliniendie von Ihnen erstellte Richtlinie aus, z. B.GreengrassV1CorePolicyaus.

Wenn die Richtlinie nicht angezeigt wird, klicken Sie auf die Schaltfläche Refresh.

ii. Klicken Sie aufErstellen eines -Objektsaus.

Ein Modal wird geöffnet, in dem Sie die Zertifikate und Schlüssel herunterladen können, mit denen der Core eine Verbindung herstelltAWS IoTaus.

8. Kehren Sie mit dem Browser-Registerkarte zurückRichtlinien an-Seite offen. Gehen Sie wie folgt vor:

a. In derRichtliniendie von Ihnen erstellte Richtlinie aus, z. B.GreengrassV1CorePolicyaus.

Wenn die Richtlinie nicht angezeigt wird, klicken Sie auf die Schaltfläche Refresh.

b. Klicken Sie aufErstellen eines -Objektsaus.

Ein Modal wird geöffnet, in dem Sie die Zertifikate und Schlüssel herunterladen können, mit denen der Core eine Verbindung herstelltAWS IoTaus.

9. In derLaden Sie Zertifikate und Schlüssel heruntermodal, laden Sie die Zertifikate des Geräts herunter.

> **Important**
>
> Bevor Sie sichFertig, laden Sie die Sicherheitsressourcen herunter.

Gehen Sie wie folgt vor:

a. FürGerätezertifikat, wählenHerunterladenum das Gerätezertifikat herunterzuladen.

b. FürDatei mit dem öffentlichen Schlüssel, wählenHerunterladenum den öffentlichen Schlüssel für das Zertifikat herunterzuladen.

c. FürPrivate Schlüsseldatei, wählenHerunterladen, um die Datei mit dem privaten Schlüssel für das Zertifikat herunterzuladen.

d. PrüfenServerauthentifizierungimAWS IoTEntwicklerhandbuchund wählen Sie das entsprechende CA-Stammzertifikat aus. Wir empfehlen die Verwendung eines Amazon Trust Services (ATS) -CA-Stammzertifikate. UnderCA-Stammzertifikate, wählenHerunterladenfür ein -CA-Stammzertifikat.

e. Wählen Sie Done (Erledigt) aus.

Notieren Sie sich die in den Dateinamen für das Gerätezertifikat und die Schlüssel allgemeine ZertifikatsID. Sie benötigen sie später.

Erstellen Sie eineAWS IoT Greengrass Gruppe für den Kern

AWS IoT GreengrassGruppen enthalten Einstellungen und andere Informationen zu ihren Komponenten, z. B. Client-Geräten, Lambda-Funktionen und Connectors. 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 dieAWS IoT Greengrass API verwendet, um eine Gruppe zu erstellen und bereitzustellen, finden Sie im gg_group_setup-Repository auf GitHub.

Um eine Gruppe für den Core zu erstellen

1. Navigieren Sie zur AWS IoT-Konsole.
2. Erweitern Sie unter Verwalten die Option Greengrass-Geräte und wählen Sie Gruppen (V1).

> **Note**
>
> Wenn Sie das Greengrass-Gerätemenü nicht sehen, wechseln Sie zu einem MenüAWS-Region, das unterstütztAWS IoT Greengrass V1. Eine Liste der unterstützten Regionen finden Sie unter AWS IoT Greengrass V1Endpunkte und Kontingente in der Allgemeine AWS-Referenz. Sie müssen dasAWS IoT Ding für Ihren Kern in einer Region erstellen, in derAWS IoT Greengrass V1 es verfügbar ist.

3. Wählen Sie auf der Seite Greengrass-Gruppen die Option Gruppe erstellen aus.
4. Führen Sie auf der Seite Greengrass -Gruppe die folgenden Schritte aus:

   a. Geben Sie für den Greengrass-Gruppennamen einen Namen ein, der die Gruppe beschreibt, z.**`MyGreengrassGroup`** B.

   b. Wählen Sie für Greengrass Core dasAWS IoT Ding aus, das Sie zuvor erstellt haben, z. MyGreengrassV1CoreB.

      Die Konsole wählt automatisch das Gerätezertifikat des Dings für Sie aus.

   c. Wählen Sie Create group (Gruppe erstellen) aus.

Installieren und AusführenAWS IoT Greengrassauf dem Core-Gerät

> **Note**
>
> In diesem Tutorial erhalten Sie Anweisungen zum Ausführen desAWS IoT GreengrassCore-Software auf einem Raspberry Pi, Sie können aber jedes unterstützte Gerät verwenden.

In diesem Abschnitt konfigurieren, installieren und führen Sie denAWS IoT GreengrassCore-Software auf Ihrem Core-Gerät.

So installieren und AusführenAWS IoT Greengrass

1. FromAWS IoT GreengrassCore-Softwarein diesem Handbuch die DateiAWS IoT GreengrassCore-Software—Installationspaket 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 Amazon EC2 EC2-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 enthältAWS IoT GreengrassCore-Software, die auf dem Core-Gerät ausgeführt wird.
   - *`certificateId`*`-certificate.pem.crt`— Die Gerätezertifikatdatei.
   - *`certificateId`*`-public.pem.key`— Die öffentliche Schlüsseldatei des Gerätezertifikats.
   - *`certificateId`*`-private.pem.key`— Die private Schlüsseldatei des Gerätezertifikats.
   - `AmazonRootCA1.pem`— Amazon-Stammzertifizierungsstelle (Certificate Authority, CA)

   In diesem Schritt übertragen Sie diese Dateien von Ihrem Computer auf Ihr Core-Gerät. 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 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 Core-Gerät. Die Schritte für die Dateiübertragung variieren je nach dem Betriebssystem Ihres Computers. Wählen Sie Ihr Betriebssystem für Schritte, die zeigen, wie Sie Dateien an Ihr Raspberry Pi-Gerät übertragen.

> **Note**
>
> Bei einem Rasberry PI lautet der Standardbenutzername **`pi`** und das Passwort ist **`raspberry`**.
> Bei einem NVIDIA Jetson TX2-Gerät lautet der Standardbenutzername **`nvidia`** und das Passwort ist **`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. Die*path-to-downloaded-files*ist in der Regel`~/Downloads`aus.

> **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 zum Herunterladen der komprimierten Dateien verwendeten, sollten sich die Dateien in den von Pi befinden`~/Downloads`Ordner, wie`/home/pi/Downloads`aus. Andernfalls sollten sich die komprimierten Dateien in den Pi befinden`~`Ordner, wie`/home/pi`aus.

3. Öffnen Sie auf dem Greengrass Core-Gerät ein Terminal und gehen zu dem Ordner mit dem VerzeichnisAWS IoT GreengrassKernsoftware und -zertifikate. Ersetzen*path-to-transferred-files*mit dem Pfad, in den Sie die Dateien auf das Core-Gerät übertragen haben. Führen Sie z. B. auf einem Raspberry Pi aus`cd /home/pi`aus.

   ```
   cd path-to-transferred-files
   ```

4. Extrahieren Sie dasAWS IoT GreengrassCore-Software auf dem Core-Gerät. Führen Sie den folgenden Befehl aus, um das Softwarearchiv, das Sie auf das Core-Gerät übertragen haben, zu entpacken. Dieser Befehl verwendet den`-C /`Argument zum Erstellen des`/greengrass`im Stammordner des Core-Geräts.

   ```
   sudo tar -xzvf 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 dieAWS IoT GreengrassCore-Software Ordner. Führen Sie die folgenden Befehle aus, um einen Ordner für Zertifikate zu erstellen und die Zertifikate und Schlüssel dorthin zu verschieben. Ersetzen*path-to-transferred-files*mit dem Pfad, in den Sie die Dateien auf das Kerngerät übertragen haben, und ersetzen

Sie*certificateId*mit der Zertifikat-ID in den Dateinamen. Ersetzen Sie z. B. auf einem Raspberry Pi*path-to-transferred-files*mit**/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. DieAWS IoT GreengrassDie Kernsoftware verwendet eine Konfigurationsdatei, die Parameter für die Software festlegt. Diese Konfigurationsdatei legt die Dateipfade für Zertifikatsdateien fest undAWS Cloudzu verwendende -Endpunkte. In diesem Schritt erstellen Sie das VerzeichnisAWS IoT GreengrassCore-Software-Konfigurationsdatei für Ihren Core. Gehen Sie wie folgt vor:

   a. Holen Sie sich den Amazon-Ressourcennamen (ARN) Ihres Core'sAWS IoTthing. Gehen Sie wie folgt vor:

      i. In derAWS IoTKonsole, unterVerwalten, unterGreengrass Geräte, wählenGruppen (V1)aus.

      ii. Auf derGreengrass Gruppendie -Gruppe, die Sie vorher erstellt haben.

      iii. UnderÜbersicht, wählenGreengrass-Kernaus.

      iv. Kopieren Sie auf der Core-Detailseite die DateiAWS IoTARN des, und speichern Sie es zur Verwendung imAWS IoT GreengrassCore-Konfigurationsdatei.

   b. Get dasAWS IoTGerätedaten-Endpunkt für IhrAWS-Kontoin der aktuellen -Region. Geräte verwenden diesen Endpunkt, um eine Verbindung herzustellenAWSalsAWS IoTthings. Gehen Sie wie folgt vor:

      i. In derAWS IoTKonsole, wählenEinstellungenaus.

      ii. UnderGerätedatenendpunktkopieren Sie das Verzeichnis-Endpunkt, und speichern Sie es zur Verwendung imAWS IoT GreengrassCore-Konfigurationsdatei.

   c. Erstellen derAWS IoT GreengrassKonfigurationsdatei für die Kernsoftware. Zum Beispiel können Sie den folgenden Befehl ausführen, um GNU nano zum Erstellen der Datei zu verwenden.

      ```
      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 anschließend 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*mit dem Namen der Amazon-Root-CA-Datei.
- Ersetzen Sie jede Instanz von*certificateId*mit der Zertifikat-ID im Namen des Zertifikats und der Schlüsseldateien.

- Ersetzen*arn: aws:iot:Region:account-id:thing/MyGreengrassV1-Kern*mit dem ARN Ihres Core, den Sie zuvor gespeichert haben.
- Ersetzen*MyGreengrassV1-Kern*mit dem Namen Ihres Core's thing.
- Ersetzen*device-data-prefix-ats.iot.region.amazonaws.com*mit demAWS IoTGerätedatenendpunkt, den Sie zuvor gespeichert haben.
- Ersetzen*Region*mit IhremAWS-Regionaus.

Weitere Informationen zu den Konfigurationsoptionen, die Sie in dieser Konfigurationsdatei angeben können, finden Sie unterAWS IoT Greengrass Core-Konfigurationsdateiaus.

7. Stellen Sie sicher, dass Ihr Core-Gerät mit dem Internet verbunden ist. Starten Sie anschließendAWS IoT Greengrassauf 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 zur Einrichtung Ihres Core-Geräts für die Ausführung von AWS IoT Greengrass beim Systemstart finden Sie unter the section called “Ausführen von Greengrass bei Systemstart”.

Sie können den folgenden Befehl ausführen, um den ordnungsgemäßen Betrieb der AWS IoT Greengrass Core-Software (Greengrass-Daemon) zu bestätigen. Ersetzten Sie *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`). Bei Problemen mit dem Starten von AWS IoT Greengrass vgl. Fehlerbehebung.

Modul 3 (Teil 1): Lambda-Funktionen aufAWS IoT Greengrass

In diesem Modul erfahren Sie, wie Sie eine Lambda-Funktion erstellen und bereitstellen, die MQTT-Nachrichten vonAWS IoT Greengrass-Core-Gerät. In diesem Modul werden die -Funktionskonfigurationen, Abonnements zum Zulassen von MQTT-Messaging und Bereitstellungen auf einem Core-Gerät beschrieben.

Modul 3 (Teil 2)behandelt die Unterschiede zwischen On-Demand- und langlebigen Lambda-Funktionen in derAWS IoT Greengrass-Cores.

Stellen Sie vor Beginn sicher, dass Sie abgeschlossen habenModul 1undModul 2und laufeAWS IoT Greengrass-Core-Gerät.

> **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

- Erstellen und Verpacken einer Lambda-Funktion
- Konfigurieren Sie die Lambda-Funktion fürAWS IoT Greengrass
- Bereitstellen von Cloud-Konfigurationen auf einem Greengrass-Core-Gerät
- Überprüfen, ob die Lambda-Funktion auf dem Core-Gerät ausgeführt wird

Erstellen und Verpacken einer Lambda-Funktion

Die -Python-Lambda-Funktion in diesem Modul verwendetAWS IoT GreengrassCore-SDK, um MQTT-Nachrichten zu veröffentlichen.

In diesem Schritt führen Sie folgende Aktionen aus:

- Herunterladen desAWS IoT GreengrassCore-SDK für Ihren Computer (nicht dasAWS IoT GreengrassCore-Gerät).

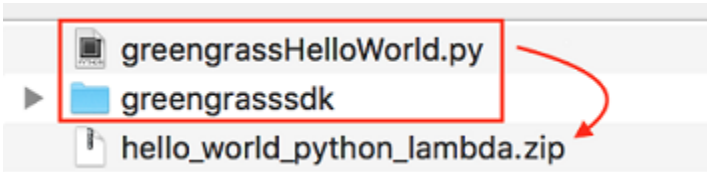
- Erstellen Sie ein Bereitstellungspaket für die Lambda-Funktion, das den Funktionscode und Abhängigkeiten
- Erstellen Sie über die Lambda-Konsole eine Lambda-Funktion und laden Sie das Bereitstellungspaket hoch.
- Veröffentlichen Sie eine Version der Lambda-Funktion und erstellen Sie einen Alias, der auf die Version zeigt.

Um dieses Modul abschließen zu können, muss Python 3.7 auf Ihrem Core-Gerät installiert sein.

1. FromAWS IoT GreengrassCore-SDKDownloads-Seite, laden Sie dieAWS IoT GreengrassCore SDK für Python auf Ihrem Computer.
2. Entpacken Sie das heruntergeladene Paket, um den Lambda-Funktionscode und das SDK zu erhalten.

   Die Lambda-Funktion in diesem Modul verwendet Folgendes:

   - Die Datei `greengrassHelloWorld.py` in `examples\HelloWorld`. Das 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 Bereitstellungspaket für die Lambda-Funktion zu erstellen,`greengrassHelloWorld.py`und die`greengrasssdk`Ordner in einen 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.

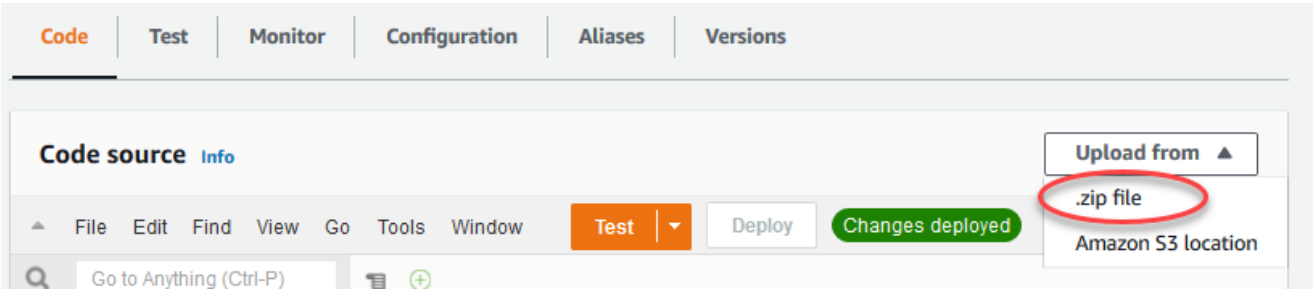
Nun können Sie Ihre Lambda-Funktion erstellen und das Bereitstellungspaket hochladen.

5. Öffnen Sie die Lambda-Konsole und wählen SieErstellen einer -Funktion.
6. Wählen Sie Author from scratch 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.
   - FürBerechtigungenbehalten Sie die Standardeinstellung bei. Dadurch wird eine Ausführungsrolle erstellt, die grundlegende Lambda-Berechtigungen gewährt. Diese Rolle wird nicht vonAWS IoT Greengrass.

   Wählen Sie Create function (Funktion erstellen).
8. Laden Sie das Bereitstellungspaket Ihrer Lambda-Funktion hoch:

   a. Auf derCodeunterCode-Quelle, wählenHochladen von. Wählen Sie in der Dropdown-ListeZIP-Datei.



   b. Klicken Sie aufHochladenund wählen Sie`hello_world_python_lambda.zip`Bereitstellungspaket. Wählen Sie dann Save (Speichern) aus.

c. Auf derCodefür die Funktion, unterRuntime Einstellungen, wählenBearbeiten, und geben Sie die folgenden Werte ein.

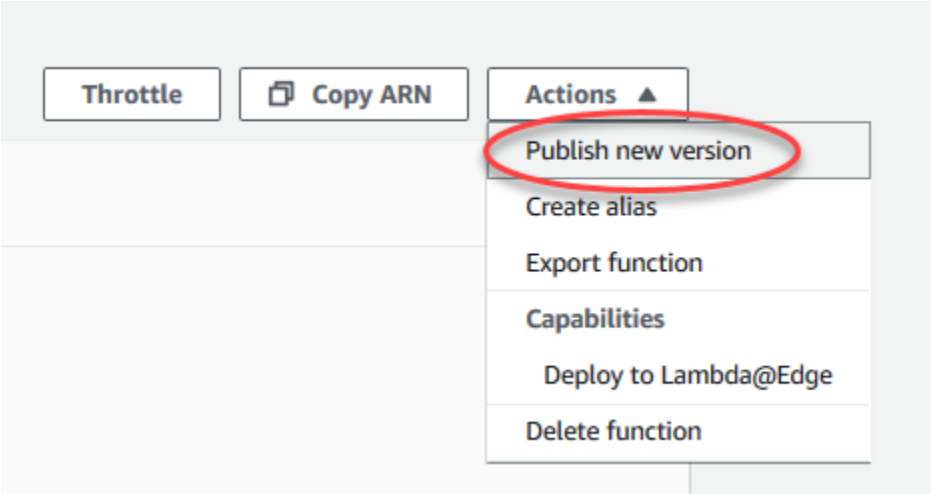
- Wählen Sie für Runtime (Laufzeit) die Option Python 3.7 aus.
- Geben Sie unter Handler **`greengrassHelloWorld.function_handler`** ein.

d. Wählen Sie Save (Speichern) aus.

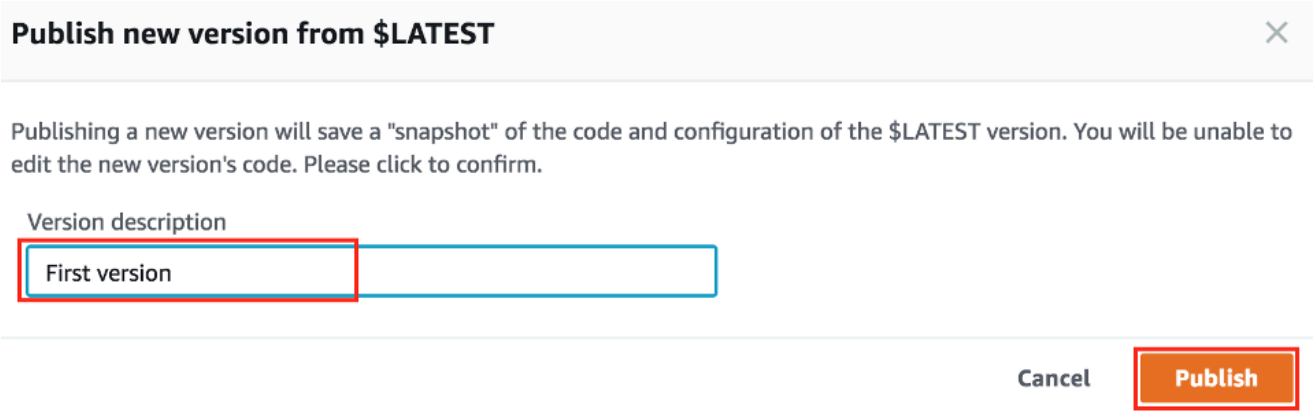
> **Note**
>
> DieTestaufAWS Lambda-Konsole funktioniert nicht mit dieser -Funktion. DieAWS IoT GreengrassDas Core SDK enthält keine Module, die erforderlich sind, um Ihre Greengrass Lambda-Funktionen unabhängig imAWS Lambdaconsole. Diese Module (zum Beispiel`greengrass_common`) werden für die Funktionen bereitgestellt, nachdem sie in Ihrem Greengrass-Kern bereitgestellt wurden.

9. Veröffentlichen der Lambda-Funktion.

a. FromAktionenWählen Sie im oberen Bereich der SeiteEine neue Version veröffentlichen.



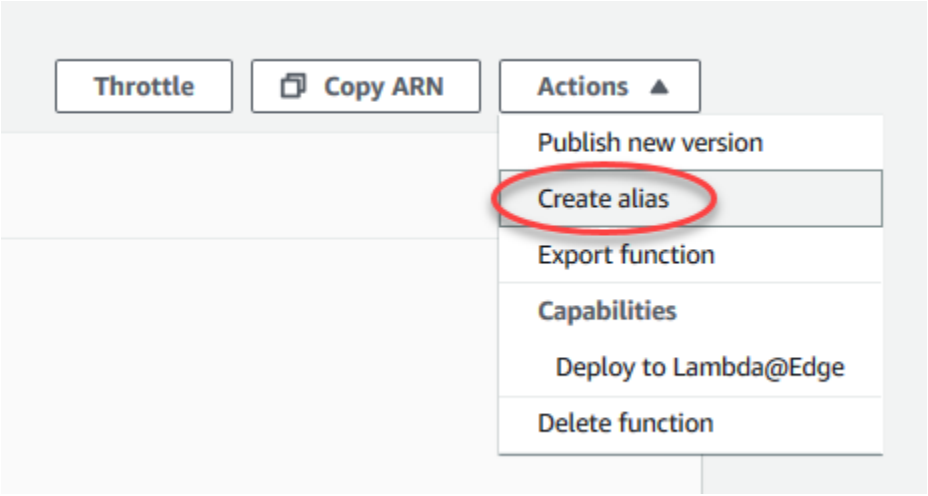
b. Geben Sie unter Version description (Versionsbeschreibung) den Wert **`First version`** ein und wählen Sie dann Publish (Veröffentlichen) aus.



10. Erstellen einesAliasfür die Lambda-FunktionAusführung:

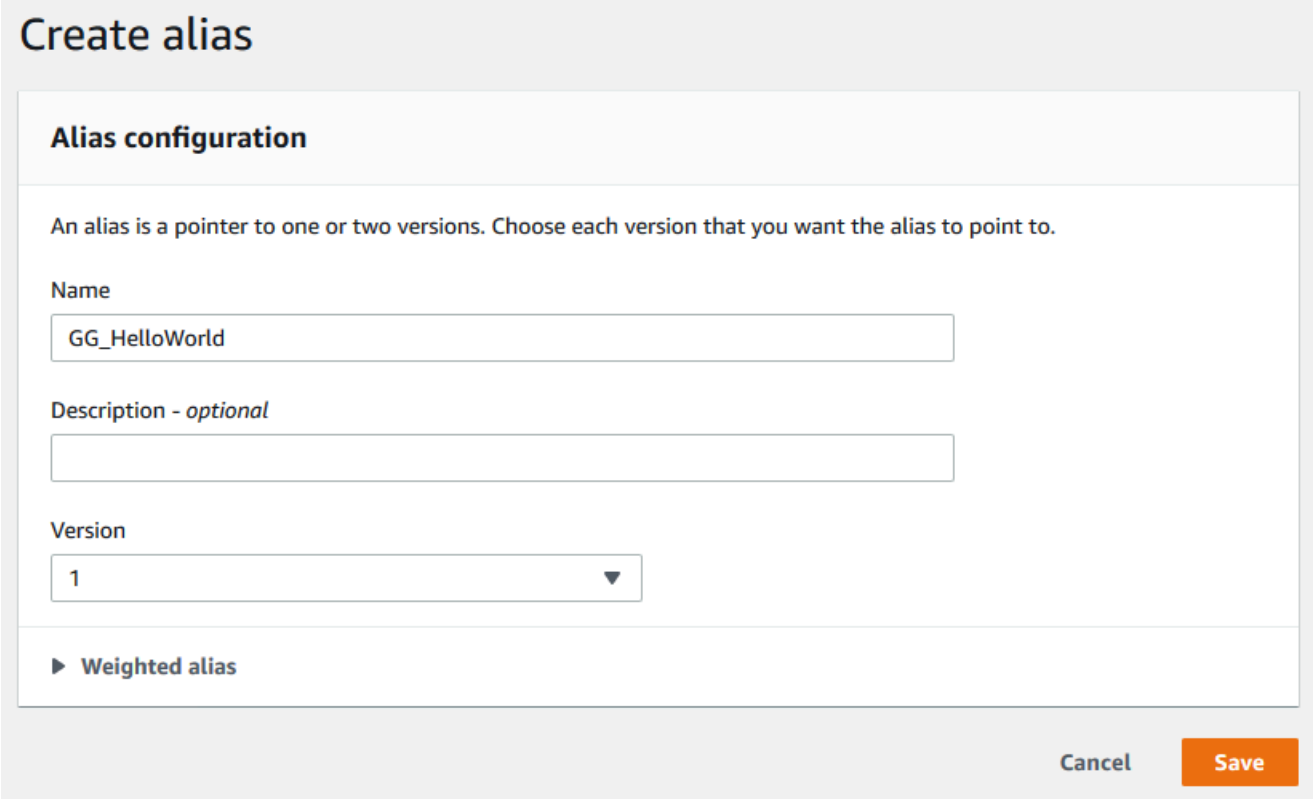
> **Note**
>
> Greengrass-Gruppen können eine Lambda-Funktion per Alias (empfohlen) oder nach Version referenzieren. Mit einem Alias lassen sich Code-Updates einfacher verwalten, da die Abonnementtabelle oder Gruppendefinition nicht geändert werden muss, wenn der Funktionscode aktualisiert wird. Stattdessen verweisen Sie einfach den Alias auf die neue Funktionsversion.

a. FromAktionenWählen Sie im oberen Bereich der SeiteErstellen eines Alias.



b. Geben Sie dem Alias**`GG_HelloWorld`**, setze die Version auf**1**(entsprechend der soeben veröffentlichten Version) und wählen SieSave.

> **Note**
>
> AWS IoT Greengrassunterstützt keine Lambda-Aliassen$LATESTVersionen.



Konfigurieren Sie die Lambda-Funktion fürAWS IoT Greengrass

Sie sind nun bereit zum Konfigurieren Ihrer Lambda-Funktion fürAWS IoT Greengrass.

In diesem Schritt führen Sie folgende Aktionen aus:

- Verwenden derAWS IoT-Konsole, um die Lambda-Funktion zur Greengrass-Gruppe hinzuzufügen.
- Konfigurieren Sie gruppenspezifische Einstellungen für die Lambda-Funktion.
- Erstellen eines Abonnements für die Gruppe, das es der Lambda-Funktion ermöglicht, MQTT-Nachrichten in zu veröffentlichenAWS IoT.
- Konfigurieren Sie lokale Protokolleinstellungen für die Gruppe.

1. In derAWS IoTNavigationsbereich der -Konsole, unterVerwalten, erweitern SieGreengrass-Geräteund anschließend aus.Gruppen (V1).
2. UnderGreengrass Gruppendie -Gruppe, die Sie in erstellt habenModul 2.
3. Wählen Sie auf der Gruppenkonfigurationsseite die Option aus.Lambda-Funktionenund scrollen Sie dann nach unten zumMeine Lambda-Funktionenund wählen Sie aus.Hinzufügen einer Lambda-Funktion.
4. Wählen Sie den Namen der Lambda-Funktion, die Sie im vorherigen Schritt erstellt haben (Greengrass_HelloWorld, nicht der Aliasname).
5. Wählen Sie für die VersionAlias: GG_HelloWorld.
6. In derKonfiguration von Lambda-Funktionendie folgenden Änderungen durch:
   - Legen Sie den Wert fürSystembenutzer und GruppezuGruppenstandard verwenden.
   - Legen Sie den Wert fürContainerisierung von Lambda-FunktionenzuGruppenstandard verwenden.
   - Legen Sie für Timeout 25 Sekunden fest. Diese Lambda-Funktion geht vor jedem Aufruf 5 Sekunden in den Ruhezustand.
   - FürPinned, wählenWahr.

> **Note**
>
> EINlangdauerndem(oderfestgeheftet) Lambda-Funktion startet automatisch danachAWS IoT Greengrassstartet und läuft weiter in einem eigenen Container. Dies steht im Gegensatz zu einemauf NachfrageLambda-Funktion, die nach einem Aufruf startet und beendet wird, sobald keine weiteren Aufgaben auszuführen sind. Weitere Informationen finden Sie unter the section called “Lebenszyklus-Konfiguration”.

7. Klicken Sie aufHinzufügen einer Lambda-Funktionum Ihre Änderungen zu speichern. Weitere Informationen zu Lambda-Funktionseigenschaften finden Sie unterthe section called “Steuern der Ausführung der Greengrass-Lambda-Funktion”.

   Erstellen Sie als Nächstes ein Abonnement, das der Lambda-Funktion ermöglicht, zu sendenMQTTNachrichten anAWS IoT Core.

   Eine Greengrass Lambda-Funktion kann MQTT-Nachrichten mit folgenden Elementen austauschen:

   - 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. Quelle oder Ziel können ein Greengrass-Gerät, eine Lambda-Funktion, ein Konnektor, ein Gerätesatten oder seinAWS 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 Abonnementthemenfilter nicht mehr als ein einzelnes+Charakter in einem Thema. Der Themenfilter erlaubt nur einen einzigen#einen Charakter am Ende eines Themas.

DieGreengrass_HelloWorldDie Lambda-Funktion sendet nur Nachrichten an diehello/worldThema inAWS IoT Core, so dass Sie nur ein Abonnement von der Lambda-Funktion zum erstellen müssenAWS IoT Core. Sie erstellen dies im nächsten Schritt.

8. Wählen Sie auf der Gruppenkonfigurationsseite die Option aus.Abonnementsund anschließend die OptionAbo hinzufügen.

   Für ein Beispiel, das zeigt, wie Sie ein Abonnement mit demAWS CLIfinden Sie unter,create-subscription-definitionimAWS CLIBefehlsreferenz.

9. In derQuelldatentyp, wählenLambda-Funktionund für dieQuelle, wählenGreengrass_HelloWorld.
10. Für denZieltyp, wählenServiceund für dieZielausgewähltIoT Cloud.
11. FürThemenfilterden Wert ein**hello/world**und anschließend aus.Ein Abonnement erstellen.
12. Konfigurieren Sie die Einstellungen für die Protokollierung der Gruppe. Für dieses Tutorial konfigurieren SieAWS IoT Greengrass-Systemkomponenten und benutzerdefinierte Lambda-Funktionen zum Schreiben von Protokollen in das Dateisystem des Core-Geräts.

    a. Wählen Sie auf der Gruppenkonfigurationsseite die Option aus.Logs (Protokolle)Registerkarte.

    b. In derKonfiguration von lokalen ProtokollenAbschnitt wählen Sie aus.Bearbeiten.

    c. Auf derKonfiguration von lokalen Protokollen bearbeitendie Standardwerte für Protokollebenen und Speichergrößen bei, und wählen Sie dannSave.

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 „Gruppeneinstellungen“ die OptionLambda-FunktionenRegisterkarte.

b. Zeigen Sie unter dem Verzeichnis die folgenden Dateien an.System Lambda-FunktionenAbschnitt, wählen SieStream-Managerund wähleBearbeiten.

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 auf Ihrem Core-Geräteterminal die folgenden Befehle aus, um zu prüfen, ob der Daemon ausgeführt wird, und starten Sie ihn gegebenenfalls.

   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. So starten Sie den Daemon:

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

Jetzt können Sie die Lambda--Funktion und die entsprechende Abonnementkonfigurationen auf Ihrem Greengrass-Core-Gerät bereitstellen.

3. In derAWS IoTNavigationsbereich der -Konsole, unterVerwalten, erweiternGreengrass-Geräteund wählen Sie dann aus.Gruppen (V1)aus.
4. UnderGreengrass Gruppendie -Gruppe, die Sie in erstellt habenModul 2aus.
5. Wählen Sie auf der Gruppenkonfigurationsseite die Option ausBereitstellenaus.
6. Auf derLambda-Funktionenauf der RegisterkarteLambda-Funktionenund wählen Sie aus.IP-Detektoraus.
7. Klicken Sie aufBearbeitenund wählen Sie ausAutomatische Erkennung und Überschreibung von MQTT-Broker-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 Problembehebung finden Sie unter Fehlerbehebung.

Überprüfen, ob die Lambda-Funktion auf dem Core-Gerät ausgeführt wird

1. Über den Navigationsbereich der RegisterkarteAWS IoTKonsole, unterTest, wählenMQTT-Test-Clientaus.
2. Wählen Sie das SymbolThema abonnierenRegisterkarte.
3. Geben Sie ein.**`hello/world`**in dieThemenfilterund erweitern Sie die DateiZusätzliche Konfigurationaus.
4. Geben Sie die Informationen ein, die in den folgenden Feldern aufgeführt sind:
   - Wählen Sie für Servicequalität 0 aus.
   - Wählen Sie für MQTT-Nutzlast-Anzeige die Option Nutzlasten als Zeichenfolgen anzeigen aus.
5. Wählen Sie Subscribe aus.

Wenn die Lambda-Funktion auf dem Gerät ausgeführt wird, veröffentlicht sie Nachrichten im Verzeichnis`hello/world`-Thema:

Obwohl die Lambda-Funktion weiterhin MQTT-Nachrichten an die Datei`hello/world`Thema, hör nicht aufAWS IoT GreengrassDaemon. Bei den verbleibenden Modulen wird nämlich davon ausgegangen, dass die Funktion ausgeführt wird.

Sie können die Funktion und das Abonnement aus der Gruppe löschen:

- Auf der Gruppenkonfigurationsseite unterLambda-Funktionendie Lambda-Funktion, die Sie entfernen möchten, und wählen SieRemoveaus.

- Auf der Gruppenkonfigurationsseite unterAbonnements, wählen Sie das Abonnement und anschließend die OptionLöschenaus.

Die Funktion und das Abonnement werden während der nächsten Gruppenbereitstellung vom Kern entfernt.

Modul 3 (Teil 2)# Funktionen von Lambda-Funktionen aufAWS IoT Greengrass

Dieses Modul untersucht die Unterschiede zwischen On-Demand- und langlebigen Lambda-Funktionen, die auf der ausgeführt werdenAWS IoT Greengrasscore.

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

Erstellen und Verpacken der Lambda-Funktion

In diesem Schritt führen Sie folgende Aktionen aus:

- Erstellen Sie ein Bereitstellungspaket, das den Funktionscode und die Abhängigkeiten enthält.
- Erstellen Sie über die Lambda-Konsole eine Lambda-Funktion, und laden Sie das Bereitstellungspaket hoch.
- Veröffentlichen Sie eine Version der Lambda-Funktion und erstellen Sie einen Alias, der auf die Version zeigt.

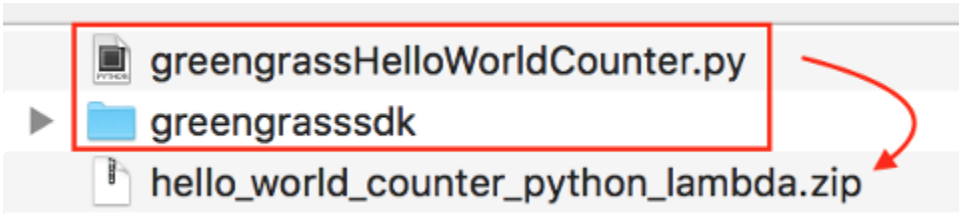
1. Gehen Sie auf Ihrem Computer dieAWS IoT GreengrassCore SDK für Python, das Sie heruntergeladen und extrahiert habenthe section called “Erstellen und Verpacken einer Lambda-Funktion”in Modul 3-1.

   Die Lambda-Funktion in diesem Modul verwendet Folgendes:

   - Die Datei `greengrassHelloWorldCounter.py` in `examples\HelloWorldCounter`. Das ist Ihr Lambda-Funktionscode.
   - Der Ordner `greengrasssdk`. Dies ist das SDK.
2. Erstellen Sie ein Bereitstellungspaket der Lambda-Funktion

   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.

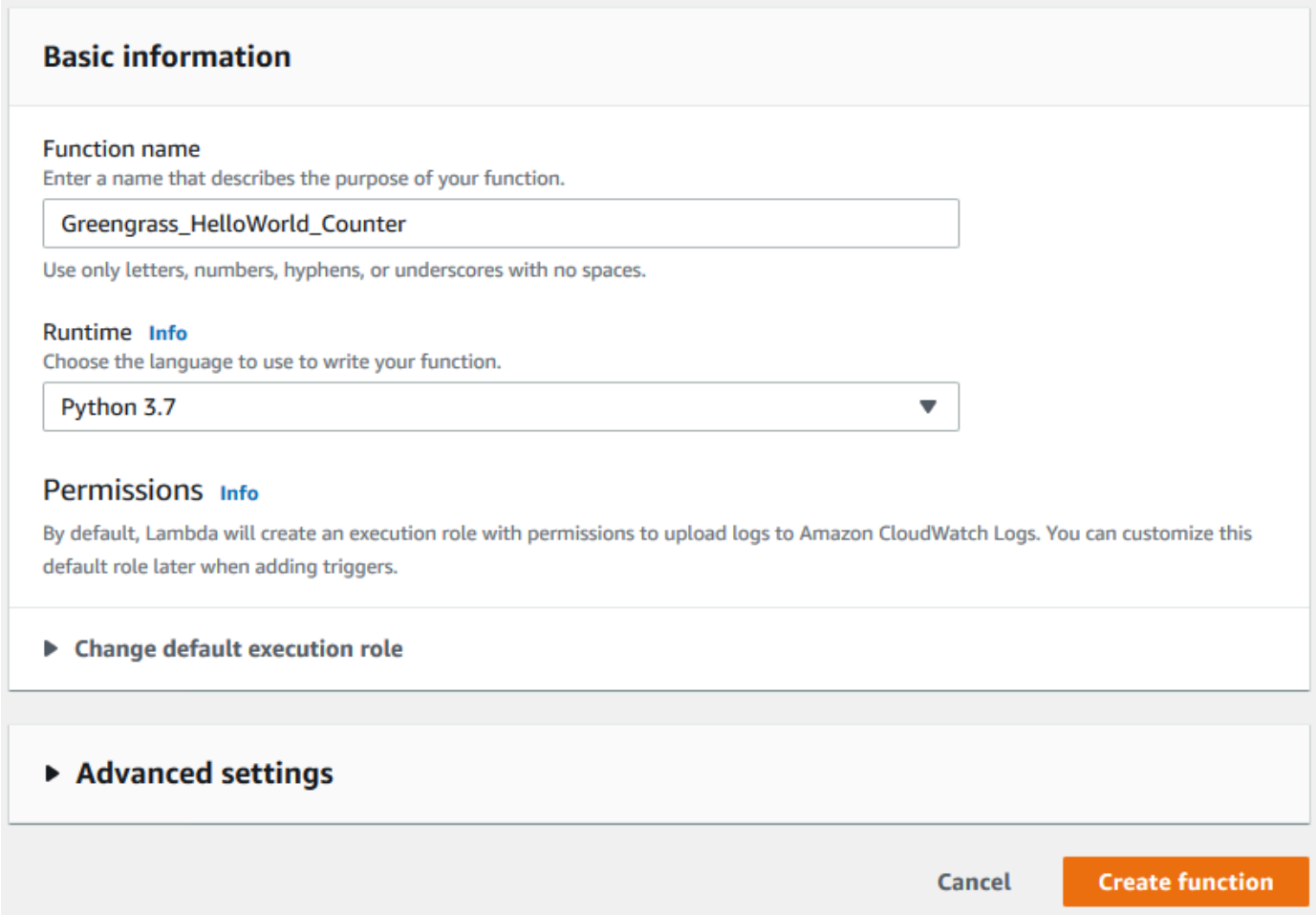
   Auf UNIX-ähnlichen Systemen (einschließlich des Mac-Terminals), auf denen `zip` installiert ist, können Sie den folgenden Befehl verwenden, um die Datei und den Ordner zu verpacken:

   ```
   zip -r hello_world_counter_python_lambda.zip greengrasssdk
    greengrassHelloWorldCounter.py
   ```

   Nun können Sie Ihre Lambda-Funktion erstellen und das Bereitstellungspaket hochladen.
3. Öffnen Sie die Lambda-Konsole und wählen SieFunktion erstellen.
4. Wählen Sie Author from scratch aus.
5. Geben Sie Ihrer Funktion den Namen **`Greengrass_HelloWorld_Counter`** und richten Sie die verbleibenden Felder wie folgt ein:

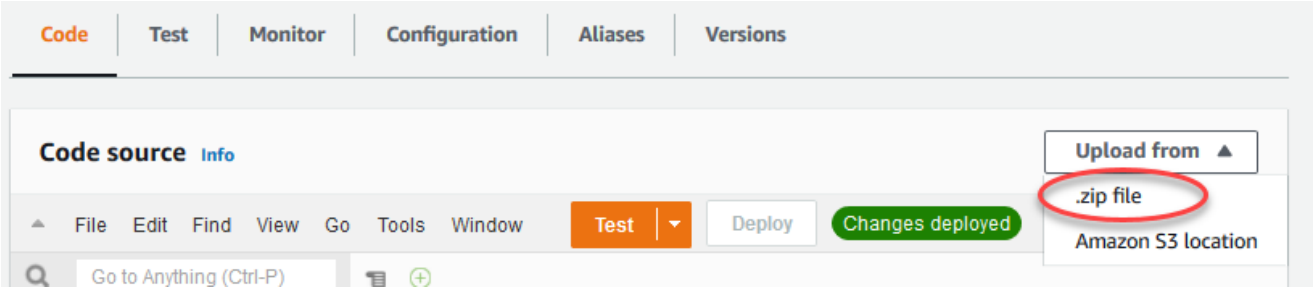
- Wählen Sie für Runtime (Laufzeit) die Option Python 3.7 aus.
- FürBerechtigungen, behalten Sie die Standardeinstellung bei. Dadurch wird eine Ausführungsrolle erstellt, die grundlegende Lambda-Berechtigungen gewährt. Diese Rolle wird nicht verwendet vonAWS IoT Greengrass. Sie können stattdessen auch die Rolle wiederverwenden, die Sie in Modul 3-1 erstellt haben.

Wählen Sie Create function (Funktion erstellen).



6. Laden Sie das Bereitstellungspaket Ihrer Lambda-Funktion

   a. Auf derCodeunterQuellcode, wählenHochladen von. Wählen Sie in der Dropdown-ListeZIP-Datei.



b. Klicken Sie aufHochladenund dann wählen Sie Ihre`hello_world_counter_python_lambda.zip`Bereitstellungspaket. Wählen Sie dann Save (Speichern) aus.

c. Auf derCodefür die Funktion, unterLaufzeitumgebung, wählenBearbeiten, und geben Sie 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**
>
> DieTest-Schaltfläche auf derAWS Lambda-Konsole funktioniert nicht mit dieser Funktion. DieAWS IoT GreengrassDas Core SDK enthält keine Module, die erforderlich sind, um Ihre Greengrass Lambda-Funktionen unabhängig imAWS Lambdaconsole. Diese Module (zum Beispiel`greengrass_common`) werden für die Funktionen bereitgestellt, nachdem sie in Ihrem Greengrass-Kern bereitgestellt wurden.

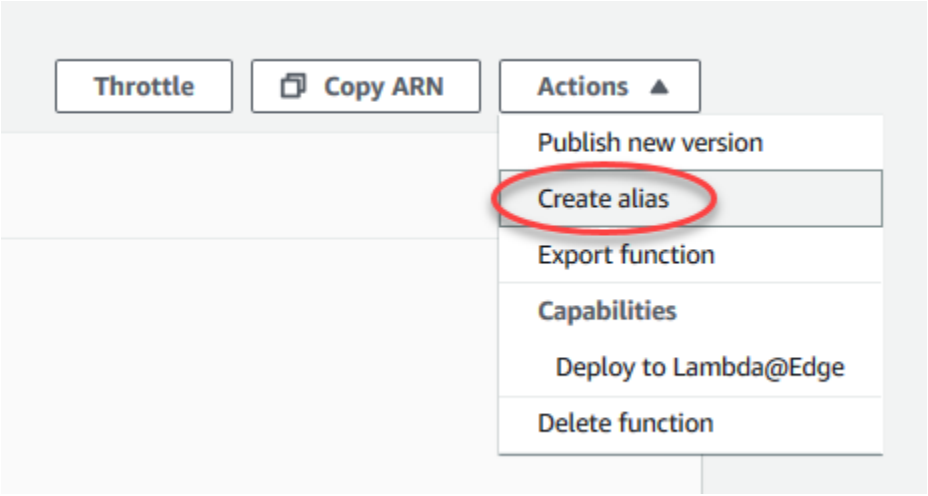
7. Veröffentlichen Sie die erste Version der -Funktion.

a. Aus demAktionen-Menü oben auf der Seite, wählen SieEine neue Version veröffentlichen. Geben Sie für Version description (Versionsbeschreibung) **`First version`** ein.

b. Wählen Sie Publish.

8. Erstellen Sie einen Alias für die Funktionsversion.

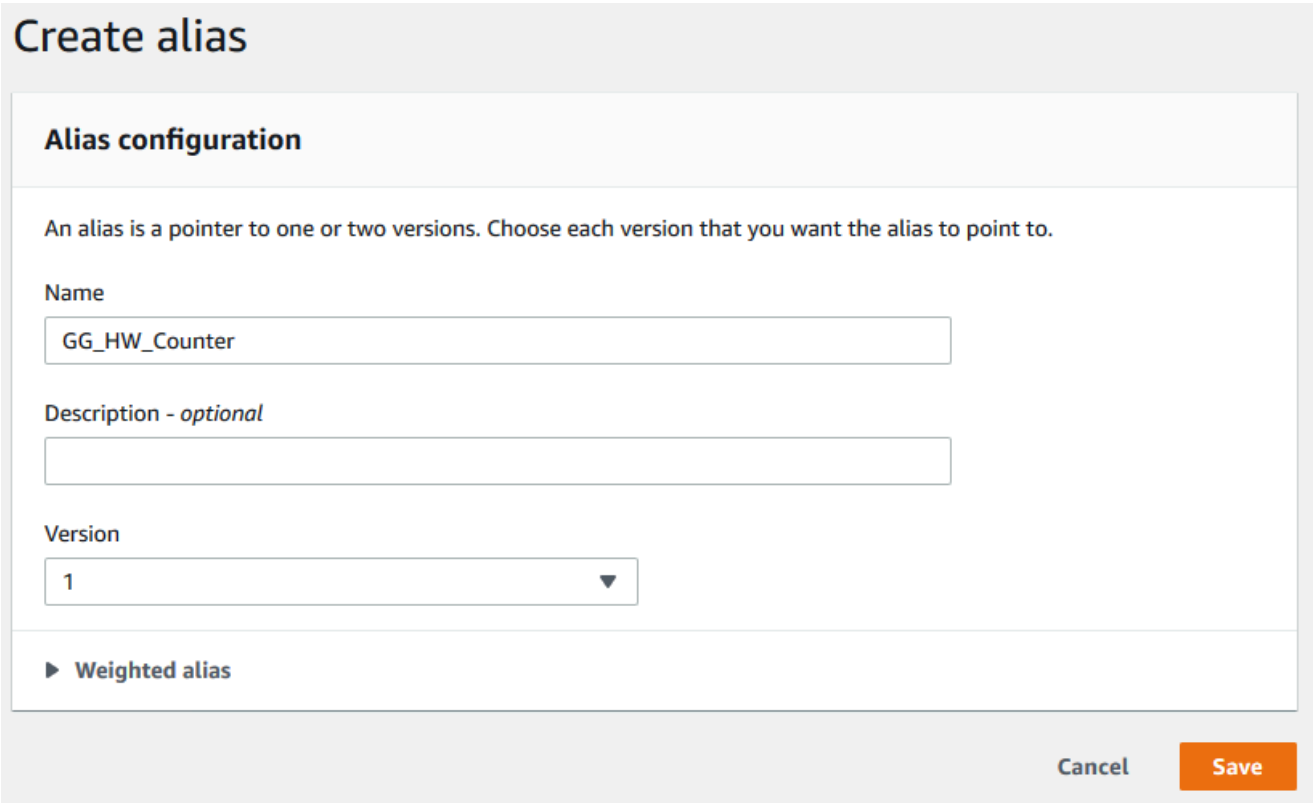
a. Aus demAktionen-Menü oben auf der Seite, wählen SieErstellen eines Alias.



b. Geben Sie unter Name **`GG_HW_Counter`** ein.

c. Wählen Sie für Version die Option 1.

d. Wählen Sie Save (Speichern) aus.



Aliase erstellen eine einzelne Entity für Ihre Lambda-Funktion, die GreenGrass-Geräte abonnieren können. So müssen Sie Abonnements nicht jedes Mal mit neuen Lambda-Funktionsversionsnummern aktualisieren, wenn die Funktion geändert wird.

Konfigurieren langlebiger Lambda-Funktionen fürAWS IoT Greengrass

Sie können jetzt Ihre Lambda--Funktion für konfigurierenAWS IoT Greengrass.

1. In derAWS IoTNavigationsbereich der -Konsole, unterVerwalten, erweiternGreengrass Geräteund wählen Sie dannGruppen (V1).
2. UnderGreengrass GruppenWählen Sie die -Gruppe, die Sie in erstellt habenModul 2.
3. Wählen Sie auf der Gruppenkonfigurationsseite die OptionLambda-Funktionenund dann unterMeine Lambda-Funktionen, wählenAdd.
4. FürLambda-Funktion, wählenGreengrass_HelloWorld_Theke.
5. FürVersioning der Lambda-Funktion, wählen Sie den Alias für die Version, die Sie veröffentlicht haben.
6. FürTimeout (seconds) (Timeout (Sekunden))den Wert ein**25**. Diese Lambda--Funktion geht vor jedem Aufruf 20 Sekunden in den Ruhezustand.
7. FürPinned, wählenWahr.
8. Übernehmen Sie die Standardwerte für alle anderen Felder und wählen SieLambda-Funktion hinzufügen.

Testen von langdauernden -Funktionen

EINlangdauerndenDie Lambda-Funktion startet automatisch, wenn dieAWS IoT Greengrasscore wird in einem einzelnen Container (oder einer Sandbox))) ausgeführt. 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 anzeigenAWS Lambda-Konsole oder in derAWS IoT GreengrassCores SDK für Pythonauf GitHubaus.

In diesem Schritt erstellen Sie Abonnements, die Lambda-Funktion undAWS IoTum MQTT-Nachrichten auszutauschen. Im Anschluss daran stellen Sie die Gruppe bereit und testen die Funktion.

1. Wählen Sie auf der Gruppenkonfigurationsseite die OptionAbonnementsund wählen Sie dann ausAddaus.
2. UnderRessourcentyp, wählenLambda-Funktionund wählen Sie dann ausGreengrass_HelloWorld_Thekeaus.
3. UnderZieltyp, wählen-Service, wählenIoT Cloudaus.
4. Geben Sie für Topic filter (Themenfilter) die Zeichenfolge **`hello/world/counter`** ein.
5. Wählen Sie Create subscription (Abonnement erstellen) aus.

   Dieses einzelne Abonnement betrifft nur eine Richtung: vom`Greengrass_HelloWorld_Counter`Lambda-Funktion zumAWS IoTaus. Um diese Lambda-Funktion über die Cloud aufrufen (oder auslösen) zu können, müssen Sie ein Abonnement in der entgegengesetzten Richtung erstellen.
6. Befolgen Sie die Schritte 1—5, um ein weiteres Abonnement hinzuzufügen, das die folgenden Werte verwendet. Dadurch können Sie die Lambda-Funktion für den Empfang von Nachrichten vom einrichtenAWS IoTaus. Sie verwenden dieses Abonnement, wenn Sie eine Nachricht vomAWS IoT-Konsole, von der die Funktion aufgerufen wird.

   - Wählen Sie für die Quelle-Serviceund wählen Sie dann ausIoT Cloudaus.
   - Wählen Sie für das ZielLambda-Funktionund wählen Sie dann ausGreengrass_HelloWorld_Thekeaus.
   - 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, wie in beschrieben, ausgeführt wirdBereitstellen von Cloud-Konfigurationen für ein Core-Gerätaus.
8. Wählen Sie auf der Gruppenkonfigurationsseite die OptionBereitstellenaus.
9. Nachdem Ihre Bereitstellung abgeschlossen wurde, kehren Sie zumAWS IoT-Konsolen-Startseite und wählen SieTestaus.
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.

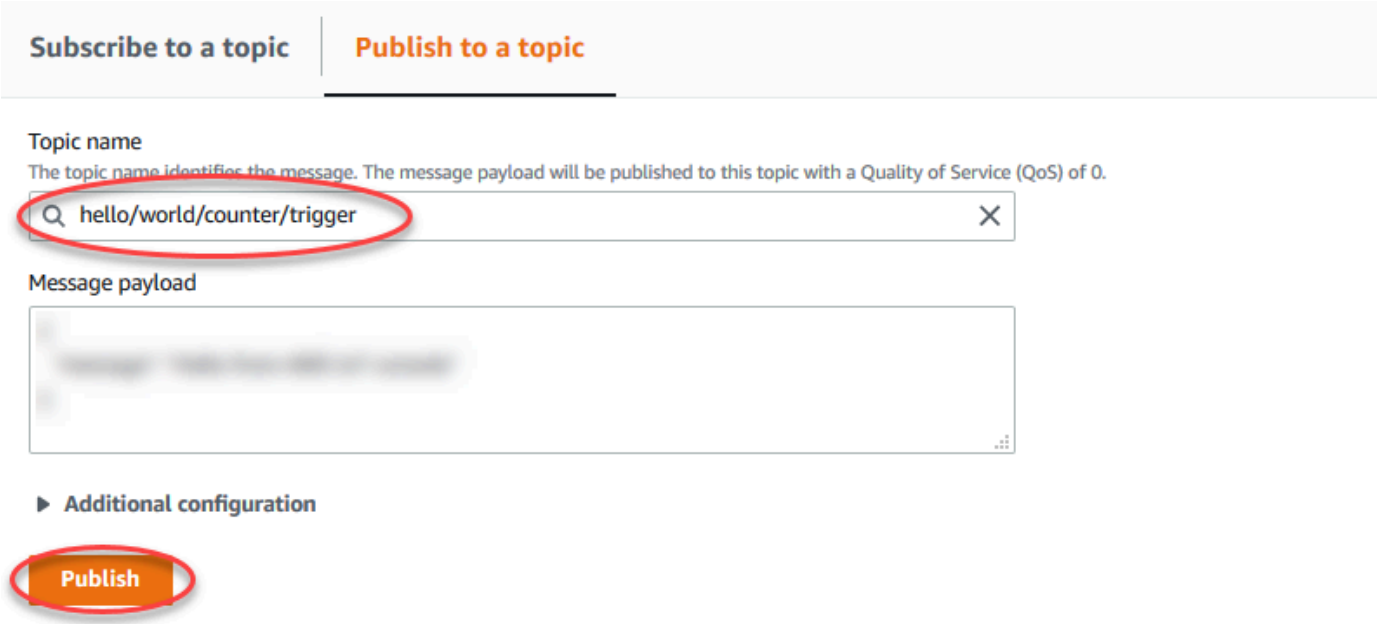
11. Wählen Sie Subscribe aus.

    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, die aufgerufen wird, wenn sie eine MQTT-Nachricht auf dem`hello/world/counter/trigger`-Thema.

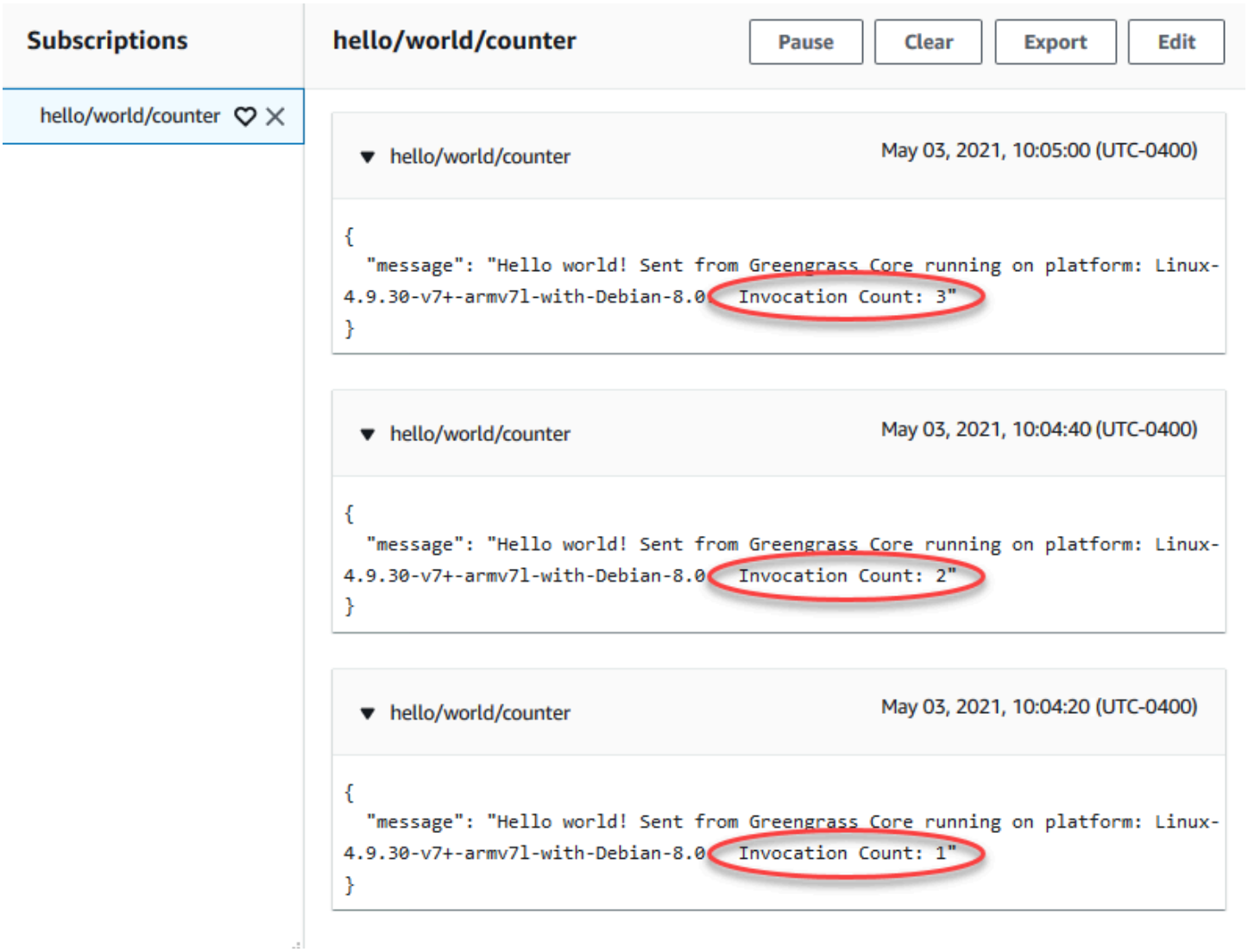
    DieGreengrass_HelloWorld_ThekezuIoT Cloud-Funktionen ermöglichen die -Funktion, Nachrichten an zu sendenAWS IoTauf der`hello/world/counter`-Thema. DieIoT CloudzuGreengrass_HelloWorld_ThekeAbonnement erlaubtAWS IoTum Meldungen an die -Funktion auf dem`hello/world/counter/trigger`-Thema.

12. Um den langdauernden Lebenszyklus zu überprüfen, rufen Sie die Lambda-Funktion auf, indem Sie eine Nachricht im`hello/world/counter/trigger`-Thema. Sie können die Standardnachricht verwenden.



> **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 in derAWS IoT GreengrassCores SDK für Pythonauf GitHubaus.

Jedes Mal, wenn eine Nachricht im Thema `hello/world/counter/trigger` veröffentlicht wird, wird die `my_counter`-Variable um einen Schritt erhöht. Die Anzahl der Aufrufe wird in den von der Lambda-Funktion gesendeten Nachrichten angegeben. Weil der Funktionshandler einen 20-sekündigen Ruhezyklus beinhaltet (`time.sleep(20)`), wobei der Handler wiederholt die Antworten vom in eine Warteschlange gestellt werdenAWS IoT GreengrassKern.

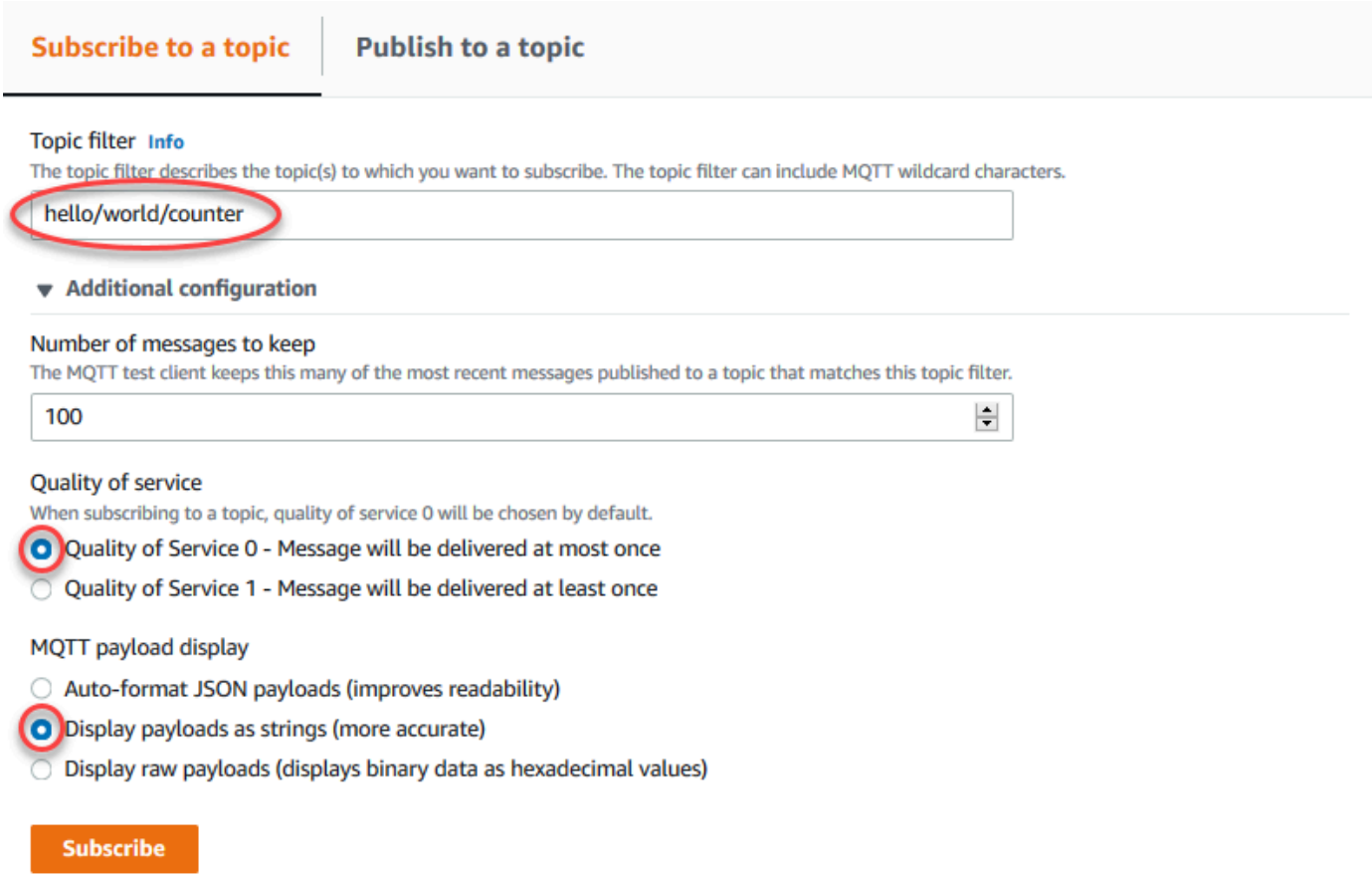


Testen von On-Demand-Funktionen

Importieren in &S3;on-ModusDie Lambda-Funktion entspricht funktional einer Cloud-basiertenAWS LambdaFunktion. Es können mehrere Aufrufe einer On-Demand-Lambda-Funktion parallel ausgeführt werden. Ein Aufruf der Lambda-Funktion erstellt einen separaten Container für die Verarbeitung der Aufrufe oder verwendet einen vorhandenen Container neu, wenn 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 OptionLambda-FunktionenRegisterkarte.
2. UnderMeine Lambda-Funktionen, wählen SieGreengrass_HelloWorld_CounterLambda-Funktion
3. Auf derGreengrass_HelloWorld_CounterDetailseite wählenBearbeitenaus.
4. FürPinned, wählenFalschund wählen Sie dannSaveaus.

5. Wählen Sie auf der Gruppenkonfigurationsseite die OptionBereitstellenaus.
6. Nachdem Ihre Bereitstellung abgeschlossen wurde, kehren Sie zuAWS IoTKonsolen-Startseite und wählenTestaus.
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.

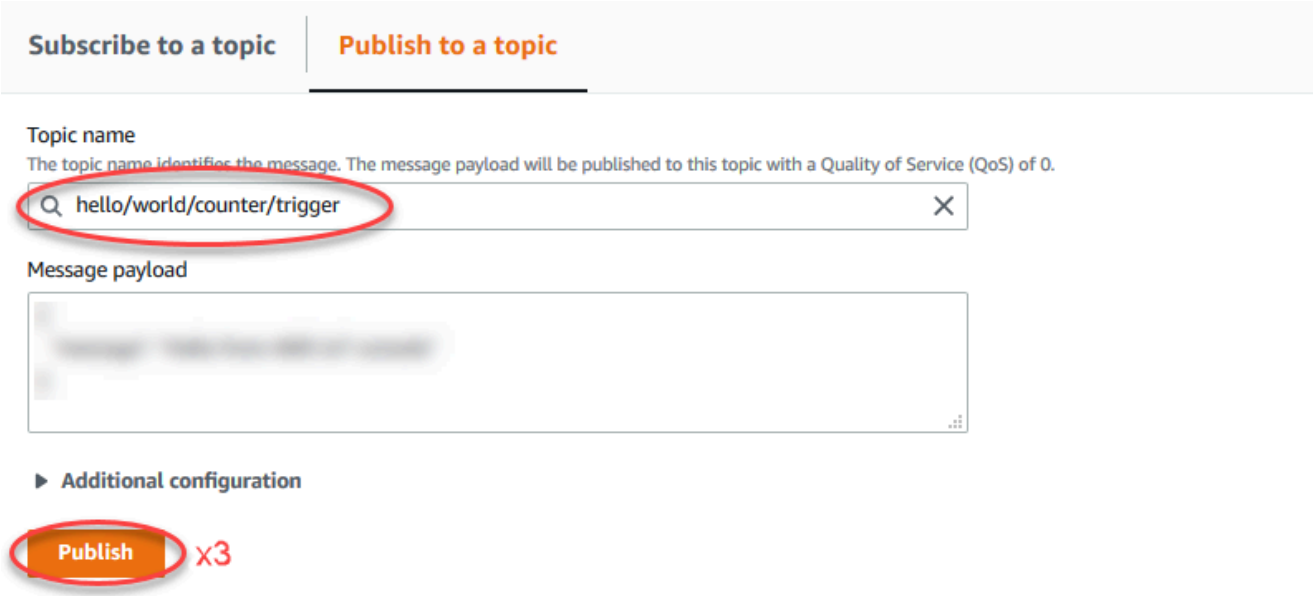


8. Wählen Sie Subscribe aus.

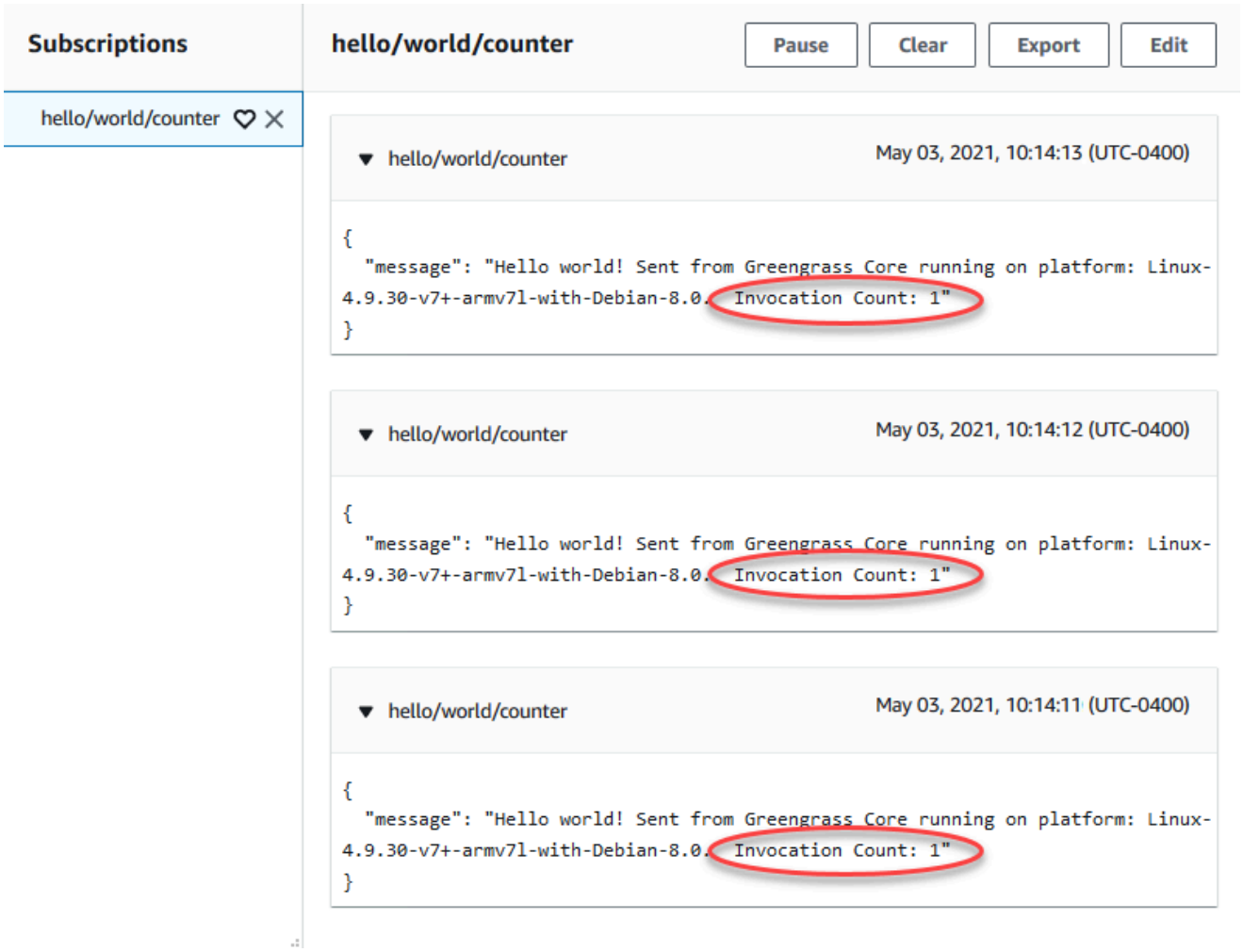
> **Note**
>
> Sie sollten nach dem Abonnement keine Nachrichten sehen.

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.

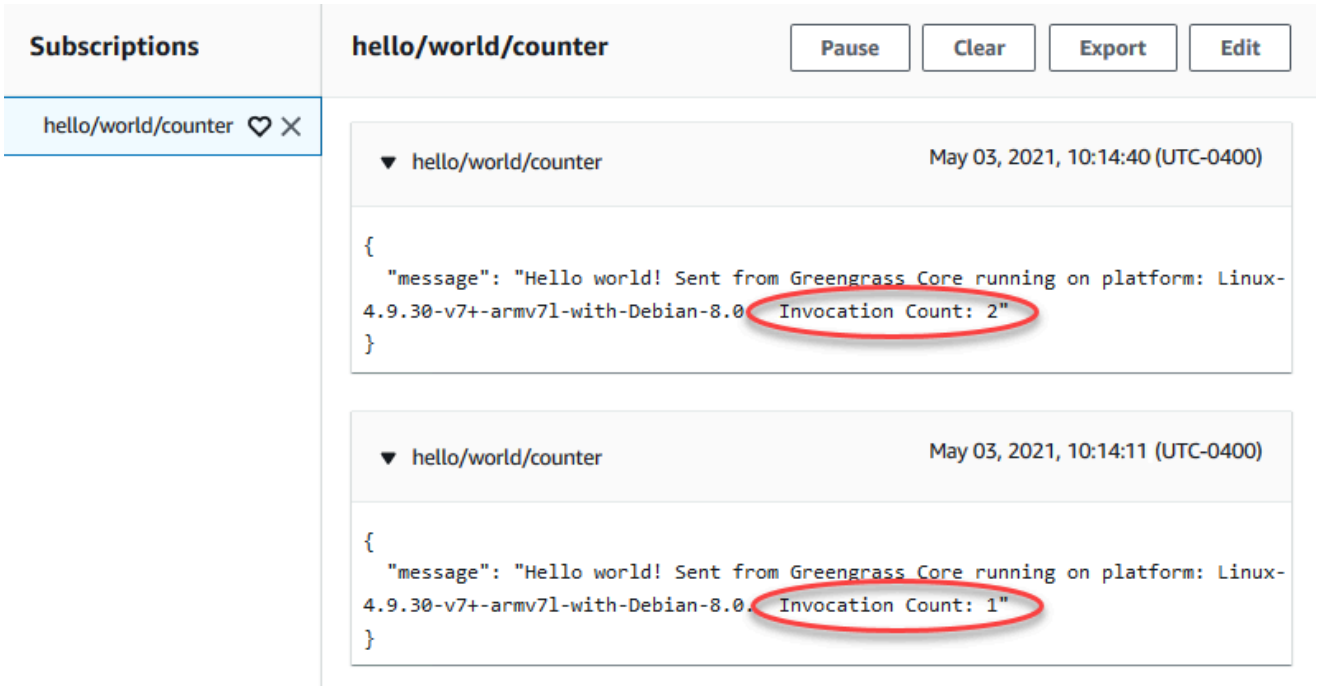
   a. Klicken Sie aufVeröffentlichendreimal hintereinander, wobei zwischen jedem Drücken der Taste nur fünf Sekunden liegen sollten.



   Jede Veröffentlichung ruft den Funktionshandler auf und erstellt einen neuen Container für jeden Aufruf. Die Zahl der Aufrufe wird durch das dreimalige Auslösen der Funktion nicht erhöht, da jede On-Demand-; -Funktion über einen eigenen Container bzw. eine eigene Sandbox verfügt.



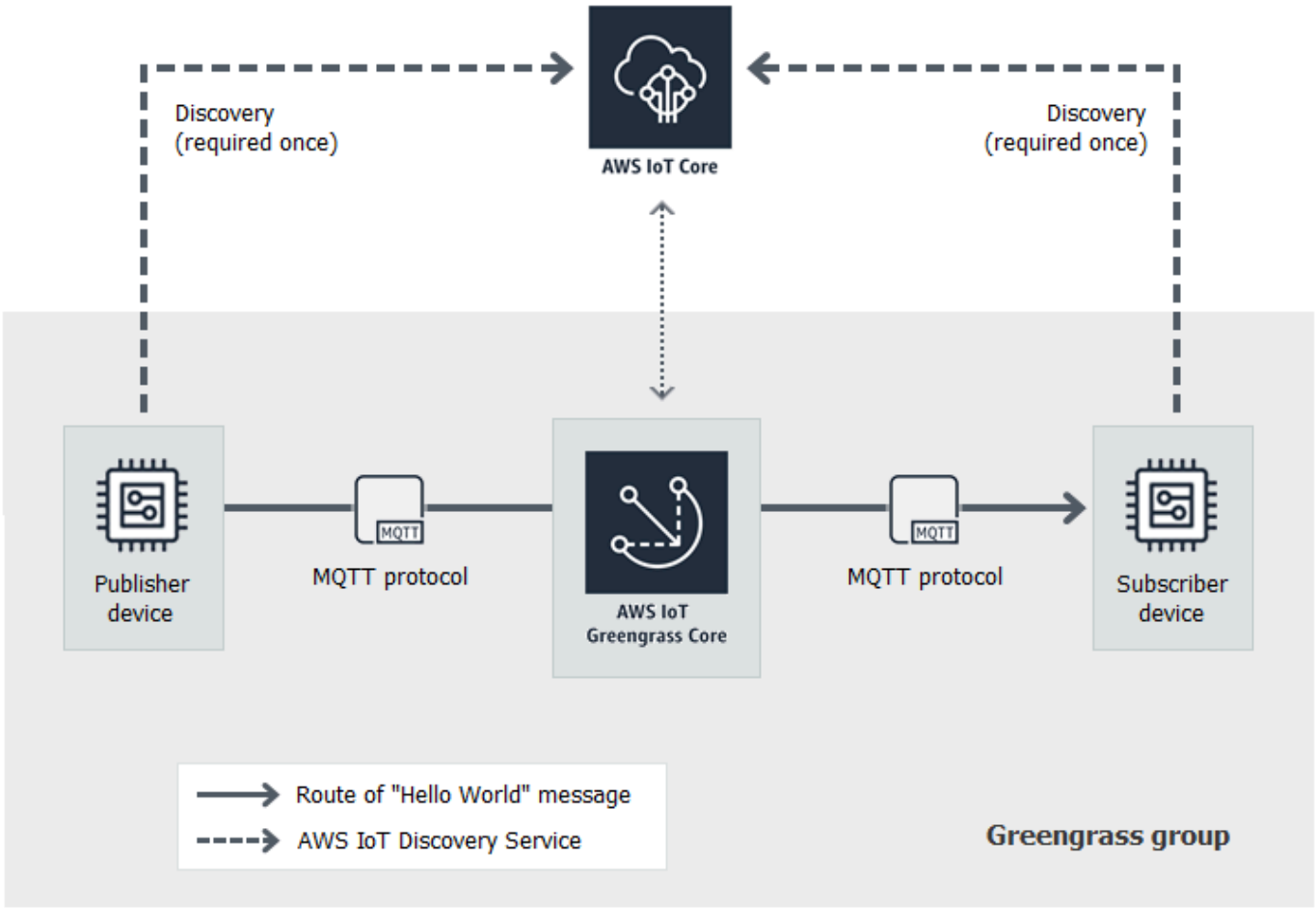
b. Wählen Sie nach ungefähr 30 Sekunden die Option Publish to topic (Veröffentlichung im Thema) aus. Der Zahl der Aufrufe sollte sich auf 2 erhöhen. Dies zeigt, dass ein Container, der durch einen vorherigen Aufruf erstellt wurde, erneut verwendet wird, und Variablen, die außerhalb des Funktionshandlers vorab verarbeitet wurden, gespeichert wurden.



Sie sollten nun die zwei Arten von Lambda-Funktionen, die auf der ausgeführt werden können, verstanden habenAWS IoT GreengrassKern. Das nächste Modul,Modul 4, zeigt Ihnen, wie lokale IoT-Geräte in einerAWS IoT GreengrassGruppe.

Modul 4: Interagieren mit Client-Geräten in einemAWS IoT GreengrassGruppe

Dieses Modul zeigt Ihnen, wie lokale IoT-Geräte, genanntClient-GeräteoderGeräte, eine Verbindung zu einerAWS IoT Greengrass-Core-Gerät. Client-Geräte, die eine Verbindung zu einemAWS IoT Greengrasscore sind Teil einerAWS IoT GreengrassGruppe und kann an der teilnehmenAWS IoT GreengrassProgrammierparadigma 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

- Erstellen Sie Client-Geräte in einerAWS IoT GreengrassGruppe
- Konfigurieren von Abonnements
- Installieren desAWS IoT Device SDKfor Python
- Testen der Kommunikation

Erstellen Sie Client-Geräte in einerAWS IoT GreengrassGruppe

In diesem Schritt fügen Sie Ihrer Greengrass-Gruppe zwei Client-Geräte hinzu. Dieser Vorgang beinhaltet die Registrierung der Geräte alsAWS IoTund Konfigurieren von Zertifikaten und SchlüsselnAWS IoT Greengrassaus.

1. In derAWS IoTNavigationsbereich der -Konsole, unterVerwalten,Greengrass-Geräteund wählen Sie dann ausGruppen (V1)aus.
2. Wählen Sie die Zielgruppe aus.
3. Klicken Sie auf der Gruppenkonfigurationsseite aufClient-Geräteund wählen Sie dann ausAssociateaus.
4. In derVerknüpfen Sie ein Client-Gerät mit dieser Gruppemodal, wählen SieCreate newAWS IoTSacheaus.

   DieErstellen von Objekten-Seite wird in einer neuen Registerkarte geöffnet.
5. Auf derErstellen von Objekten-SeiteEinzelnes Objektund wählen Sie dann ausWeiteraus.
6. Auf derDing-Eigenschaften angebenSeite, registrieren Sie dieses Client-Gerät als**`HelloWorld_Publisher`**und wählen Sie dann ausWeiteraus.
7. Auf derGerätezertifikat-SeiteWeiteraus.
8. Auf derRichtlinien an Zertifikateinen der folgenden Schritte aus:
   - Wählen Sie eine vorhandene Richtlinie aus, die Berechtigungen gewährt, die Client-Geräte benötigen, und wählen Sie dannCreate thingaus.

     Ein Modal wird geöffnet, in dem Sie die Zertifikate und Schlüssel herunterladen können, die das Gerät für die Verbindung mit demAWS Cloudund der Core.
   - Erstellen Sie eine neue Richtlinie, die Berechtigungen für Clientgeräte gewährt, und fügen Sie sie hinzu Gehen Sie wie folgt vor:

     a. Wählen Sie Create Policy (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. FürRichtliniennameeinen beschreibenden Namen für die Richtlinie ein, z. B.**`GreengrassV1ClientDevicePolicy`**aus.

        ii. Auf derRichtlinienanweisungunterRichtliniendokument, wählenJSONaus.

iii. Geben Sie das folgende -Richtliniendokument ein: Diese Richtlinie ermöglicht es dem Client-Gerät, Greengrass-Kerne zu erkennen und über alle MQTT-Themen zu kommunizieren. Weitere Informationen dazu, wie Sie den Zugriff dieser Richtlinie einschränken, finden Sie unterGeräteauthentifizierung und -autorisierung für AWS IoT Greengrassaus.

```
{
  "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 mit demRichtlinien an Zertifikat-Seite geöffnet. Gehen Sie wie folgt vor:

i. In derRichtliniendie von Ihnen erstellte Richtlinie aus, z. B.GreengrassV1ClientDevicePolicyaus.

Falls die Richtlinie nicht angezeigt wird, klicken Sie auf die Schaltfläche Refresh aus.

ii. Klicken Sie aufCreate thingaus.

Ein Modal wird geöffnet, in dem Sie die Zertifikate und Schlüssel herunterladen können, die das Gerät für die Verbindung mit demAWS Cloudund der Core.

9. In derLaden Sie Zertifikate und Schlüssel heruntermodal, laden Sie die Zertifikate des Geräts herunter.

> **Important**
>
> Bevor Sie sichFertigund laden Sie die Sicherheitsressourcen herunter.

Gehen Sie wie folgt vor:

a. FürGerätezertifikat, wählenHerunterladenum das Gerätezertifikat herunterzuladen.

b. FürDatei mit dem öffentlichen Schlüssel, wählenHerunterladenum den öffentlichen Schlüssel für das Zertifikat herunterzuladen.

c. FürPrivate Schlüsseldatei, wählenHerunterladenum die Datei mit dem privaten Schlüssel für das Zertifikat herunterzuladen.

d. PrüfenServerauthentifizierungimAWS IoTEntwicklerhandbuchund wählen Sie das entsprechende CA-Stammzertifikat aus. Wir empfehlen die Verwendung eines Amazon Trust Services (ATS) -CA-Stammzertifikats. UnderCA-Stammzertifikate, wählenHerunterladenfür ein -CA-Stammzertifikat.

e. Wählen Sie Done (Erledigt) aus.

Notieren Sie sich die in den Dateinamen für das Gerätezertifikat und die Schlüssel gemeinsame Zertifikatsnummer. Sie benötigen sie später.

10. Kehren Sie mit demVerknüpfen Sie ein Client-Gerät mit dieser Gruppemodal offen. Gehen Sie wie folgt vor:

a. FürAWS IoTThing-Name, wähle dasHelloWorld_Publisherwas du geschaffen hast.

Falls Sie das Objekt nicht sehen, wählen Sie die Aktualisierungsschaltfläche aus.

b. Wählen Sie Associate aus.

11. Wiederholen Sie die Schritte 3 bis 10, um der Gruppe ein zweites Client-Gerät hinzuzufügen.

Geben Sie diesem Client-Gerät**`HelloWorld_Subscriber`**aus. Laden Sie die Zertifikate und Schlüssel für dieses Client-Gerät auf Ihren Computer herunter. Notieren Sie sich wieder die gemeinsame ID des Zertifikats in den Dateinamen für die HelloWorld_Abonnenten-Gerät.

Sie sollten jetzt zwei Client-Geräte in Ihrer Greengrass-Gruppe haben:

- HelloWorld_Herausgeber
- HelloWorld_Abonnent

12. Erstellen Sie auf Ihrem Computer einen Ordner für die Sicherheitsanweisung dieser Client-Geräte. Kopieren Sie die Zertifikate und Schlüssel in diesen Ordner.

Konfigurieren von Abonnements

In diesem Schritt aktivieren Sie die Option HelloWorld_Publisher-Clientgerät zum Senden von MQTT-Nachrichten an HelloWorld_Abonnenten-Client-Gerät.

1. Wählen Sie auf der Gruppenkonfigurationsseite die OptionAbonnementsund wählen Sie anschließend ausAddaus.
2. Auf derErstellen eines Abonnements-Seite, machen Sie Folgendes, um das Abonnement zu konfigurieren:

   a. FürRessourcentyp, wählenClient-Gerätund wählen Sie anschließend ausHelloWorld_Herausgeberaus.

   b. UNDERZieltyp, wählenClient-Gerätund wählen Sie anschließend ausHelloWorld_Abonnentaus.

   c. Geben Sie für Topic filter (Themenfilter) die Zeichenfolge **`hello/world/pubsub`** ein.

   > **Note**
   >
   > Sie können Abonnements aus den früheren Modulen löschen. Auf der Seite der GruppeAbonnementsdie Abonnements aus, die gelöscht werden sollen, und klicken Sie anschließendLöschenaus.

   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. Mithilfe dieser Informationen können Client-Geräte den Core erkennen. Gehen Sie wie folgt vor:

   a. Wählen Sie auf der Gruppenkonfigurationsseite die OptionLambda-Funktionen-Registerkarte.

   b. UNDERSystem-Lambda-Funktionen, wählenIP-Detektorund wählen Sie anschließend ausBearbeitenaus.

   c. In derEinstellungen für IP-Detektor bearbeiten, wählenAutomatische Erkennung und Überschreibung von MQTT-Broker-und wählen Sie anschließend ausSaveaus.

4. Stellen Sie sicher, dass der Greengrass-Daemon, wie in beschrieben, ausgeführt wirdBereitstellen von Cloud-Konfigurationen für ein Core-Gerätaus.

5. Wählen Sie auf der Gruppenkonfigurationsseite die OptionBereitstellenaus.

Der Bereitstellungsstatus wird im Header der Seite unter dem Namen der Gruppe angezeigt. Um die Bereitstellungsdetails anzuzeigen, wählen Sie die OptionBereitstellungen-Registerkarte.

Installieren desAWS IoT Device SDKfor Python

Client-Geräte können dasAWS IoT Device SDKfür Python zum Kommunizieren mitAWS IoTundAWS IoT Greengrass-Geräte (unter Verwendung der Programmiersprache Python). Weitere Informationen, einschließlich der Voraussetzungen, finden Sie in derAWS IoT Device SDKfor PythonReadmeauf GitHubaus.

In diesem Schritt installieren Sie das SDK und rufen das`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. Herunterladen desAWS IoT Device SDKfor Pythonalszip-Datei und extrahieren Sie die Datei zu einem geeigneten Speicherort 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 ist dieser Dateipfad als *`path-to-SDK-folder`* dargestellt.

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* dargestellt.
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
```

DieAWS IoT Device SDKfür Python benötigt OpenSSL-Version 1.0.1 (oder höher), kompiliert mit der Python-Programmdatei. 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 den folgenden Befehl aus, um zu installieren:AWS IoT Device SDKfür Python:

   ```
   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 den folgenden Befehl aus, um zu installieren:AWS IoT Device SDKfür Python:

```
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. Nach demAWS IoT Device SDKfür Python installiert ist, navigieren Sie zu`samples`und öffnen Sie den`greengrass`folder.

   Für dieses Tutorial kopieren Sie die `basicDiscovery.py`-Beispielfunktion, die die Zertifikate und Schlüssel verwendet, die Sie in the section called “Erstellen Sie Client-Geräte in einerAWS IoT GreengrassGruppe” heruntergeladen haben.

3. Kopieren`basicDiscovery.py`in den Ordner, der die Datei enthält HelloWorld_Publisher und HelloWorld_Abonnentengerätezertifikate und -schlüssel.

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 AWS IoT Greengrass, stellen Sie sicher, dass die Regeln für eingehende Sicherheitsgruppen für die Instance ICMP-Verkehr für Echo-Anforderungsnachrichten zulassen. Weitere Informationen finden Sie unter Hinzufügen von Regeln zu einer Sicherheitsgruppe im Amazon EC2 EC2-Benutzerhandbuch.
> Möglicherweise müssen Sie auf Windows-Host-Computern in der App „Windows-Firewall mit erweiterter Sicherheit“ eine eingehende Regel aktivieren, die eingehende Echo-Anfragen (z. B. File and Printer Sharing (Echo Request - ICMPv4-In) (Datei- und Druckerfreigabe [Echo-Anfrage – ICMPv4-In])) zulässt oder selber eine erstellen.

2. Holen Sie sich Ihren AWS IoT Endpunkt.

   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 die *AWS_IOT_ENDPOINT*-Platzhalter in den Befehlen der 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**
   >
   > Client-IDs, die für MQTT-Verbindungen verwendet werden, müssen mit dem Ding-Namen des Client-Geräts übereinstimmen. Das `basicDiscovery.py` Skript setzt die Client-ID für MQTT-Verbindungen auf den Ding-Namen, 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.

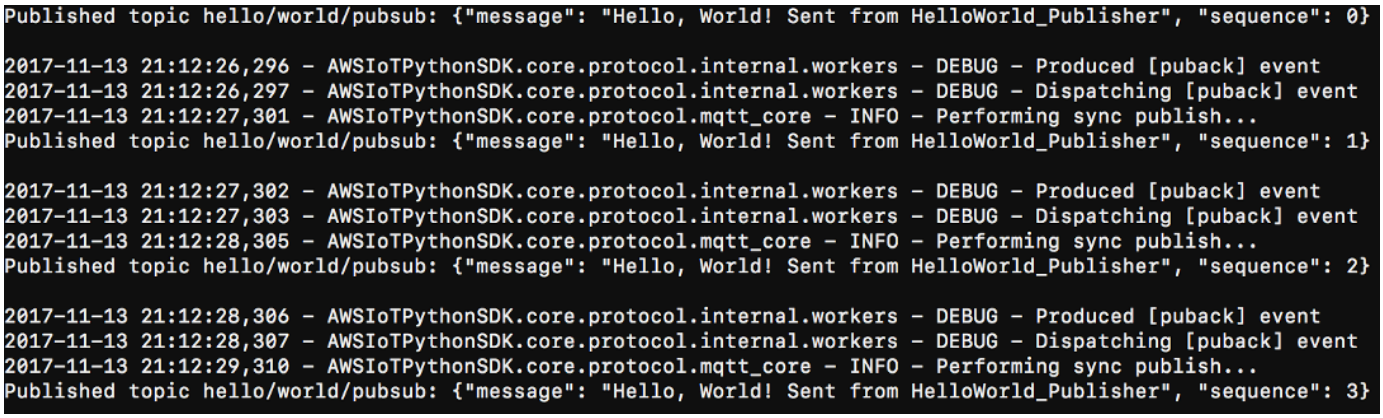
   - Ersetzen Sie *`path-to-certs-folder`* mit dem Pfad zu dem Ordner, in dem sich die Zertifikate, Schlüssel und die Datei `basicDiscovery.py` befinden.
   - Ersetzen Sie *`AWS_IOT_ENDPOINT`* mit Ihrem Endpunkt.
   - Ersetzen Sie die beiden *`CertIdPublisher-Instanzen`* durch die Zertifikat-ID im Dateinamen Ihres HelloWorld _Publisher-Clientgeräts.

   ```
   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.

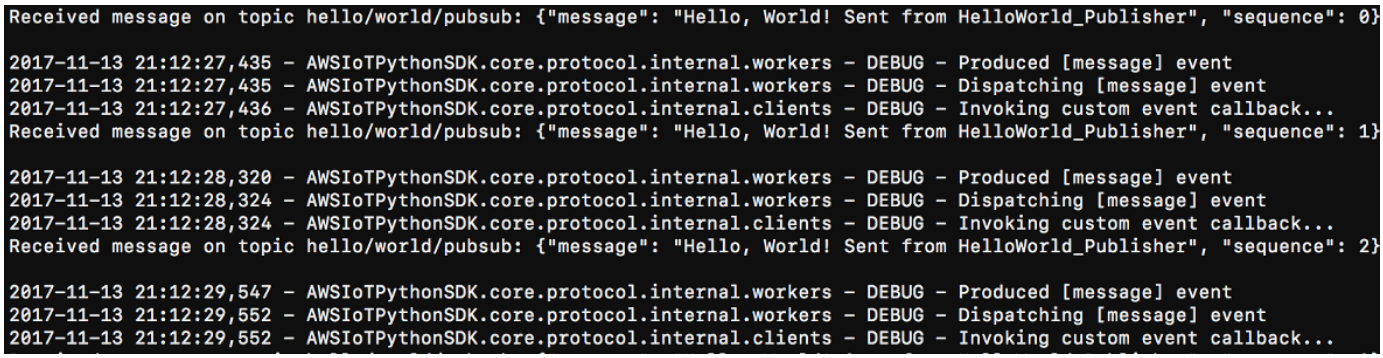


5. Führen Sie im Fenster des HelloWorld _Subscriber-Client-Geräts die folgenden Befehle aus.

   - Ersetzen Sie *`path-to-certs-folder`* mit dem Pfad zu dem Ordner, in dem sich die Zertifikate, Schlüssel und die Datei `basicDiscovery.py` befinden.
   - Ersetzen Sie *`AWS_IOT_ENDPOINT`* mit Ihrem Endpunkt.
   - Ersetzen Sie die beiden *`CertIdAbonnenteninstanzen`* durch die Zertifikat-ID im Dateinamen für Ihr HelloWorld _Subscriber-Clientgerät.

```
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}`.



Schließen Sie das HelloWorld_Publisher Fenster, um zu verhindern, dass sich Nachrichten im Fenster ansammeln. HelloWorld_Subscriber

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 mit der 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.

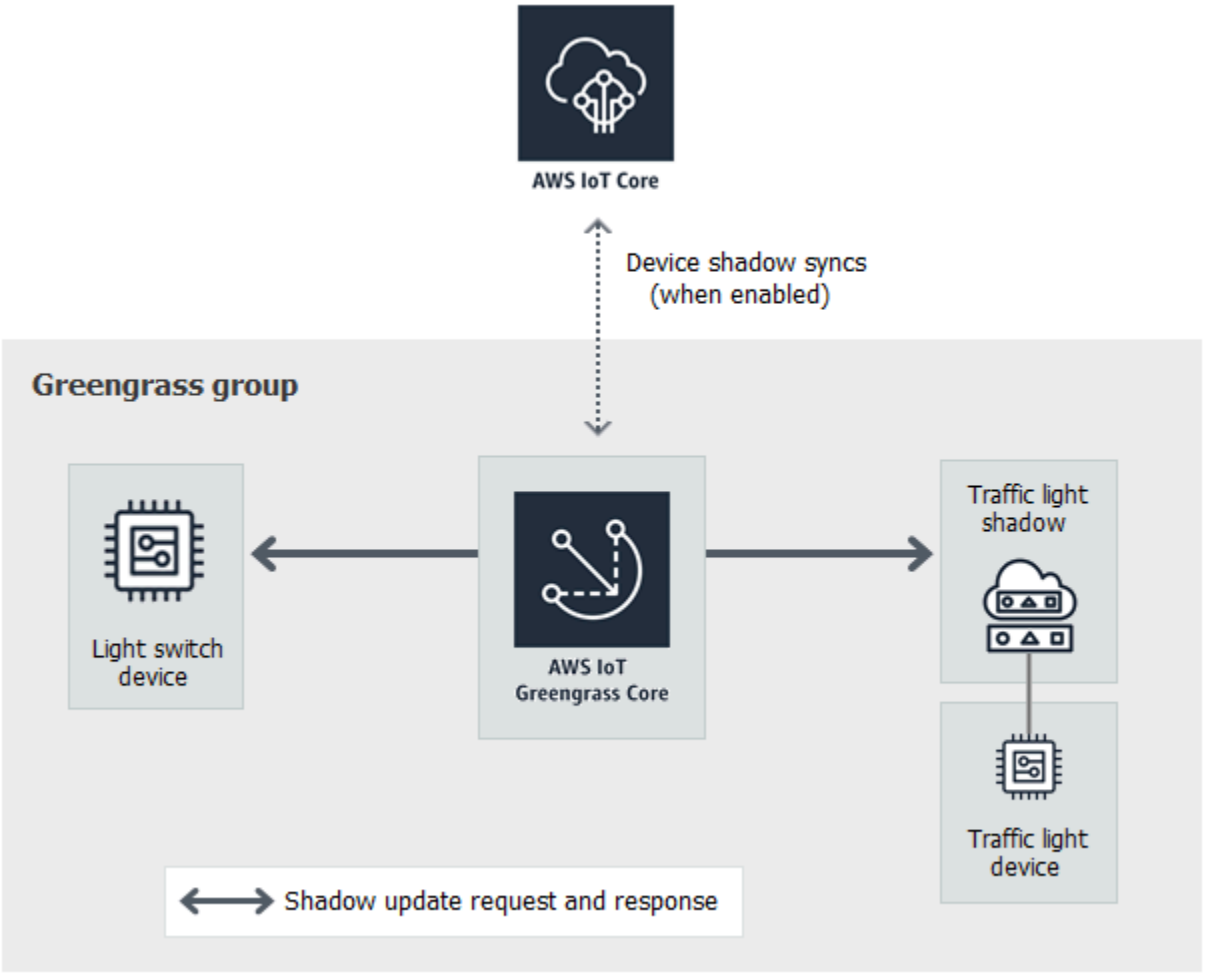
b. Wählen Sie unter MQTT-Broker-Endpunkte die Option Endpunkte verwalten aus.

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

In diesem fortgeschrittenen Modul wird gezeigt, wie Client-Geräte mit interagieren könnenAWS IoTGeräteschattenin einemAWS IoT GreengrassGruppe. 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`) kann den Status eines anderen Client-Geräts ändern (`GG_TrafficLight`) und wie diese Zustände mit dem synchronisiert werden könnenAWS IoT Greengrasscloud:



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 außerdem wissen, wie Client-Geräte mit einemAWS IoT Greengrass-Kern (Modul 4) enthalten. 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 mitAWS IoT, wenn dieAWS IoT GreengrassCore ist mit dem Internet verbunden. 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 einen eigenen Schatten. Weitere Informationen finden Sie unterDevice Shadow-Service fürAWS IoTimAWS IoTEntwicklerhandbuchaus.

1. Wählen Sie auf der Gruppenkonfigurationsseite die OptionClient-GeräteRegisterkarte.
2. AusClient-Geräteund fügen Sie zwei neue Client-Geräte inAWS IoT GreengrassGruppe. Ausführliche Informationen zu den Schritten dieses Prozesses finden Sie unter the section called “Erstellen Sie Client-Geräte in einerAWS IoT GreengrassGruppe”.

   - Geben Sie die Client-Geräte**`GG_Switch`**und**`GG_TrafficLight`**aus.
   - Generieren und laden Sie die Sicherheits-Ressourcen für beide Client-Geräte herunter.
   - Notieren Sie sich die Zertifikat-ID in den Dateinamen der Sicherheits-Ressourcen für die Client-Geräte. Sie werden diese Werte in einem späteren Schritt verwenden.
3. Erstellen Sie auf Ihrem Computer einen Ordner für die Sicherheitsanmeldedaten 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.AWS Cloudaus. Falls nicht, wählen Sie das Client-Gerät ausSchattensynchronisierungund danach auf.Shadow Sync mit Cloud deaktivierenaus.
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 OptionAbonnementsund wählen Sie dannAddaus.
   b. Für-Quelltyp, wählenClient-Gerätund danach auf.GG_Switchaus.
   c. FürZieltyp, wählen-Serviceund danach auf.Local Shadow Serviceaus.
   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 unterShadow MQTT-ThemenimAWS IoTEntwicklerhandbuchaus.

Quelle	Ziel	Topic	Hinweise
GG_Switch	Local Shadow Service	$aws/things/things /things/GG_Traffic Light/shadow/update	Der GG_Switch sendet eine Aktualisierungsanfrage an das Aktualisierungsthema.
Local Shadow Service	GG_Switch	$aws/things/things /things/GG_Traffic Light/thadow/update/ accepted	Die GG_Switch muss wissen, ob die Aktualisierungsanforderung akzeptiert wurde.
Local Shadow Service	GG_Switch	$aws/things/things /things/GG_Traffic Light/Shadow/update/GG	Die GG_Switch muss wissen, ob die Aktualisierungsanforderung abgelehnt wurde.
GG_TrafficLight	Local Shadow Service	$aws/things/things /things/GG_Traffic Light/shadow/update	Die GG_TrafficLight sendet eine Aktualisierung seines Status an das Aktualisierungsthema.
Local Shadow Service	GG_TrafficLight	$aws/things/things /things/GG_Traffic Light/things/update/ delta	Der lokale Schattenservice sendet eine empfangene Aktualisierung an GG_TrafficLight durch das Delta-Thema.

Quelle	Ziel	Topic	Hinweise
Local Shadow Service	GG_TrafficLight	$aws/things/things/things/GG_TrafficLight/thadow/update/accepted	Die GG_TrafficLight muss wissen, ob die Statusaktualisierung akzeptiert wurde.
Local Shadow Service	GG_TrafficLight	$aws/things/things/things/GG_TrafficLight/Shadow/update/GG	Die GG_TrafficLight muss wissen, ob die Statusaktualisierung abgelehnt wurde.

Die neuen Abonnements werden aufAbonnementsRegisterkarte.

> **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. Mithilfe dieser Informationen können Client-Geräte den Core erkennen. Gehen Sie wie folgt vor:

   a. Wählen Sie auf der Gruppenkonfigurationsseite die OptionLambda-FunktionenRegisterkarte.

   b. UnderSystem-Lambda-Funktionen, wählenIPdetektorund danach auf.Bearbeitenaus.

   c. In derEinstellungen für IP-Detektor bearbeiten, wählenAutomatische Erkennung und Überschreibung von MQTT-Broker-und danach auf.Saveaus.

7. Stellen Sie sicher, dass der Greengrass-Daemon, wie inBereitstellen von Cloud-Konfigurationen für ein Core-Gerätaus.

8. Wählen Sie auf der Gruppenkonfigurationsseite die OptionBereitstellenaus.

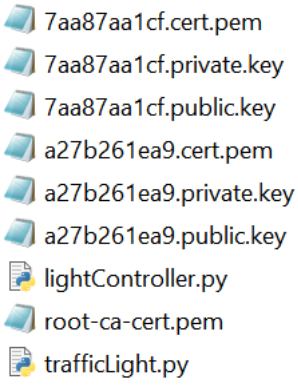
Download der benötigten Dateien

1. Installieren Sie das, falls noch nicht erfolgt, installieren Sie das herunter:AWS IoT Device SDKfor Python. Anweisungen hierzu finden Sie in Schritt 1 unter the section called “Installieren desAWS IoT Device SDKfor Python”.

Dieses SDK wird von Client-Geräten für die Kommunikation mit verwendetAWS IoTund mitAWS IoT Greengrass-Core-Geräte.

2. Aus TrafficLightBeispiele-Ordner auf GitHubladen Sie das herunter:`lightController.py`und`trafficLight.py`-Dateien auf Ihrem Computer herunter. Speichern Sie sie in dem Ordner, der GG_Switch und GG_ GG_Switch und GG_TrafficLight Zertifikate und Schlüssel für Clientgeräte.

   Die`lightController.py`-Skript entspricht dem GG_Switch und das Skript`trafficLight.py`-Skript entspricht GG_TrafficLight Client-Gerät.



> **Note**
>
> Die Python-Beispieldateien werden in der DateiAWS IoT GreengrassCore-SDK for Python-Repository aus praktischen Gründen, sie verwenden das jedoch nichtAWS IoT GreengrassCore-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 AWS IoT Greengrass, stellen Sie sicher, dass die Regeln für eingehende Sicherheitsgruppen für die Instance ICMP-Verkehr für Echo-Anforderungsnachrichten zulassen. Weitere Informationen finden Sie unter Hinzufügen von Regeln zu einer Sicherheitsgruppe im Amazon EC2 EC2-Benutzerhandbuch.
> Möglicherweise müssen Sie auf Windows-Host-Computern in der App „Windows-Firewall mit erweiterter Sicherheit“ eine eingehende Regel aktivieren, die eingehende Echo-Anfragen (z. B. File and Printer Sharing (Echo Request - ICMPv4-In) (Datei- und Druckerfreigabe [Echo-Anfrage – ICMPv4-In])) zulässt oder selber eine erstellen.

2. Holen Sie sich Ihren AWS IoT Endpunkt.

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 die *AWS_IOT_ENDPOINT*-Platzhalter in den Befehlen der 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 Befehlszeile). 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.

   - Ersetzen Sie *path-to-certs-folder* mit dem Pfad zu dem Ordner, in dem sich die Zertifikate, Schlüssel und die Python-Dateien befinden.
   - Ersetzen Sie *AWS_IOT_ENDPOINT* mit Ihrem Endpunkt.
   - Ersetzen Sie die beiden *CertIdSwitch-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.

   - Ersetzen Sie *path-to-certs-folder* mit dem Pfad zu dem Ordner, in dem sich die Zertifikate, Schlüssel und die Python-Dateien befinden.
   - Ersetzen Sie *AWS_IOT_ENDPOINT* mit Ihrem Endpunkt.
   - Ersetzen Sie die beiden *CertIdLight-Instances* 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 ++++++++++
{u'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 dieser Shadow nicht mehr aktualisiert werden. 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 Geräteschatten zum SynchronisierenAWS IoTaus. 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. In derAWS IoT-Konsole aus.AWS IoT Greengrassaus. Wählen Sie anschließend die Option ausClient-GeräteRegisterkarte.
2. Wählen Sie GG_TrafficLight gerät, wählen SieSynchronisierung Shadowund anschließend die Option ausShadow Sync mit Cloud aktivierenaus.

   Sie sollten eine Benachrichtigung erhalten, dass der Gerätesadow Sync-Status aktualisiert wurde.
3. Wählen Sie auf der Gruppenkonfigurationsseite die Option ausBereitstellenaus.
4. Führen Sie in den beiden Befehlszeilenfenster die Befehle aus dem vorherigen Test für die Option aus dem vorherigen Test ausGG_SwitchundGG_TrafficLightClient-Geräte.
5. Überprüfen Sie nun den Schattenzustand in derAWS IoTconsole. Wählen Sie IhreAWS IoT GreengrassGruppe, wählen SieClient-Geräte-Registerkarte, wählen SieGG_TrafficLight, wählen SieDevice Shadowsund anschließend die Option ausClassic Shadowaus.

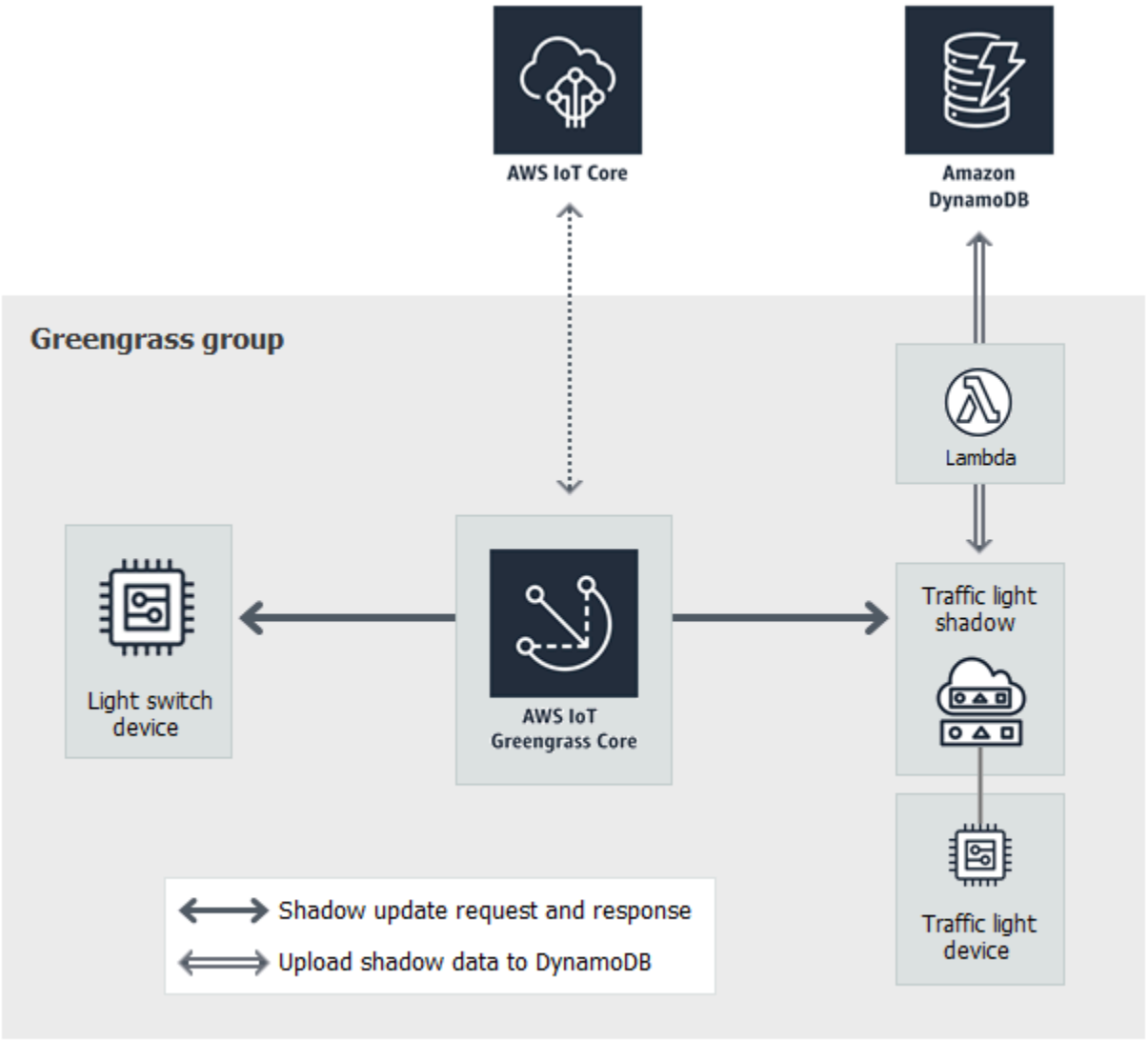
   Weil du die Synchronisierung von GG_ aktiviert hastTrafficLight Shadow aufAWS IoTsollte der Schattenzustand in der Cloud automatisch aktualisiert werden, sobald GG_Switch eine Aktualisierung sendet. Diese Funktionalität kann verwendet werden, um den Status eines Client-Geräts anAWS IoTaus.

> **Note**
>
> Falls erforderlich, können Sie Probleme mithilfe desAWS IoT GreengrassKernprotokolle, 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.

Behalten Sie die Client-Geräte und Abonnements eingerichtet. Sie werden sie im nächsten Modul wieder verwenden. Zudem werden Sie die gleichen Befehle ausführen.

Modul 6: Zugreifen auf andereAWSDienstleistungen

In diesem erweiterten Modul erfahren Sie mehr darüberAWS IoT GreengrassKerne können mit anderen interagierenAWSDienste in der Cloud. Es baut auf dem Ampelbeispiel vonModul 5und fügt eine Lambda-Funktion hinzu, die Schattenzustände verarbeitet, und lädt eine Zusammenfassung in eine Amazon DynamoDB DynamoDB-Tabelle.



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 5abschließen. 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 der Operationen klein sind und in das kostenlose Kontingent von Amazon Web Services fallen, können bei der Ausführung einiger dieser Schritte in diesem Modul Gebühren für Ihr Konto

anfallen. Weitere Informationen zu Preisen erhalten Sie unterDokumentation zu DynamoDB - Preisenaus.

Themen

- Konfigurieren der Gruppenrolle
- So erstellen Sie die Lambda-Funktion erstellen und konfigurieren Sie noch zu konfigurieren
- Konfigurieren von Abonnements
- Testen der Kommunikation

Konfigurieren der Gruppenrolle

Die Gruppenrolle ist eineIAM-Rolledie Sie erstellen und Ihrer Greengrass-Gruppe anfügen. Diese Rolle enthält die Berechtigungen, die Lambda-Funktionen (und andereAWS IoT GreengrassFunktionen) zum Zugreifen verwendenAWS-Services. 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.

Dann wird in derAWS IoT-Konsole fügen Sie die Rolle 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 Lernprogramm erstellen Sie eine Berechtigungsrichtlinie, die das Beschreiben, Erstellen und Aktualisieren von Aktionen für eine Amazon DynamoDB-Tabelle gestattet. Anschließend fügen Sie die Richtlinie an eine neue Rolle an und ordnen die Rolle Ihrer Greengrass-Gruppe zu.

Zuerst erstellen Sie eine vom Kunden verwaltete Richtlinie, die die von der Lambda-Funktion in diesem Modul erforderlichen Berechtigungen gewährt.

1. Wählen Sie im Navigationsbereich der IAM-KonsoleRichtlinienund wählen Sie dannRichtlinie erstellenaus.
2. Ersetzen Sie auf der Registerkarte JSON den Platzhalterinhalt durch die folgende Richtlinie. Die Lambda-Funktion in diesem Modul verwendet diese Berechtigungen zum Erstellen und Aktualisieren einer DynamoDB-Tabelle mit dem Namen`CarStats`aus.

   ```
   {
       "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. Tagsund wählen Sie dannWeiter: Prüfen. 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 Roles (Rollen) und dann Create role (Rolle erstellen).
6. UnderTrusted entity type (, wählenAWSBedienungaus.

7. UnderAnwendungsfall,Anwendungsfälle für andereAWSDienstleistungenwählenGreengrasswählen SieGreengrassund wählen Sie dannWeiteraus.
8. UnderBerechtigungsrichtlinien, wählen Sie das neue**greengrass_CarStats_Table**und wählen Sie dannWeiteraus.
9. Geben Sie für Role name (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 Create role (Rolle erstellen) aus.

    Fügen Sie die Rolle nun Ihrer Greengrass-Gruppe hinzu.
12. In derAWS IoTNavigationsbereich der -Konsole, underVerwalten, erweiternGreengrass-Geräteund wählen Sie dannGruppen (V1)aus.
13. UnderGreengrass Gruppenwählen Sie Ihre Gruppe aus.
14. Klicken Sie aufEinstellungenund wählen Sie dannRolle zuordnenaus.
15. Klicken Sie aufGreengrass_Group_Roleaus der Liste der Rollen und wählen Sie dannRolle zuordnenaus.

So erstellen Sie die Lambda-Funktion erstellen und konfigurieren Sie noch zu konfigurieren

In diesem Schritt erstellen Sie eine Lambda-Funktion, die Anzahl der Fahrzeuge, die Ampel passieren, die Ampel passieren, die Ampel passieren ist. Jedes Mal, wenn der`GG_TrafficLight` Schattenzustand wechselt`G`, simuliert die Lambda-Funktion das Vorbeifahren einer zufälligen Anzahl von Autos (von 1 bis 20). Bei jeder dritten`G` leichten Änderung 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`carAggregator.py` Datei ab dem GitHub Ordner mit den TrafficLight Beispielen in den`car_aggregator` Ordner herunter. Dies ist Ihr Lambda-Funktionscodes ist.

> **Note**
>
> Diese Python-Beispieldatei wird der Einfachheit halber imAWS IoT Greengrass Core SDK-Repository gespeichert, verwendet jedoch nicht dasAWS IoT Greengrass Core SDK.

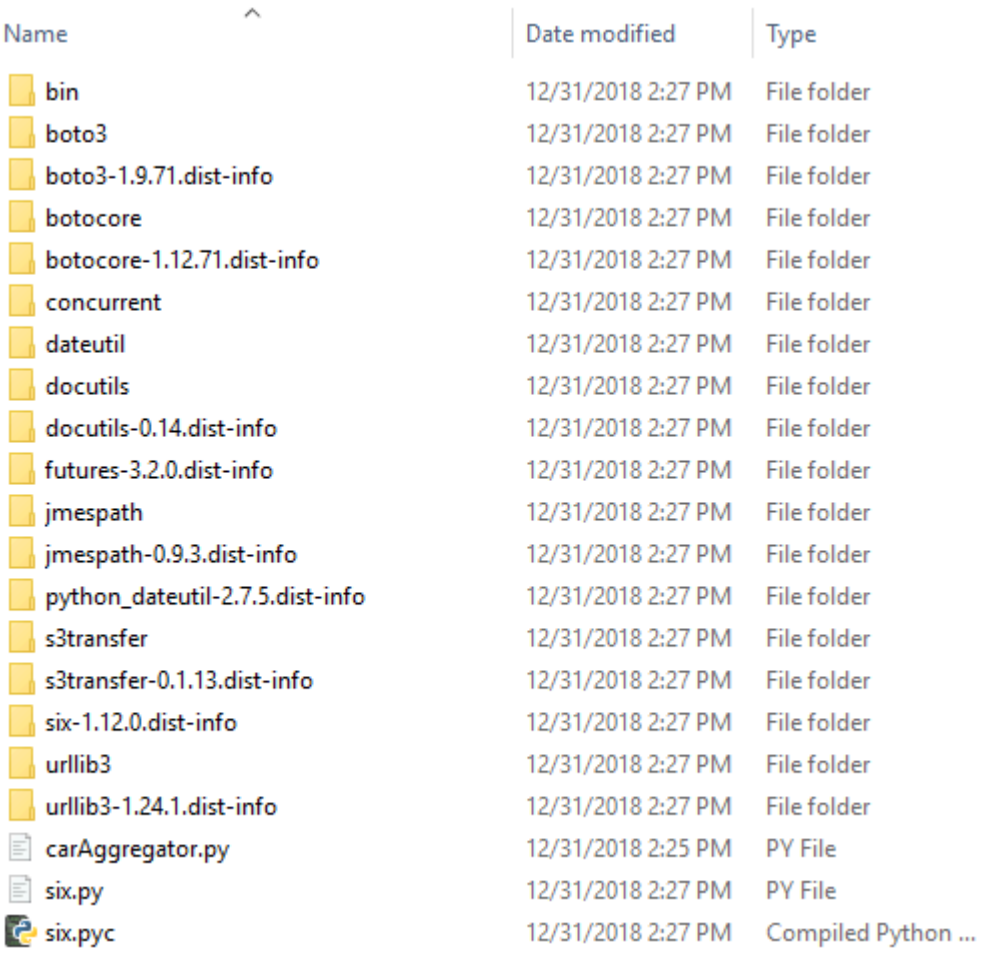
3. Wenn Sie nicht in der Region USA Ost (Nord-Virginia) arbeiten, öffnen Sie die folgende Zeile`carAggregator.py` und wechseln`region_name` Sie zu der RegionAWS-Region, die derzeit in derAWS IoT Konsole ausgewählt ist. Eine Liste der unterstütztenAWS-Region s finden Sie AWS IoT Greengrassin der Allgemeine 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 for Python (Boto3)Paket und seine Abhängigkeiten in dem`car_aggregator` Ordner zu installieren. Die Funktionen von Greengrass Lambda verwenden dasAWS SDK, um auf andereAWS 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:

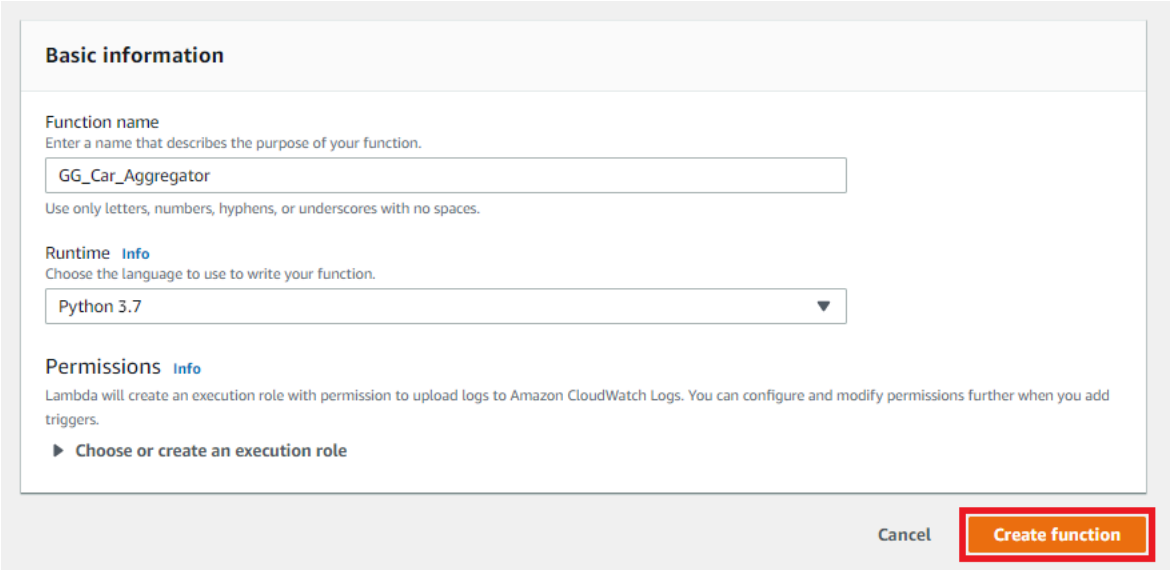


Name	Date modified	Type
bin	12/31/2018 2:27 PM	File folder
boto3	12/31/2018 2:27 PM	File folder
boto3-1.9.71.dist-info	12/31/2018 2:27 PM	File folder
botocore	12/31/2018 2:27 PM	File folder
botocore-1.12.71.dist-info	12/31/2018 2:27 PM	File folder
concurrent	12/31/2018 2:27 PM	File folder
dateutil	12/31/2018 2:27 PM	File folder
docutils	12/31/2018 2:27 PM	File folder
docutils-0.14.dist-info	12/31/2018 2:27 PM	File folder
futures-3.2.0.dist-info	12/31/2018 2:27 PM	File folder
jmespath	12/31/2018 2:27 PM	File folder
jmespath-0.9.3.dist-info	12/31/2018 2:27 PM	File folder
python_dateutil-2.7.5.dist-info	12/31/2018 2:27 PM	File folder
s3transfer	12/31/2018 2:27 PM	File folder
s3transfer-0.1.13.dist-info	12/31/2018 2:27 PM	File folder
six-1.12.0.dist-info	12/31/2018 2:27 PM	File folder
urllib3	12/31/2018 2:27 PM	File folder
urllib3-1.24.1.dist-info	12/31/2018 2:27 PM	File folder
carAggregator.py	12/31/2018 2:25 PM	PY File
six.py	12/31/2018 2:27 PM	PY File
six.pyc	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 Bereitstellungspaket für die Lambda-Funktion verwenden ist.
6. Erstellen Sie in der Lambda-Konsole eine Funktion mit dem Namen**`GG_Car_Aggregator`** und legen Sie die verbleibenden Felder wie folgt fest:

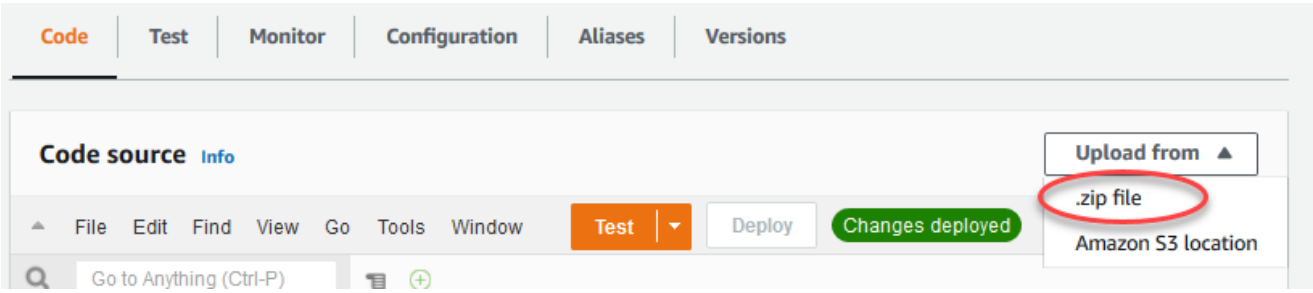
   - Wählen Sie für Runtime (Laufzeit) die Option Python 3.7 aus.
   - Behalten Sie für Berechtigungen die Standardeinstellung bei. Dadurch wird eine Ausführungsrolle erstellt, die grundlegende Lambda-Berechtigungen gewährt. Diese Rolle wird nicht verwendet vonAWS IoT Greengrass.

   Wählen Sie Create function (Funktion erstellen).



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 aus der Dropdownliste die ZIP-Datei aus.



   b. Wählen Sie Upload und dann Ihr`car_aggregator.zip` Deployment-Paket 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 namens**`GG_CarAggregator`**. step-by-step Eine Anleitung finden Sie in den Schritten zum Veröffentlichen der Lambda-Funktion und zum Erstellen eines Alias in Modul 3 (Teil 1).

9. Fügen Sie in derAWS IoT Konsole die Lambda-Funktion, die Sie gerade erstellt haben, zu IhrerAWS 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 Version der Lambda-Funktion den Alias für die Version, die Sie veröffentlicht haben.

d. Geben Sie in Memory Limit (Speicherlimit) **`64 MB`** ein.

e. Wählen Sie für Angeheftet 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 GG_ ermöglichtTrafficLight shadow, um aktualisierte Zustandsinformationen an die -Funktion GG_Car_Aggregator Lambda 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 Option aus.Abonnementsund wählen Sie dann ausAddaus.
2. Auf derErstellen eines Abonnements-Seite, machen Sie das Folgende:

   a. FürQuelldatentyp, wählen-Serviceund anschließend ausLocal Shadow Serviceaus.

   b. FürZieltyp, wählenLambda-Funktionund anschließend ausGG_Car_Aggregatoraus.

   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. Stellen Sie sicher, dass der Greengrass-Daemon, wie in beschrieben, ausgeführt wirdBereitstellen von Cloud-Konfigurationen für ein Core-Gerätaus.
4. Wählen Sie auf der Gruppenkonfigurationsseite die Option ausBereitstellenaus.

Testen der Kommunikation

1. Öffnen Sie zwei Befehlszeilenfenster auf Ihrem Computer. So wie inModul 5, ein Fenster für das GG_Switch-Client-Gerät und das andere für GG_TrafficLight -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 GG_ ausTrafficLight Client-Gerät:

   ```
   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. Bei jedem dritten grünen Licht (alle 3 Minuten) wird der Funktionshandler der Lambda-Funktion ausgelöst und ein neuer DynamoDB-Datensatz erstellt. Nach`lightController.py`und`trafficLight.py`drei Minuten lang ausgeführt wurden, wechseln Sie zurAWS Management Consoleund öffnen Sie die DynamoDB-Konsole.
3. Klicken Sie aufUSA Ost (Nord-Virginia)imAWS-RegionMenü. Dies ist die Region, in der die `GG_Car_Aggregator`-Funktion die Tabelle erstellt.
4. Wählen Sie im Navigationsbereich und dann aus.Tabellenund dann ausCarStatsTabelle.
5. Klicken Sie aufArtikel (Ansicht)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`. Der Zugriff auf AWS IoT Greengrass-Protokolle erfordert Root-Berechtigungen.

   ```
   sudo su
   cd /greengrass/ggc/var/log
   ```

   b. Prüfen Sie `runtime.log` auf Fehler.

   ```
   cat system/runtime.log | grep 'ERROR'
   ```

   c. Prüfen Sie das Protokoll, das von der Lambda-Funktion generiert wurde.

   ```
   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 im`ggc-host`-Datei stimmt mit dem IP-Adressendpunkt für Ihren Core überein.

   Weitere Informationen finden Sie unter Fehlerbehebung.

Dies ist das Ende des grundlegenden Tutorials. Du solltest jetzt das verstehenAWS IoT GreengrassProgrammiermodell und seine grundlegenden Konzepte, einschließlichAWS IoT Greengrass-Kerne, Gruppen, Abonnements oder Client-Geräte und der Bereitstellungsprozess Lambda Edge ausgeführte -Funktionen verfügen.

Sie können die DynamoDB-Tabelle und die Greengrass Lambda-Funktionen und -Abonnements löschen. Um die Kommunikation zwischen denAWS IoT GreengrassKerngerät und dasAWS IoT-Cloud ein Terminal auf dem Core-Gerät und führen Sie einen der folgenden Befehle aus:

- Schließen Sie dasAWS IoT Greengrass-Core-Gerät:

```
sudo halt
```

- So beenden Sie den AWS IoT Greengrass-Daemon:

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

Modul 7: Simulation der Hardware-Sicherheitsintegration

Diese Funktion ist verfügbar fürAWS IoT GreengrassCore v1.7 und höher.

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 Hardware-Sicherheit auf einemAWS IoT GreengrassKern, einschließlich allgemeiner Anforderungen, siehethe section called “Integration von Hardware-Sicherheit”aus.

> **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 beim Onboarding einesunterstütztes Hardware-Sicherheitsmodul, kontaktieren Sie IhreAWSVertreter von Enterprise-Support

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 mitAWSaus. 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.

- In einem Terminal auf IhremAWS IoT GreengrassCore-Gerät, führen Sie 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 bei der Verwendung dieses Befehls auf Ihrem System Probleme auftreten, siehe 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 `libsofthsm2.so` befindet sich im Verzeichnis `softhsm`.

   b. Kopieren Sie den vollständigen Pfad in die Datei (z. B. `/usr/lib/x86_64-linux-gnu/softhsm/libsofthsm2.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 zu verwenden over-the-air (OTA) Updates mit HardwaresicherheitPKCS11Objekt muss auch das`OpenSSLEngine`eigentum. Weitere Informationen finden Sie unter the section called “OTAUpdates konfigurieren”.

6. Bearbeiten Sie das Objekt `crypto`:

   a. Konfigurieren Sie das Objekt `PKCS11`.

      - Geben Sie für `P11Provider` den vollständigen Pfad zur Datei `libsofthsm2.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/softhsm/libsofthsm2.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. OASIS-Ausschuss

Anmerkung 02. 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-Agent, 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 Cores 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 OTA-Updates der AWS IoT Greengrass-Software.

- Der Greengrass-Core muss über mindestens 400 MB Festplattenspeicher im lokalen Speicher verfügen. Der OTA-Update-Agent benötigt etwa das Dreifache der Laufzeitnutzungsanforderung 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 zur Authentifizierung mit AWS IoT Core und AWS IoT Greengrass versorgt sein. 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 Cores unterstützt, die MQTT-Datenverkehr so konfigurieren, dass Port 443 anstelle des Standardports 8883 verwendet wird. 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 in 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 Lese-/Schreibberechtigungen 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 den Amazon S3-URLs AWS IoT Greengrass Softwareupdate-Artefakten vorzugeben. Diese Unterzeichnerrolle ermöglicht Ihnen AWS IoT Core den Zugriff auf Software-Update-Artefakte, die in Amazon S3 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-URL-Signerrolle 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 AWS-Region Ziel-s zulässt. 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 der von AWS verwalteten Richtlinie

  Die von GreenGrassOTA UpdateArtifactAccess verwaltete Richtlinie wird bereitgestellt von. AWS IoT Greengrass Verwenden Sie diese Richtlinie, wenn Sie den Zugriff in allen Amazon Web Services-Regionen zulassen möchtenAWS IoT Greengrass, die von aktuellen und zukünftigen unterstützt werden.

- Erstellen Sie eine benutzerdefinierte Richtlinie

  Sie sollten eine benutzerdefinierte Richtlinie erstellen, wenn Sie die Amazon Web Services-Regionen explizit angeben möchten, in denen Ihre Cores bereitgestellt werden. Die folgende Beispielrichtlinie erlaubt den Zugriff auf AWS IoT Greengrass-Software-Updates in sechs Regionen:

```
{
    "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:

```
{
     "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:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "greengrass:CreateSoftwareUpdateJob"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
```

```
                "iot:CreateJob"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "arn-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 Cores gehen alle Verbindungen mit dem 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 Arbeiten werden verworfen.

Die folgenden Statusinformationen bleiben während einer OTA-Aktualisierung erhalten:

- Core-Konfiguration
- Konfiguration der Greengrass-Gruppe
- Lokale Schatten

- Greengrass-Protokolle
- Protokolle des OTA-Update-Agenten

Greengrass OTA-Update-Agent

Der Greengrass OTA-Update-Agent ist die Softwarekomponente auf dem Gerät, die Aktualisierungsaufträge verarbeitet, die in der Cloud erstellt und bereitgestellt werden. Der OTA-Update-Agent wird im gleichen 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* steht für den Pfad, unter dem die AWS IoT Greengrass Core-Software auf Ihrem Gerät installiert ist. Normalerweise ist dies das Verzeichnis `/greengrass`.

Sie können den OTA-Aktualisierungsagenten starten, indem Sie die Binärdatei manuell ausführen oder indem Sie sie als Teil eines Init-Skripts, z. B. einer systemd-Dienstdatei, 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 sequentiell aus. Der OTA-Update-Agent ignoriert alle anderen AWS IoT Jobtypen.

Der folgende Auszug zeigt ein Beispiel für eine systemd-Dienstdatei zum Starten, Stoppen und Neustarten des OTA-Aktualisierungsagenten:

```
[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-Agenten 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 stoppen, bevor das Gerät heruntergefahren wird.

Um den OTA-Aktualisierungsagenten anzuweisen, diese Skripts auszuführen, müssen Sie das `"managedRespawn" : true` Flag in die Datei config.json aufnehmen. 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_pre_update.sh` `ota_post_update.sh` AND-Skripts den OTA-Dienst nicht beenden oder starten.

Der OTA-Update-Agent führt die Skripts von der 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 Greengrassüberprüft den Inhalt dieser Skripte nicht. Als bewährte Methode sollten Sie überprüfen, ob Ihre Skripts ordnungsgemäß funktionieren, und bei Fehlern die entsprechenden Exit-Codes ausgeben.

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-Agenten:

- 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

Führen Sie die folgenden Schritte aus, um ein OTA-Update der AWS IoT Greengrass-Software auf einem oder mehreren Cores 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 Cores:
>
> - Die config.json-Datei legt `"managedRespawn" : true` fest.
> - Das Verzeichnis/*greengrass-root* /usr/scripts enthält die folgenden 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 Hauptgeräteterminal den OTA-Aktualisierungsagenten.

```
cd /greengrass-root/ota/ota_agent
sudo ./ggc-ota
```

> **Note**
>
> *greengrass-root* steht für den Pfad, unter 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-Agenten auf einem Core, da dies zu Konflikten führen kann.

3. Verwenden Sie die AWS IoT Greengrass API, um einen Software-Update-Job zu erstellen.

   a. Rufen Sie die CreateSoftwareUpdateJob-API auf. In diesem Beispielverfahren verwenden wir AWS CLI-Befehle.

Mit dem folgenden Befehl wird eine Aufgabe erstellt, die die AWS IoT Greengrass-Core-Software auf einem Core 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 das 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 DescribeJobdie AWS IoT Core API auf, um den Jobstatus zu sehen. Ersetzen Sie den Beispielwert 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 Aufruf von `CreateSoftwareUpdateJob` können Sie andere AWS IoT-Aufgabenbefehle verwenden, um das Softwareupdate zu verfolgen. Weitere Informationen finden Sie unter Jobs im AWS IoTDeveloper Guide.

Das folgende Beispiel zeigt, wie AWS CLI zum Erstellen einer Aufgabe verwendet wird, 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 ARNs einzelner Cores und ARNs von Objektgruppen enthalten, deren Mitglieder Cores sind. Weitere Informationen zu Dinggruppen finden Sie unter Statische Dinggruppen im AWS IoTEntwicklerhandbuch.

`--update-targets-operating-system`

Das Betriebssystem des Core-Geräts.

Gültige Werte: `ubuntu`, `amazon_linux`, `raspbian` oder `openwrt`

`--software-to-update`

Gibt an, ob die Core-Software oder die OTA-Update-Agent-Software aktualisiert werden soll.

Gültige Werte: `core` oder `ota_agent`.

`--s3-url-signer-role`

Der ARN der IAM-Rolle, der zur Vorgabe der Amazon S3-URL verwendet wird, die auf die AWS IoT Greengrass Softwareupdate-Artefakte verweist. Die der Rolle zugewiesene Berechtigungsrichtlinie muss die `s3:GetObject` Aktion für die Buckets in den AWS-Region Ziel-s zulassen. 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-Aktualisierungsagenten 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

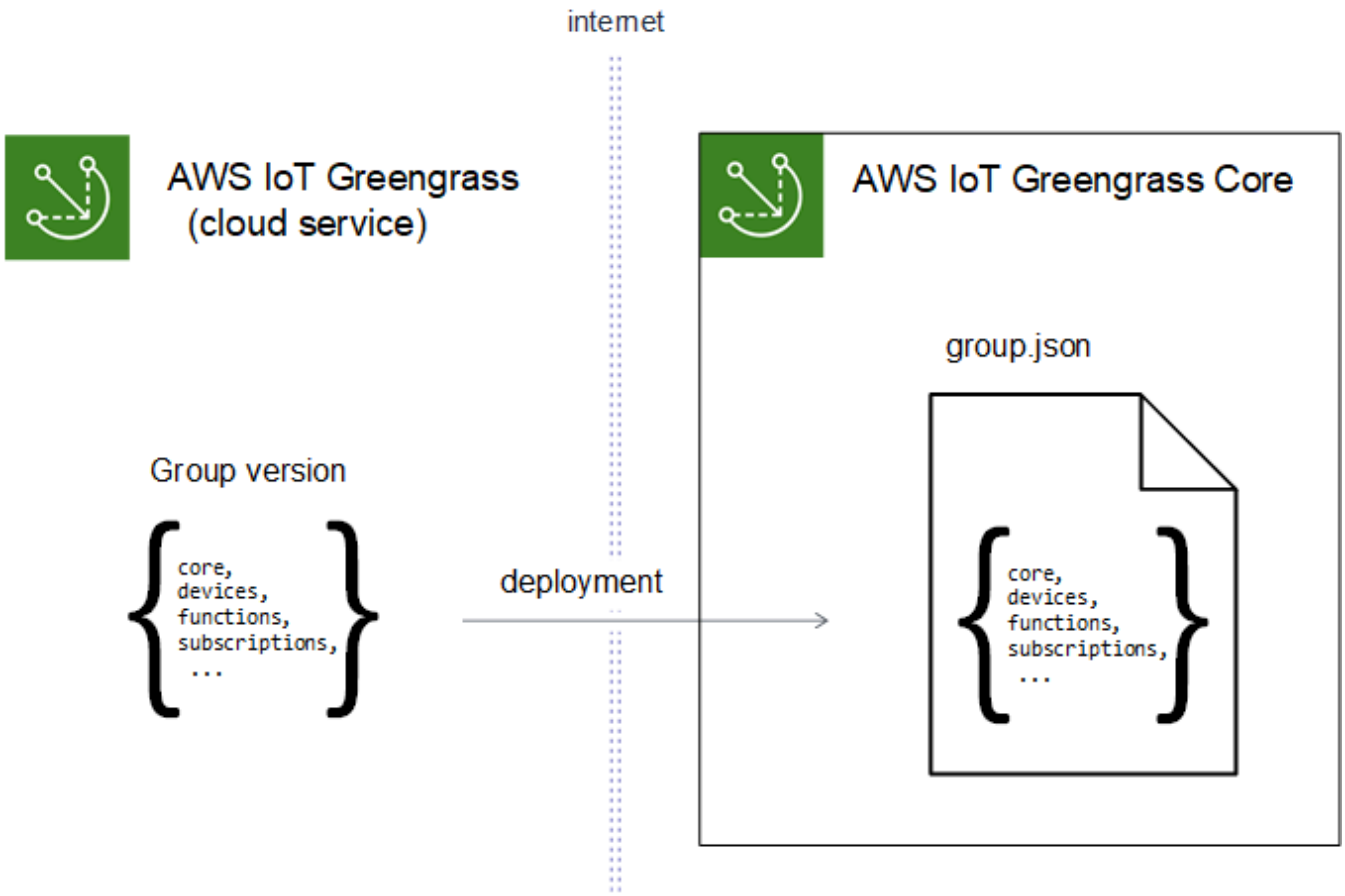
Bereitstellen von AWS IoT Greengrass-Gruppen für einen AWS IoT Greengrass Core

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 der interagierenAWS 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. Dabei handelt es sich um ein AWS IoT-Gerät, das die AWS IoT Greengrass-Core-Software ausführt. Der Core fungiert als Edge-Gateway und stellt AWS IoT Core-Funktionen in der Edge-Umgebung bereit. 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 Geräte-SDK oder die AWS IoT Greengrass Discovery-API verwenden, um Verbindungsinformationen für den Kern abzurufen. Nur Client-Geräte, die Mitglieder der Gruppe sind, können eine Verbindung zum Kern herstellen.
- Lambda-Funktionen . Benutzerdefinierte Serverless-Anwendungen, die Code auf dem -Kern ausführen. Lambda-Funktionen werden in einer Greengrass-Gruppe erstellt AWS Lambda und referenziert. Weitere Informationen finden Sie unter Lokale Lambda-Funktionen ausführen.
- Konnektoren Vordefinierte Serverless-Anwendungen, die Code auf dem -Kern ausführen. Konnektoren können eine integrierte Integration mit lokaler Infrastruktur, GeräteprotokollenAWS, und anderen Cloud-Services bieten. 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 , Machine-Learning-Modelle und Secrets , die für die Zugriffskontrolle durch Greengrass-Lambda-Funktionen und -Konnektoren verwendet werden.
- Protokolliert . Protokollierungskonfigurationen 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 in der AWS Cloud und stellen sie dann auf einem Kern 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.

Bereitstellen von Gruppen über die AWS IoT Konsole

Sie können eine Gruppe bereitstellen und ihre Bereitstellungen auf der Konfigurationsseite der Gruppe in der AWS IoT Konsole 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 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 Gruppenkonfiguration die Registerkarte Bereitstellungen aus.
2. Um weitere Informationen zu einer Bereitstellung, einschließlich Fehlermeldungen, anzuzeigen, wählen Sie Bereitstellungen von der AWS IoTKonsole unter Greengrass-Geräte 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 IoTKonsole 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 aus.

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 IoTKonsole 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 aus.

Bereitstellen von Gruppen mit der AWS IoT Greengrass-API

Die AWS IoT Greengrass-API bietet die folgenden Aktionen zum Bereitstellen von AWS IoT Greengrass-Gruppen und Verwalten von Gruppenbereitstellungen. Sie können diese Aktionen über die AWS CLI, dieAWS 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 Ereignisse für den Empfang von Bereitstellungsbenachrichtigungen konfigurieren. 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 zurückset zen, 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 ListGroups Aktion verwenden, um die ID der Zielgruppe aus Ihrer Gruppenliste zu finden. Verwenden Sie beispielsweise in der AWS CLI den `list-groups`-Befehl.

```
aws greengrass list-groups
```

Sie können auch die `query`-Option zum Filtern der Ergebnisse einschließen. Beispielsweise:

- 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 andere Versions-IDs für eine Gruppe abzurufen, verwenden Sie die Gruppen-ID mit ListGroupVersions.

> **Note**
>
> Sie finden diese Werte auch in der -AWS IoTKonsole. Die Gruppen-ID wird auf der Seite Einstellungen der Gruppe angezeigt. Gruppenversions-IDs werden 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 keinen angebenAWS-Region, AWS CLI verwenden Befehle die Standardregion aus Ihrem Profil. Um Gruppen in einer anderen Region zurückzugeben, schließen Sie die Option *`Region`* ein. Beispielsweise:

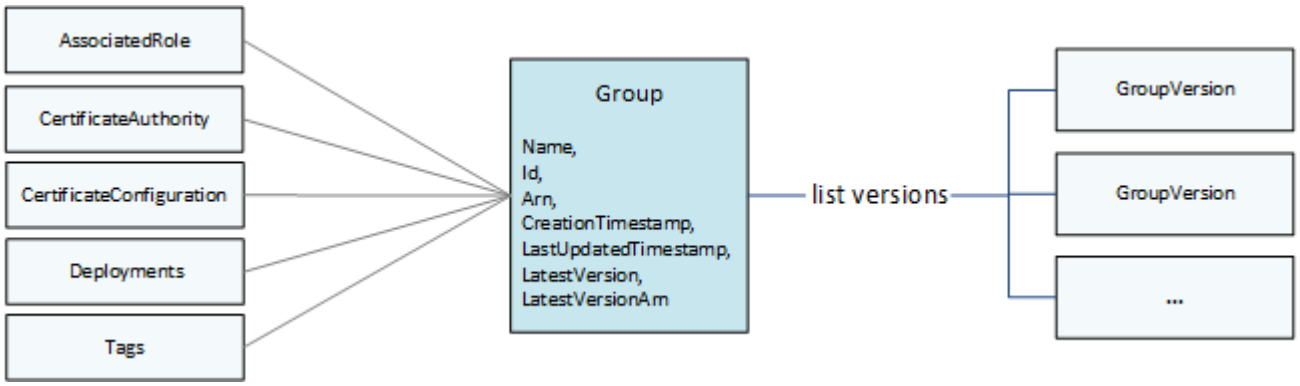
```
aws greengrass list-groups --region us-east-1
```

Übersicht über das AWS IoT Greengrass-Gruppenobjektmodell

Beim Programmieren mit der AWS IoT Greengrass-API ist es hilfreich, das Greengrass-Gruppenobjektmodell zu verstehen.

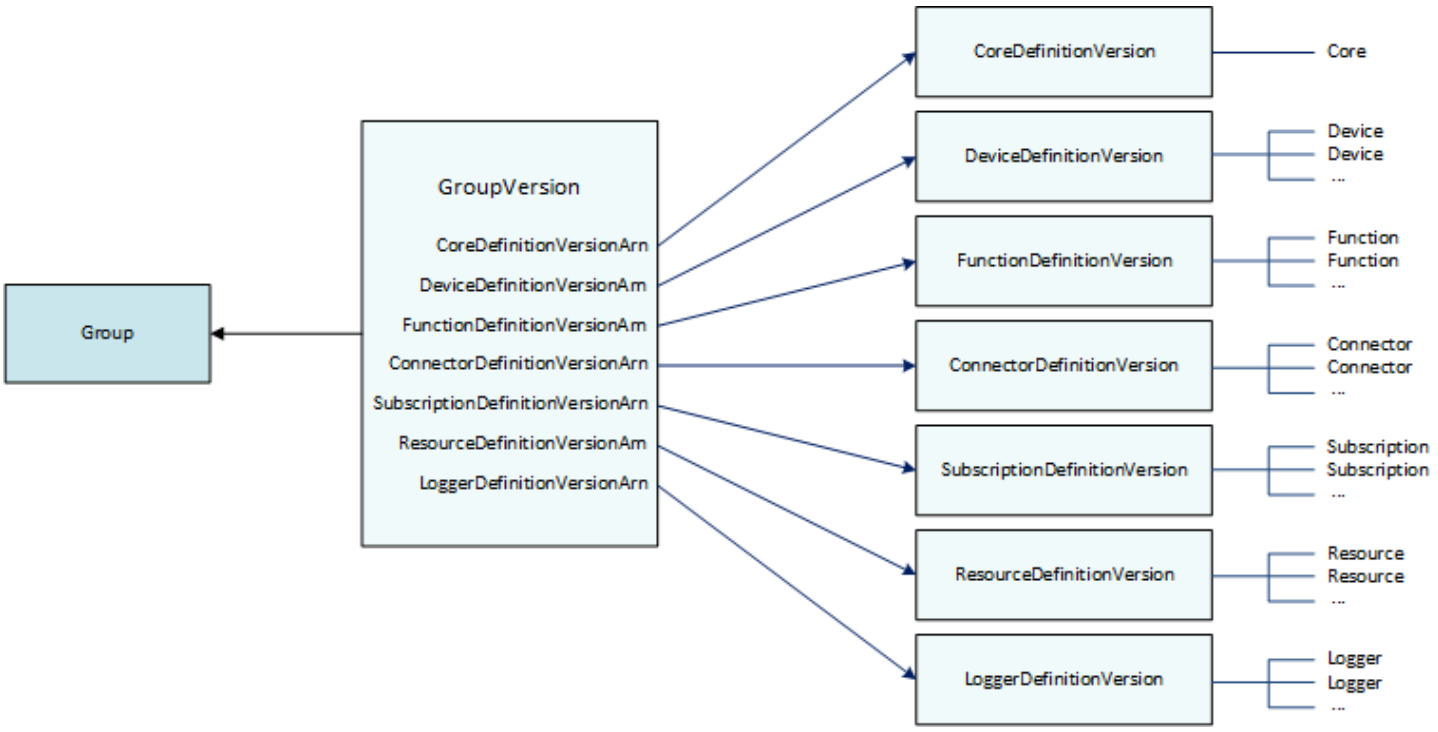
Gruppen

In der AWS IoT Greengrass-API besteht das `Group`-Objekt der obersten Ebene aus Metadaten und einer Liste von `GroupVersion`-Objekten. `GroupVersion`-Objekte sind mit einer `Group`-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, verweisen die `GroupVersion` -Referenzen:

- Die `CoreDefinitionVersion`, die den erforderlichen Core enthält.
- Die `FunctionDefinitionVersion`, die die drei Funktionen enthält.
- Die `DeviceDefinitionVersion`, die 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:

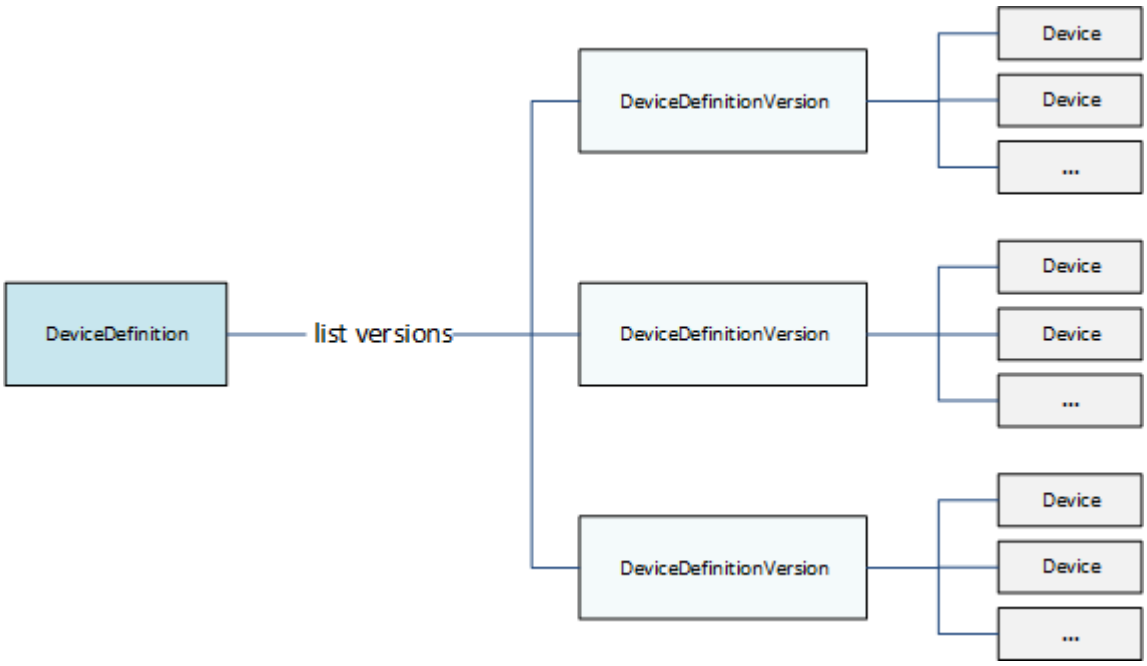
- Eine Definition, die auf eine Liste von DefinitionVersion Objekten eines bestimmten Typs verweist. Beispielsweise verweist eine `DeviceDefinition` auf eine Liste von `DeviceDefinitionVersion`-Objekten.
- Ein DefinitionVersion, der eine Reihe von Entitäten eines bestimmten Typs enthält. Eine `DeviceDefinitionVersion` enthält beispielsweise eine Liste von `Device`-Objekten.

- Einzelne Entitäten, die ihre Eigenschaften und Verhalten definieren. Ein `Device` definiert beispielsweise den ARN des entsprechenden Client-Geräts in der AWS IoT Registrierung, den ARN seines Gerätezertifikats und ob sein lokaler Schatten 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 DefinitionVersion Objekte erstellen, die neue oder aktualisierte Entitäten enthalten.

Sie können neue DefinitionVersions Objekte neuen oder vorhandenen Definition sobjektenzuordnen. 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 eine , `GroupVersion` die alle DefinitionVersion Objekte 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 eine Greengrass-Servicerolle Ihrem AWS-Konto in der zugeordnet seinAWS-Region, in der Sie das bereitstellen`GroupVersion`.

> **Note**
>
> Die `Update`-Aktionen in der API werden verwendet, um den Namen einer `Group` oder eines Komponentendefinitionsobjekts zu ändern.

Aktualisieren von Entitäten, die auf AWS Ressourcen verweisen

Greengrass-Lambda-Funktionen und geheime Ressourcen definieren Greengrass-spezifische Eigenschaften und verweisen auch auf entsprechende AWS Ressourcen. Um diese Entitäten zu aktualisieren, können Sie Änderungen an der entsprechenden AWS Ressource anstelle Ihrer Greengrass-Objekte 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 gepackte Abhängigkeiten zu aktualisieren, nehmen Sie Ihre Änderungen in vorAWS Lambda. Während der nächsten Gruppenbereitstellung werden diese Änderungen von AWS Lambda abgerufen und in Ihre lokale Umgebung kopiert.
- Um Greengrass-spezifische Eigenschaften zu aktualisieren, erstellen Sie eine `FunctionDefinitionVersion`, der die aktualisierten `Function`-Eigenschaften enthält.

> **Note**
>
> Greengrass-Lambda-Funktionen können eine Lambda-Funktion nach Alias-ARN oder Versions-ARN referenzieren. 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 Referenz
- -AWS IoT GreengrassBefehle in der -AWS CLIBefehlsreferenz

Abrufen von Bereitstellungsbenachrichtigungen

Amazon EventBridge -Ereignisregeln bieten Ihnen Benachrichtigungen über Statusänderungen für Ihre Greengrass-Gruppenbereitstellungen. EventBridge bietet einen Stream von Systemereignissen nahezu in Echtzeit, der Änderungen in beschreibtAWSRessourcen schätzen.AWS IoT Greengrasssendet diese Ereignisse an EventBridge auf einemmindestens einmalGrundlage. Dies bedeutet, dassAWS IoT Greengrasssendet möglicherweise mehrere Kopien eines bestimmten

Ereignisses, um die Lieferung sicherzustellen. Außerdem empfangen Ihre Ereignis-Listener die Ereignisse möglicherweise nicht in der Reihenfolge, in der die Ereignisse aufgetreten sind.

> **Note**
>
> Amazon EventBridge ist ein Ereignisbus-Service, über den Sie Ihre Anwendungen mit Daten aus verschiedenen Quellen verbinden können, z.Greengrass-Kerngeräteund Bereitstellungsbenachrichtigungen. Weitere Informationen finden Sie unterWas ist Amazon? EventBridge?imAmazon EventBridge -Benutzerhandbuchaus.

AWS IoT Greengrass gibt ein Ereignis aus, wenn Gruppenbereitstellungen den Status ändern. Sie können ein erstellen EventBridge -Regel, die für alle Statusübergänge oder Übergänge in von Ihnen angegebene Status ausgeführt wird. Wenn eine Bereitstellung einen Status erhält, der eine Regel auslöst, EventBridge ruft 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:

- Starten Sie Vorgänge nach der Bereitstellung, z. B. das Herunterladen von Komponenten und das Benachrichtigen von Mitarbeitern.
- Senden Sie Benachrichtigungen bei erfolgreicher oder fehlgeschlagener Bereitstellung.
- Veröffentlichen Sie benutzerdefinierte Metriken zu Bereitstellungsereignissen.

AWS IoT Greengrass gibt ein Ereignis aus, wenn eine Bereitstellung in folgende Status wechselt: `Building`, `InProgress`, `Success` und `Failure`.

> **Note**
>
> Die Überwachung des Status einer Sammelbereitstellungsoperation wird derzeit nicht unterstützt. Allerdings gibt AWS IoT Greengrass Statusänderungsereignisse für einzelne Gruppenbereitstellungen aus, die Teil einer Sammelbereitstellung 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`aus. Die erste Bereitstellung einer Gruppenversion.
- `ReDeployment`aus. Eine erneute Bereitstellung einer Gruppenversion.
- `ResetDeployment`aus. Löscht Bereitstellungsinformationen, die in der gespeichert sindAWS Cloudund auf derAWS IoT GreengrassCores. Weitere Informationen finden Sie unter the section called “Zurücksetzen von Bereitstellungen”.
- `ForceResetDeployment`aus. Löscht Bereitstellungsinformationen, die in der gespeichert sindAWS Cloudund meldet Erfolg, ohne darauf zu warten, dass der Core reagiert. 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 Bereitstellungsartefakte.
- `InProgress`aus. Die Bereitstellung wird auf der ausgeführtAWS IoT GreengrassCores.
- `Success`aus. Die Bereitstellung war erfolgreich.
- `Failure`aus. 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 das Erstellen von EventBridge Regeln

Bevor Sie ein erstellen EventBridge Regel fürAWS IoT Greengrasswie folgt:

- Machen Sie sich mit den Ereignissen, Regeln und Zielen in vertraut EventBridgeaus.
- Erstellen und konfigurieren Sie die Ziele, die anhand Ihres aufgerufen werden EventBridge Regeln. 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 unterWas ist Amazon? EventBridge?undErste Schritte mit Amazon EventBridgeimAmazon EventBridge -Benutzerhandbuchaus.

Konfigurieren von Bereitstellungsbenachrichtigungen (Konsole)

Führen Sie die folgenden Schritte aus, um einen zu erstellen EventBridge -Regel, die ein Amazon 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 unterErstellen einer EventBridge Regel, die bei einem Ereignis aus einem ausgelöst wirdAWSRessourceimAmazon EventBridge -Benutzerhandbuchaus.

1. Öffnen SieAmazon EventBridgeKonsoleaus.
2. Wählen Sie im Navigationsbereich Rules aus.
3. Wählen Sie Create rule (Regel erstellen).
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 Ereignisbus haben.

5. Wählen Sie für Event bus (Ereignisbus) den Ereignisbus aus, den Sie dieser Regel zuordnen möchten. Wenn Sie möchten, dass diese Regel mit Ereignissen aus Ihrem eigenenKonto übereinstimmt, wählen SieAWSStandard-Ereignisbusaus. Wenn ein AWS-Service in Ihrem Konto ein Ereignis ausgibt, wird dieses stets an den Standardereignisbus Ihres Kontos weitergeleitet.
6. Bei Rule type (Regeltyp) wählen Sie Rule with an event pattern (Regel mit einem Ereignismuster) aus.
7. Wählen Sie Next (Weiter).
8. Als Event source (Ereignisquelle) wählen Sie AWS-Services aus.
9. FürEreignismuster, wählenAWSDienstleistungenaus.
10. FürAWSBedienung, wählen Sie Greengrass.
11. Wählen Sie für Ereignistyp die Option Greengrass Deployment Status Change (Änderung des Status der Greengrass-Bereitstellung).

> **Note**
>
> DieAWSAPI-Aufruf über CloudTrailEreignistyp basiert aufAWS IoT GreengrassIntegration von inAWS CloudTrailaus. Sie können diese Option verwenden, um -Regeln zu erstellen, die durch Lese- oder Schreibaufrufe an denAWS IoT GreengrassAPI. Weitere Informationen finden Sie unter the section called “Protokollierung von AWS IoT Greengrass-API-Aufrufen 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, 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 Next (Weiter).
15. Bei Target types (Zieltypen) wählen Sie AWS-Service aus.
16. FürWählen Sie ein Ziel aus, konfigurieren Sie Ihr Ziel. In diesem Beispiel wird ein Amazon SNS-Thema verwendet, Sie können jedoch auch andere Zieltypen zum 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 Next (Weiter).
17. UnderTags, definieren Sie Tags für die Regel oder lassen Sie die Felder leer.
18. Wählen Sie Next (Weiter).
19. Überprüfen Sie die Details der Regel und wählen Sie dann Create rule (Regel erstellen) aus.

Konfigurieren von Bereitstellungsbenachrichtigungen (CLI)

Führen Sie die folgenden Schritte aus, um einen zu erstellen EventBridge -Regel, die ein Amazon 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.

   - Ersetzen*group-id*mit der ID IhresAWS IoT GreengrassGruppe.

   ```
   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.

   - Ersetzen*thema-arn*mit dem ARN Ihres Amazon SNS-Themas.

   ```
   aws events put-targets \
   ```

```
  --rule TestRule \
  --targets "Id"="1","Arn"="topic-arn"
```

> **Note**
>
> Um Amazon zuzulassen EventBridge um Ihr Zielthema aufzurufen, müssen Sie Ihrem Thema eine ressourcenbasierte Richtlinie hinzufügen. Weitere Informationen finden Sie unterAmazon SNS SNS-BerechtigungenimAmazon EventBridge -Benutzerhandbuchaus.

Weitere Informationen finden Sie unterEreignisse und Ereignismuster in EventBridgeimAmazon EventBridge -Benutzerhandbuchaus.

Konfigurieren von Bereitstellungsbenachrichtigungen (AWS CloudFormation)

Verwenden vonAWS CloudFormationzu erstellende Vorlagen EventBridge -Regeln, die Benachrichtigungen über Statusänderungen für Ihre Greengrass-Gruppenbereitstellungen senden. Weitere Informationen finden Sie unterAmazon EventBridge RessourcentypenreferenzimAWS CloudFormation-Benutzerhandbuchaus.

Weitere Informationen finden Sie auch unter

- Bereitstellen von AWS IoT Greengrass-Gruppen
- Was ist Amazon? EventBridge?imAmazon EventBridge -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 der Kern der Gruppe neu abgebildet 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 Bereitstellungskonfiguration 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 Core-Gerät der Gruppe an, alle ihre bereitstellungsbezogenen Informationen zu bereinigen (Lambda-Funktionen, Benutzerprotokolle, Schattendatenbank und Serverzertifikat, aber nicht die benutzerdefinierten `config.json` oder 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.

Zurücksetzen von Bereitstellungen über die AWS IoT Konsole

Sie können Gruppenbereitstellungen von der Gruppenkonfigurationsseite in der -AWS IoTKonsole aus zurücksetzen.

1. Erweitern Sie im Navigationsbereich der AWS IoTKonsole 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 ein, **`confirm`** um zuzustimmen, und wählen Sie Bereitstellung zurücksetzen aus.

Zurücksetzen von Bereitstellungen mit der AWS IoT Greengrass-API

Sie können die `ResetDeployments`-Aktion in der AWS CLI, derAWS IoT Greengrass-API oder dem 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 nicht übernommen hat, `DeploymentStatus` ist `InProgress`.

Wenn das Zurücksetzen fehlschlägt, werden Fehlerinformationen in der Antwort zurückgegeben.

Weitere Informationen finden Sie auch unter

- Bereitstellen von AWS IoT Greengrass-Gruppen
- ResetDeployments in der AWS IoT Greengrass Version 1 API-Referenz zu
- GetDeploymentStatus in der AWS IoT Greengrass Version 1 API-Referenz zu

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.

Dieses Tutorial beschreibt, wie Sie mit dem AWS CLI eine Gruppen-Sammelbereitstellung in AWS IoT Greengrass erstellen und überwachen können. Das Beispiel für die Sammelbereitstellung in diesem Tutorial enthält mehrere Gruppen. Sie können das Beispiel verwenden, um in Ihrer Implementierung so viele Gruppen hinzufügen, wie Sie benötigen.

Das Tutorial enthält die folgenden allgemeinen Schritte:

1. Erstellen und Hochladen der Eingabedatei für die Sammelbereitstellung

2. Erstellen und Konfigurieren einer IAM-Ausführungsrolle für Sammelbereitstellungen
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 in der gleichen erstellt wurdeAWS-RegionalsAWS IoT Greengrassaus. Weitere Informationen finden Sie unterErstellen und Konfigurieren eines S3-BucketsimAmazon Simple Storage Service — Benutzerhandbuchaus.

> **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 über jede Gruppe in Ihrer Sammelbereitstellung enthält. AWS IoT Greengrass verwendet diese Informationen, um jede Gruppe in Ihrem Namen bereitzustellen, wenn Sie Ihre Sammelbereitstellung 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 inAWS IoTconsole. Die Gruppen-ID wird auf der Seite Einstellungen der Gruppe angezeigt. Gruppenversions-IDs werden in der Gruppe angezeigtBereitstellungen-Registerkarte.

```
aws greengrass list-groups
```

Die Antwort enthält Informationen über jede einzelne 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`-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*, aus dem folgenden Beispiel. Diese Datei enthält Informationen über jede AWS IoT Greengrass-Gruppe, die in eine Sammelbereitstellung einbezogen 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`. Derzeit unterstützt AWS IoT Greengrass nur `NewDeployment` Sammelbereitstellungstypen.

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://my-bucket/
```

Schritt 2: Erstellen und Konfigurieren einer IAM-Ausführungsrolle

In diesem Schritt verwenden Sie die IAM-Konsole, um eine eigenständige Ausführungsrolle zu erstellen. Sie stellen dann eine Vertrauensbeziehung zwischen der Rolle und herAWS IoT Greengrassund stellen Sie sicher, dass Ihr IAM-Benutzer`PassRole`-Berechtigungen für Ihre Ausführungsrolle. Dies ermöglicht es AWS IoT Greengrass, Ihre Ausführungsrolle zu übernehmen und die Bereitstellungen in Ihrem Namen zu 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.

```
{
   "Version": "2012-10-17",
   "Statement": [
       {
           "Sid": "VisualEditor0",
           "Effect": "Allow",
           "Action": "greengrass:CreateDeployment",
           "Resource": [
               "arn:aws:greengrass:region:accountId:/greengrass/groups/groupId1",
               "arn:aws:greengrass:region:accountId:/greengrass/groups/groupId2",
               "arn:aws:greengrass:region:accountId:/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`aws:SourceArn`und`aws:SourceAccount`Globale - Bedingungskontextschlüssel in Ihrer Vertrauensrichtlinie, um zu verhindernconfused StellvertreterSicherheitsproblem. Die Bedingungskontextschlüssel schränken den Zugriff so ein, dass nur Anforderungen zugelassen werden, die vom angegebenen Konto und Greengrass-Arbeitsbereich kommen. Weitere Hinweise zu dem „verwirrten Stellvertreter“ finden Sie unterVermeidung des Problems des verwirrten Stellvertreters (dienstübergreifend)aus.

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "",
         "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:*"
        }
      }
    }
  ]
}
```

3. IAM gebenPassRole-Berechtigungen für Ihre Ausführungsrolle für Ihren IAM-Benutzer. Dieser IAM-Benutzer ist derjenige, der für die Einleitung der Sammelbereitstellung verwendet wird.PassRole-Berechtigungen ermöglichen es Ihrem IAM-Benutzer, Ihre Ausführungsrolle an zu übergebenAWS IoT Greengrasszur Verwendung. Weitere Informationen finden Sie unter Erteilen von Benutzerberechtigungen zur Übergabe einer Rolle an einen AWS-Service.

   Verwenden Sie das folgende Beispiel, um die Ihrer Ausführungsrolle zugeordnete IAM-Richtlinie zu aktualisieren. Ändern Sie dieses Beispiel, falls erforderlich.

```
 {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1508193814000",
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::account-id:user/executionRoleArn"
            ]
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "greengrass.amazonaws.com"
                }
            }
        }
    ]
}
```

Schritt 3: Ermöglichen des Zugriff auf Ihren S3-Bucket für Ihre Ausführungsrolle

Um Ihre Sammelbereitstellung zu starten, muss Ihre Ausführungsrolle in der Lage sein, Ihre Eingabedatei für die Sammelbereitstellung aus Ihrem Amazon S3 S3-Bucket zu lesen. Fügen Sie die folgende Beispielrichtlinie Ihrem Amazon S3 S3-Bucket hinzu, damit seine`GetObject`-Berechtigungen sind für Ihre Ausführungsrolle zugänglich.

Weitere Informationen finden Sie unter Wie füge ich eine S3-Bucket-Richtlinie hinzu?

```
{
    "Version": "2008-10-17",
    "Id": "examplePolicy",
    "Statement": [
        {
            "Sid": "Stmt1535408982966",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "executionRoleArn"
                ]
            },
            "Action": "s3:GetObject",
            "Resource":
            "arn:aws:s3:::my-bucket/objectKey"
        }
    ]
}
```

Sie können den folgenden Befehl in Ihrem Terminal verwenden, um die Richtlinien Ihres Buckets zu überprüfen:

```
aws s3api get-bucket-policy --bucket my-bucket
```

> **Note**
>
> Sie können Ihre Ausführungsrolle direkt ändern, um ihr die -Berechtigungen Ihres Amazon S3 S3-Buckets zu gewährenGetObject-Berechtigungen stattdessen. Fügen Sie dazu Ihrer Ausführungsrolle die folgende Beispielrichtlinie an.
>
> ```
> {
>     "Version": "2012-10-17",
>     "Statement": [
>         {
>             "Sid": "VisualEditor0",
>             "Effect": "Allow",
>             "Action": "s3:GetObject",
>             "Resource": "arn:aws:s3:::my-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`-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 Sammelbereitstellungsoperationen derzeit nicht unterstützt werden, können Sie Amazon erstellen EventBridge -Ereignisregeln, 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`aus. Die Anforderung der Sammelbereitstellung ist eingegangen, und die Ausführung bereitet sich auf den Start vor.
- `Running`aus. Die Sammelbereitstellung wurde gestartet.
- `Completed`aus. Die Ausführung der Sammelbereitstellung hat die Verarbeitung aller Datensätze abgeschlossen.
- `Stopping`aus. Die Ausführung der Sammelbereitstellung hat den Befehl zum Stoppen erhalten und wird in Kürze beendet. Sie können keine neue Sammelbereitstellung starten, solange sich eine frühere Bereitstellung noch nicht im Status `Stopping` befindet.
- `Stopped`aus. Die Ausführung der Sammelbereitstellung wurde manuell gestoppt.
- `Failed`aus. Die Sammelbereitstellung ist auf einen Fehler gestoßen und hat die Ausführung 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.

Check 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.

Überprüfen des AWS IoT Greengrass-Core-Protokolls

Sie können Probleme mithilfe der AWS IoT Greengrass Core-Protokolle beheben. 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 Überwachen mit AWS IoT Greengrass-Protokollen.

Weitere Informationen finden Sie auch unter

Weitere Informationen finden Sie in den folgenden -Ressourcen:

- Bereitstellen von AWS IoT Greengrass-Gruppen
- Amazon S3-API-BefehleimAWS CLIBefehlsreferenz
- AWS IoT GreengrassKommandosimAWS CLIBefehlsreferenz

Ausführen von Lambda-Funktionen auf dem AWS IoT Greengrass Core

AWS IoT Greengrass bietet eine containerisierte Lambda-Laufzeitumgebung für benutzerdefinierten Code, den Sie in erstellenAWS Lambda. Lambda-Funktionen, die in einer -AWS IoT GreengrassCore-Ausführung in der lokalen Lambda-Laufzeit des Cores bereitgestellt werden. Lokale Lambda-Funktionen können durch lokale Ereignisse, Nachrichten aus der Cloud und andere Quellen ausgelöst werden, die Client-Geräten lokale Rechenfunktionen bieten. Sie können beispielsweise Greengrass-Lambda-Funktionen verwenden, um Gerätedaten zu filtern, bevor Sie die Daten in die Cloud übertragen.

Um eine Lambda-Funktion in einem Kern bereitzustellen, fügen Sie die Funktion einer Greengrass-Gruppe hinzu (indem Sie auf die vorhandene Lambda-Funktion verweisen), konfigurieren gruppenspezifische Einstellungen für die Funktion und stellen die Gruppe dann bereit. Wenn die Funktion auf -AWSServices 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, Isolation, Speicherlimits und mehr. Weitere Informationen finden Sie unter the section called “Steuern der Ausführung der Greengrass-Lambda-Funktion”.

> **Note**
>
> Diese Einstellungen ermöglichen auch die Ausführung von AWS IoT Greengrass in einem Docker-Container. Weitere Informationen finden Sie unter the section called “Ausführen von AWS IoT Greengrass in einem Docker-Container”.

In der folgenden Tabelle sind die unterstützten AWS Lambda-Laufzeiten und die Versionen der AWS IoT Greengrass Core-Software aufgelistet, die sie ausführen 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 höher
Java 8	1.1 oder höher
Node.js 12.x*	1.10 oder höher
Node.js 8.10*	1.9 oder höher
Node.js 6.10*	1.1 oder höher
C, C++	1.6 oder höher

* Sie können Lambda-Funktionen ausführen, die diese Laufzeiten auf unterstützten Versionen von verwendenAWS IoT Greengrass, aber Sie können sie nicht in erstellenAWS Lambda. Wenn sich die Laufzeit auf Ihrem Gerät von der für diese Funktion angegebenen AWS Lambda-Laufzeitumgebung unterscheidet, können Sie Ihre eigene Laufzeit mithilfe von `FunctionRuntimeOverride` in auswählen`FunctionDefintionVersion`. Weitere Informationen finden Sie unter CreateFunctionDefinition. Weitere Informationen zu unterstützten Laufzeiten finden Sie unter Laufzeit-Supportrichtlinie im AWS Lambda Entwicklerhandbuch für .

-SDKs für Greengrass-Lambda-Funktionen

AWS bietet drei SDKs, die von Greengrass-Lambda-Funktionen verwendet werden können, die auf einem -AWS IoT GreengrassKern ausgeführt werden. Da diese SDKs in verschiedenen Paketen enthalten sind, können sie von Funktionen gleichzeitig genutzt werden. Um ein SDK in einer Greengrass-Lambda-Funktion zu verwenden, fügen Sie es in das Bereitstellungspaket der Lambda-Funktion ein, das Sie in hochladenAWS Lambda.

AWS IoT Greengrass Core-SDK

Ermöglicht lokalen Lambda-Funktionen die Interaktion mit dem Kern für:

- Austauschen von MQTT-Nachrichten mit AWS IoT Core.
- Tauschen Sie MQTT-Nachrichten mit Connectors, 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 auf bereit GitHub.

- AWS IoT Greengrass Core SDK für Java
- AWS IoT Greengrass Core SDK für Node.js
- AWS IoT Greengrass Core SDK für Python
- AWS IoT Greengrass Core SDK für C

So schließen Sie die AWS IoT Greengrass Core-SDK-Abhängigkeit in das Bereitstellungspaket der Lambda-Funktion ein:

1. Laden Sie die Sprache oder Plattform des AWS IoT Greengrass Core-SDK-Pakets herunter, das der Laufzeit Ihrer Lambda-Funktion entspricht.
2. Entpacken Sie das heruntergeladene Paket, um das SDK zu erhalten. Das SDK ist der `greengrasssdk`-Ordner.
3. Fügen Sie `greengrasssdk` in das Bereitstellungspaket der Lambda-Funktion ein, das Ihren Funktionscode enthält. Dies ist das Paket, in das Sie hochladen, AWS Lambda wenn Sie die Lambda-Funktion erstellen.

StreamManagerClient

Für Stream-Manager-Operationen können nur die folgenden AWS IoT Greengrass Core-SDKs verwendet werden: SDKs

- 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, um sie in Ihre Bereitstellungspakete für Python-Lambda-Funktionen aufzunehmen:

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 des AWS IoT Greengrass Core SDK für Python auf dem Core-Gerät

Wenn Sie Python-Lambda-Funktionen ausführen, können Sie auch verwenden, `pip` um 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 Bereitstellungspaket der Lambda-Funktion 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 Bereitstellungspakete für Lambda-Funktionen aufnehmen.

AWS IoT Greengrass Machine Learning SDK

Ermöglicht lokale Lambda-Funktionen, Machine Learning (ML)-Modelle zu nutzen, die im Greengrass-Kern als ML-Ressourcen bereitgestellt werden. Lambda-Funktionen können das -SDK verwenden, um einen lokalen Inferenzservice aufzurufen und mit diesem zu interagieren, der auf dem Core als Konnektor 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 werden 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 höher	Erstversion.

Informationen zum Download finden Sie unter the section called “AWS IoT Greengrass ML-SDK-Software”.

AWS-SDKs

Ermöglicht lokale Lambda-Funktionen, um direkte Aufrufe an -AWSServices wie Amazon S3, DynamoDBAWS IoT , und zu tätigenAWS 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. Greengrass-Lambda-Funktionen können nicht mit Cloud-Services kommunizieren, wenn der Kern offline ist.

Laden Sie das AWS-SDK aus dem Ressourcencenter für die ersten Schritte herunter.

Weitere Informationen zum Erstellen eines Bereitstellungspakets finden Sie unter the section called “Erstellen und Verpacken einer Lambda-Funktion” im Tutorial Erste Schritte oder Erstellen eines Bereitstellungspakets im AWS Lambda -Entwicklerhandbuch.

Migrieren von cloudbasierten Lambda-Funktionen

Das AWS IoT Greengrass Core SDK folgt dem AWS SDK-Programmiermodell, das das Portieren von Lambda-Funktionen, die für die Cloud entwickelt wurden, zu Lambda-Funktionen vereinfacht, die auf einem -AWS IoT GreengrassKern ausgeführt werden.

Die folgende Python-Lambda-Funktion verwendet beispielsweise die , AWS SDK for 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 GreengrassKern zu portieren, ändern Sie in der `import` Anweisung und `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 Programmiermodellen ermöglicht es Ihnen auch, Ihre Lambda-Funktionen in der Cloud zu entwickeln und sie dann AWS IoT Greengrass mit minimalem Aufwand zu zu migrieren. Lambda-ausführbare 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.

Referenzieren von Lambda-Funktionen nach Alias oder Version

Greengrass-Gruppen können eine Lambda-Funktion nach Alias (empfohlen) oder 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 den Alias einfach auf die neue Funktionsversion. Aliase werden während der Gruppenbereitstellung in 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 AWS IoT GreengrassPrinzip der Versionsunveränderlichkeit entspricht.

Eine gängige Praxis, um Ihre Greengrass-Lambda-Funktionen mit Codeänderungen auf dem neuesten Stand zu halten, ist die Verwendung eines Alias namens **`PRODUCTION`** in Ihrer Greengrass-Gruppe und Ihren Abonnements. Wenn Sie neue Versionen Ihrer Lambda-Funktion in die Produktion heraufstufen, verweisen Sie den Alias auf die neueste stabile Version und stellen Sie die Gruppe erneut bereit. Sie können diese Methode auch für ein Rollback auf eine frühere Version verwenden.

Steuern der Ausführung von Greengrass-Lambda-Funktionen mithilfe einer gruppenspezifischen Konfiguration

AWS IoT Greengrass bietet eine cloudbasierte Verwaltung von Greengrass-Lambda-Funktionen. Obwohl der Code und die Abhängigkeiten einer Lambda-Funktion mit verwaltet werdenAWS Lambda, können Sie konfigurieren, wie sich die Lambda-Funktion verhält, wenn sie in einer Greengrass-Gruppe ausgeführt wird.

Gruppenspezifische Konfigurationseinstellungen

AWS IoT Greengrass stellt die folgenden gruppenspezifischen Konfigurationseinstellungen für Greengrass-Lambda-Funktionen bereit.

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 die Gruppen-ID auswählen, die über die erforderlichen Berechtigungen zum Ausführen der Lambda-Funktion 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 zum Ausführen 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. Wenn Sie als Root ausführen, erhöht sich das folgende Risiko:
>
> - Das Risiko unbeabsichtigter Änderungen, z. B. versehentliches Löschen einer kritischen Datei.
> - Das Risiko für Ihre Daten und Ihr Gerät durch böswillige Personen.
> - Das Risiko von Container-Escapes, wenn Docker-Container mit `--net=host` und ausgeführt werden`UID=EUID=0`.
>
> Wenn Sie eine Ausführung als Root wünschen, müssen Sie die AWS IoT Greengrass-Konfiguration entsprechend aktualisieren. Weitere Informationen finden Sie unter the section called “Ausführen einer Lambda-Funktion als Root”.

Systembenutzer-ID (Nummer)

Die Benutzer-ID für den Benutzer, der über die erforderlichen Berechtigungen zum Ausführen der Lambda-Funktion verfügt. Diese Einstellung ist nur verfügbar, wenn Sie als Andere Benutzer-ID/Gruppen-ID ausführen möchten. Sie können den getent passwd Befehl auf Ihrem AWS IoT Greengrass Core-Gerät verwenden, um die Benutzer-ID zu suchen, die Sie zum Ausführen der Lambda-Funktion verwenden möchten.

Wenn Sie dieselbe UID verwenden, um Prozesse und die Lambda-Funktion auf einem Greengrass-Kerngerät auszuführen, kann Ihre Greengrass-Gruppenrolle den Prozessen temporäre Anmeldeinformationen erteilen. Die Prozesse können die temporären Anmeldeinformationen für alle Greengrass-Kernbereitstellungen verwenden.

Systemgruppen-ID (Nummer)

Die Gruppen-ID für die Gruppe, die über die erforderlichen Berechtigungen zum Ausführen der Lambda-Funktion verfügt. Diese Einstellung ist nur verfügbar, wenn Sie als Andere Benutzer-ID/Gruppen-ID ausführen möchten. Sie können den getent group Befehl auf Ihrem AWS IoT Greengrass Core-Gerät verwenden, um die Gruppen-ID zu suchen, die Sie zum Ausführen der Lambda-Funktion verwenden möchten.

Containerisierung von Lambda-Funktionen

Wählen Sie aus, ob die Lambda-Funktion mit der Standard-Containerisierung 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 seinen Isolationsgrad.

- Containerisierte Lambda-Funktionen werden im Greengrass-Containermodus 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 werden als regulärer Linux-Prozess ohne Isolierung ausgeführt.

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

Wir empfehlen, Lambda-Funktionen in einem Greengrass-Container auszuführen, es sei denn, Ihr Anwendungsfall erfordert, dass sie ohne Containerisierung ausgeführt werden. Wenn Ihre Lambda-Funktionen in einem Greengrass-Container ausgeführt werden, können Sie angeschlossene lokale Ressourcen und Geräteressourcen verwenden und die Vorteile der Isolation und der erhöhten Sicherheit nutzen. Bevor Sie die Containerisierung ändern, schlagen Sie unter the section called “Überlegungen bei der Auswahl der Containerisierung von Lambda-Funktionen” nach.

> **Note**
>
> Um auszuführen, ohne Ihren Geräte-Kernel-Namespace und Ihre Cgroup zu aktivieren, 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 “Festlegen der Standard-Containerisierung für Lambda-Funktionen in einer Gruppe”.

Speicherlimit

Die Speicherzuweisung für die Funktion. Der Standardwert ist 16 MB.

> **Note**
>
> Die Einstellung für das Speicherlimit ist nicht 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 On-Demand oder langlebig sein. Die Standardeinstellung ist „On-Demand“.

Eine On-Demand-Lambda-Funktion beginnt in einem neuen oder wiederverwendeten Container, wenn sie aufgerufen wird. Anforderungen an die Funktion werden möglicherweise von allen verfügbaren Containern verarbeitet. Eine langlebige oder angeheftete Lambda-Funktion wird nach dem AWS IoT Greengrass Start automatisch gestartet und läuft weiterhin in einem eigenen Container (oder einer 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.

Codierungstyp

Der erwartete Kodierungstyp der Eingabenutzlast für die Funktion: JSON oder Binär. Der Standardwert ist JSON.

Unterstützung 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**
>
> Lambda-ausführbare Dateien unterstützen nur den Binärkodierungstyp, nicht JSON.

Prozessargumente

Die Befehlszeilenargumente werden bei der Ausführung an die Lambda-Funktion übergeben.

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 von bis zu 10 lokalen Ressourcen, geheimen Ressourcen und Machine-Learning-Ressourcen???, auf die die Lambda-Funktion zugreifen darf, sowie die entsprechende - `read-only` oder -`read-write`Berechtigung. In der -Konsole werden diese zugehörigen Ressourcen auf der Gruppenkonfigurationsseite auf der Registerkarte Ressourcen aufgeführt.

Der Containerisierungsmodus wirkt sich darauf aus, wie Lambda-Funktionen auf lokale Geräte- und Volume-Ressourcen sowie Machine-Learning-Ressourcen zugreifen können.

- Nicht containerisierte Lambda-Funktionen müssen direkt über das Dateisystem auf dem Core-Gerät auf lokale Geräte- und Volume-Ressourcen zugreifen.
- Damit nicht containerisierte Lambda-Funktionen auf Machine-Learning-Ressourcen in der Greengrass-Gruppe zugreifen können, müssen Sie die Eigenschaften des Ressourcenbesitzers und der Zugriffsberechtigungen für die Machine-Learning-Ressource 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 unter CreateFunctionDefinition in der AWS IoT Greengrass Version 1 API-Referenz oder create-function-definition in der AWS CLI -Befehlsreferenz. Um Lambda-Funktionen auf einem Greengrass-Kern bereitzustellen, erstellen Sie eine Version der Funktionsdefinition, die Ihre Funktionen enthält, erstellen Sie eine Gruppenversion, die auf die Version der Funktionsdefinition und andere Gruppenkomponenten verweist, und stellen Sie dann die Gruppe bereit.

Ausführen einer Lambda-Funktion als Root

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

Bevor Sie eine oder mehrere Lambda-Funktionen als Stamm ausführen können, müssen Sie zuerst die AWS IoT Greengrass Konfiguration aktualisieren, um den Support zu aktivieren. Die Unterstützung 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 Stamm (UID und GID von 0) ausführen und Sie die AWS IoT Greengrass Konfiguration nicht aktualisiert haben. Im Laufzeitprotokoll (*greengrass_root*/ggc/var/log/system/runtime.log) wird ein Fehler ähnlich dem folgenden verzeichnet:

```
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. Wenn Sie als Root ausführen, erhöht sich das folgende Risiko:
>
> - Das Risiko unbeabsichtigter Änderungen, z. B. versehentliches Löschen einer kritischen Datei.
> - Das Risiko für Ihre Daten und Ihr Gerät durch böswillige Personen.
> - Das Risiko von Container-Escapes, wenn Docker-Container mit `--net=host` und ausgeführt werden`UID=EUID=0`.

So erlauben Sie die Ausführung von Lambda-Funktionen als Root

1. Wechseln Sie auf Ihrem AWS IoT Greengrass-Gerät zum Ordner *greengrass-root*/config.

> **Note**
>
> Standardmäßig ist *greengrass-root* das /greengrass-Verzeichnis.

2. Bearbeiten Sie die Datei config.json, indem Sie `"allowFunctionsToRunAsRoot" : "yes"` zum Feld `runtime` hinzufügen. Beispielsweise:

```
{
  "coreThing" : {
    ...
  },
  "runtime" : {
    ...
    "allowFunctionsToRunAsRoot" : "yes"
  },
  ...
}
```

3. Starten Sie AWS IoT Greengrass mit den folgenden Befehlen neu:

```
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 Stamm auszuführen.

Sie können den Wert von `"allowFunctionsToRunAsRoot"` in ändern `"no"` und neu startenAWS IoT Greengrass, wenn Sie die Ausführung von Lambda-Funktionen als Stamm verhindern möchten.

Überlegungen bei der Auswahl 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 GreengrassContainer 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, Lambda-Funktionen in einem Greengrass-Container auszuführen, es sei denn, Ihr Anwendungsfall erfordert, dass sie ohne Containerisierung ausgeführt werden. Durch die Ausführung Ihrer Lambda-Funktionen in einem Greengrass-Container haben Sie mehr Kontrolle über die Einschränkung des Zugriffs auf -Ressourcen.

Es folgen einige Beispiel-Anwendungsfälle für die Ausführung ohne Containerisierung:

- Sie möchten AWS IoT Greengrass auf einem Gerät ausführen, das den Container-Modus nicht unterstützt (weil Sie beispielsweise eine spezielle Linux-Verteilung verwenden oder die Kernel-Version zu alt ist).
- Sie möchten Ihre Lambda-Funktion in einer anderen Container-Umgebung mit eigenem OverlayFS ausführen, aber es treten OverlayFS-Konflikte auf, wenn Sie in einem Greengrass-Container ausführen.
- 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 es sind Probleme aufgetreten, wenn Sie sie als containerisierte Lambda-Funktion ausführen.

Containerisierungsunterschiede

Containerisierung	Hinweise
Greengrass-Container	• Alle AWS IoT Greengrass Funktionen sind verfügbar, wenn Sie eine Lambda-Funktion in einem Greengrass-Container ausführen. • Lambda-Funktionen, die in einem Greengras s-Container ausgeführt werden, haben keinen Zugriff auf den bereitgestellten Code anderer Lambda-Funktionen, auch wenn sie mit derselben Gruppen-ID ausgeführt werden. Mit anderen Worten, Ihre Lambda-Funktionen werden mit einer größeren Isolation voneinander ausgeführt. • Da Lambda-Funktionen, die in einem - AWS IoT GreengrassContainer ausgeführt werden, alle untergeordneten Prozesse im selben Container wie die Lambda-Funktion

Containerisierung	Hinweise
	ausgeführt werden, werden die untergeordneten Prozesse beendet, wenn die Lambda-Funktion beendet wird.

Containerisierung	Hinweise
Kein Container	• Die folgenden Funktionen sind für nicht containerisierte Lambda-Funktionen nicht verfügbar: • Speicherlimits 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 Machine-Learning-Ressource zugreift, müssen Sie einen Ressourcenbesitzer identifizieren und Zugriffsberechtigungen für die Ressource festlegen, nicht für die Lambda-Funktion. Dazu ist AWS IoT Greengrass Core-Software v1.10 oder höher erforderlich. 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) auslösen, z. B. mit dem Nohup-Dienstprogramm, werden nicht automatisch von beendet, AWS IoT Greengrass wenn die übergeordnete Lambda-Funktion beendet wird.

> **Note**
>
> Die Standardeinstellung für Containerisierung für die Greengrass-Gruppe gilt nicht für Konnektoren.

Das Ändern der Containerisierung für eine Lambda-Funktion kann zu Problemen führen, wenn Sie sie bereitstellen. Wenn Sie Ihrer Lambda-Funktion lokale Ressourcen zugewiesen haben, die mit Ihren neuen Containerisierungseinstellungen nicht mehr verfügbar sind, schlägt die Bereitstellung fehl.

- Wenn Sie eine Lambda-Funktion von der Ausführung in einem Greengrass-Container in die Ausführung ohne Containerisierung ändern, werden 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 ohne Containerisierung in eine Ausführung in einem Container ändern, verliert Ihre Lambda-Funktion 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 konfigurieren, bevor Sie sie bereitstellen.

So ändern Sie die Containerisierungseinstellungen für eine Lambda-Funktion

1. Erweitern Sie im Navigationsbereich der AWS IoTKonsole 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 aus.
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 die CreateFunctionDefinition und CreateFunctionDefinitionVersion in der AWS IoT Greengrass API-Referenz zu 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 `MemorySize` Parameter löschen.

Bestimmen der von Ihrem Greengrass-Gerät unterstützten Isolierungsmodi

Mithilfe des AWS IoT Greengrass-Abhängigkeitsprüfers können Sie bestimmen, welche Isolierungsmodi (Greengrass-Container/kein Container) von Ihrem Greengrass-Gerät unterstützt werden.

So führen Sie den AWS IoT Greengrass-Abhängigkeitsprüfer aus

1. Laden Sie den AWS IoT Greengrass Abhängigkeitsprüfer aus dem GitHub Repository herunter und führen Sie ihn aus.

   ```
   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.

Festlegen der Standardzugriffsidentität für Lambda-Funktionen in einer Gruppe

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

Um den Zugriff auf Geräteressourcen besser kontrollieren zu können, können Sie die Standardzugriffsidentität konfigurieren, die zum Ausführen von Lambda-Funktionen in der Gruppe verwendet wird. Diese Einstellung bestimmt die Standardberechtigungen, die Ihren Lambda-Funktionen gewährt werden, wenn sie auf dem Core-Gerät 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 zur Ausführung der zugrunde liegenden AWS IoT Greengrass Core-Software verwendet. Dies besteht aus System-Lambda-Funktionen, die Vorgänge wie Nachrichtenweiterleitung, lokale Schattensynchronisierung und automatische IP-Adresserkennung verwalten.

Die Standardzugriffsidentität kann so konfiguriert werden, dass sie als Konten des AWS IoT Greengrass-Standardsystems (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 zum Ausführen benutzerdefinierter oder System-Lambda-Funktionen verwendet werden.

So ändern Sie die Standardzugriffsidentität für Ihre AWS IoT Greengrass-Gruppe

1. Erweitern Sie im Navigationsbereich der AWS IoTKonsole 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 Standardumgebung der Lambda-Funktionslaufzeit die Option Bearbeiten aus.
4. Wählen Sie auf der Seite Standardumgebung der Lambda-Funktionslaufzeit 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 für 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 für Systemgruppen-ID (Nummer) die Gruppen-ID für die Gruppe ein, die standardmäßig über die Berechtigungen verfügt, die Sie 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 “Ausführen einer Lambda-Funktion als Root”.

Die Änderungen werden wirksam, wenn die Gruppe bereitgestellt wird.

Festlegen der Standard-Containerisierung für Lambda-Funktionen in einer Gruppe

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-Containermodus 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 Standardeinstellung der Gruppe ausgeführt werden. Bevor Sie die Containerisierungseinstellungen ändern, schlagen Sie unter the section called “Überlegungen bei der Auswahl 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.

So ändern Sie die Containerisierungseinstellungen für Ihre AWS IoT Greengrass-Gruppe

1. Erweitern Sie im Navigationsbereich der AWS IoTKonsole 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 aus.
4. Wählen Sie unter Standardumgebung der Lambda-Funktionslaufzeit die Option Bearbeiten aus.
5. Ändern Sie auf der Seite Bearbeiten der standardmäßigen Lambda-Funktionslaufzeitumgebung unter Standard-Lambda-Funktionscontainerisierung die Containerisierungseinstellung.
6. Wählen Sie Speichern.

Die Änderungen werden wirksam, wenn die Gruppe bereitgestellt wird.

Kommunikationsabläufe für Greengrass-Lambda-Funktionen

Greengrass-Lambda-Funktionen unterstützen verschiedene Methoden der Kommunikation mit anderen Mitgliedern der AWS IoT Greengrass Gruppe, lokalen -Services und Cloud-Services (einschließlich AWS-Services).

Kommunikation über MQTT-Nachrichten

Lambda-Funktionen können MQTT-Nachrichten mit einem Publish-Subscribe-Muster senden und empfangen, das von Abonnements gesteuert wird.

Dieser Kommunikationsablauf 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 Kern offline ist, können Greengrass-Lambda-Funktionen Nachrichten mit Client-Geräten, Connectors, anderen Funktionen und lokalen Schatten 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 Volume-Ressourcen und Machine-Learning-Modellen auf einem Core-Gerät zu interagieren, verwenden Greengrass-Lambda-Funktionen 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 Zugreifen auf lokale Ressourcen und the section called “Zugreifen auf Machine-Learning-Ressourcen über Lambda-Funktionscode”.

  > **Note**
  >
  > Wenn Sie Ihre Lambda-Funktion ohne Containerisierung ausführen, können Sie keine angefügten 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 -AWSServices zu kommunizieren. Weitere Informationen finden Sie unter AWS SDK .

- Lambda-Funktionen können Schnittstellen von Drittanbietern verwenden, um mit externen Cloud-Services zu kommunizieren, ähnlich wie bei Cloud-basierten Lambda-Funktionen.

> **Note**
>
> Greengrass-Lambda-Funktionen können nicht mit AWS oder anderen Cloud-Services 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 Connectors in einer Gruppe sowie mit AWS IoT oder dem lokalen Schattenservice 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 von der abrufen kann`context`, die an den Handler übergeben wird. 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 geparst und anschließend das geparste Thema und die Nachricht veröffentlicht.

> **Note**
>
> In der AWS IoT Greengrass-API wird das Thema eines Abonnements durch die `subject`-Eigenschaft dargestellt.

```
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. Anweisungen finden Sie unter the section called “Modul 3 (Teil 1): Lambda-Funktionen aufAWS IoT Greengrass”.

Quelle	Ziel	Themenfilter
IoT Cloud	Diese Funktion	`test/input_message`

Zielhenfil
ter

Test/
Funktion
c_results

Nach Abschluss der Bereitstellung rufen Sie die Funktion auf.

1. Öffnen Sie in der -AWS IoTKonsole die Seite MQTT-Testclient.
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 Lebenszyklus der Greengrass-Lambda-Funktion 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 die Lebenszyklen „On-Demand“ (Standard) oder „Langlebig“:

- 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 senn, 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 angeheftete) 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 Eingabe 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 wird angehalten (z. B. während einer Gruppenbereitstellung oder eines Geräteneustarts) oder die Funktion wechselt in einen Fehlerstatus (z. B. ein Handler-Timeout, eine nicht erkannte Ausnahme oder wenn sie seine Speicherlimits überschreitet).

Weitere Informationen zur Wiederverwendung von Containern finden Sie unter Grundlegendes zur Wiederverwendung von Containern in AWS Lambda im AWS Compute Blog.

Lambda-Ausführungsdateien

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

Eine Lambda-ausführbare 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 Ressourcenbedarf an kompiliertem Code profitieren. Lambda-ausführbare Dateien können durch Ereignisse aufgerufen werden, andere Funktionen aufrufen und auf lokale Ressourcen zugreifen.

Lambda-ausführbare Dateien unterstützen nur den binären Kodierungstyp (nicht JSON). Andernfalls können Sie sie in Ihrer Greengrass-Gruppe verwalten und wie andere Greengrass-Lambda-Funktionen bereitstellen. Der Prozess der Erstellung von Lambda-Ausführungen unterscheidet sich jedoch vom Erstellen von Python-, Java- und Node.js-Lambda-Funktionen:

- Sie können die AWS LambdaKonsole nicht verwenden, um eine Lambda-ausführbare Datei zu erstellen (oder zu verwalten). Sie können eine Lambda-ausführbare Datei nur mithilfe der AWS Lambda API erstellen.
- Sie laden den Funktionscode in 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.

Lambda-ausführbare 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 aufgerufen werden, bevor Threads erstellt und andere AWS IoT Greengrass Core SDK-Funktionen aufgerufen werden.
- Rufen Sie auf`gg_runtime_start`, 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 Codeausschnitt ist die `main` Methode aus dem Codebeispiel simple_handler.c auf 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 AWS IoT Greengrass Core SDK for C.

Erstellen einer ausführbaren Lambda-Datei

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 Lambda-ausführbare 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**
>
> Für können `role`Sie den ARN einer beliebigen Lambda-AWS IoT GreengrassAusführungsrolle angeben. verwendet diese Rolle nicht, aber der Parameter ist erforderlich, um die Funktion zu erstellen. Weitere Informationen zu Lambda-Ausführungsrollen finden Sie unter -AWS LambdaBerechtigungsmodell im AWS Lambda - Entwicklerhandbuch.

```
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 nach 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 LambdaKonsole zeigt keine ausführbaren Lambda-Dateien an. Wenn Sie den Funktionscode aktualisieren, müssen Sie auch die AWS Lambda-API verwenden.

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 AWS IoT Greengrass-API tun.

Ausführen von AWS IoT Greengrass in einem Docker-Container

AWS IoT Greengrass kann zur Ausführung in einem Docker-Container konfiguriert werden.

Sie können ein Dockerfile über Amazon herunterladenCloudFront, auf dem dieAWS IoT Greengrass Core-Software und die 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.

Damit Sie mit AWS IoT Greengrass zu experimentieren beginnen können, stellt AWS auch vorgefertigte Docker-Images bereit, auf denen die AWS IoT Greengrass Core-Software und die 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 AArch64)-Basis-Images.

> **Important**
>
> Am 30. Juni 2022 wurde die Wartung für die Docker-Images derAWS IoT Greengrass Core-Software v1.xAWS IoT Greengrass beendet, 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. Die Docker-Images derAWS IoT Greengrass Core-Software v1.x erhalten jedoch nach dem Ende der Wartungsarbeiten am 30. Juni 2022 keine Sicherheitspatches oder Bugfixes mehr. Wenn Sie einen Produktions-Workload ausführen, der von diesen Docker-Images abhängt, empfehlen wir Ihnen, Ihre eigenen Docker-Images

mit den bereitgestellten Dockerfiles zuAWS IoT Greengrass erstellen. Weitere Informationen finden Sie unter AWS IoT Greengrass Docker-Software.

In diesem Thema wird beschrieben, wie Sie dasAWS 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. Abrufen desAWS IoT Greengrass Container-Image von Amazon ECR
2. Erstellen und Konfigurieren von Greengrass-Gruppe und Core
3. Lokales Ausführen von AWS IoT Greengrass
4. Konfigurieren der "Kein Container"-Containerisierung für die Gruppe
5. Lambda-Funktionen im Docker-Container bereitstellen
6. (Optional) Stellen Sie Client-Geräte, die mit Greengrass interagieren, im Docker-Container bereit

Die folgenden Funktionen werden nicht unterstützt, wenn 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 “AWSVon bereitgestellte Greengrass-Konnektoren”. 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 im Core zugreifen.

Diese Funktionen werden nicht unterstützt, wenn die Lambda-Laufzeitumgebung für die Greengrass-Gruppe auf Kein Container gesetzt ist, was für die AusführungAWS IoT Greengrass in einem Docker-Container erforderlich ist.

Voraussetzungen

Bevor Sie mit diesem Lernprogramm beginnen, müssen Sie die folgenden Schritte tun.

- Sie müssen die folgende Software und Versionen auf Ihrem Host-Computer installieren, basierend auf der von Ihnen ausgewähltenAWS Command Line Interface (AWS CLI) Version.

  AWS CLI version 2

  - Docker Version 18.09 oder höher. Frühere Versionen funktionieren möglicherweise auch, wir empfehlen jedoch 18.09 oder höher.
  - AWS CLIVersion.
    - Informationen zur Installation derAWS CLI Version 2 finden Sie unter Installation derAWS CLI Version 2.
    - Informationen zurAWS CLI Konfiguration von finden Sie unter Konfiguration vonAWS CLI.

  > **Note**
  >
  > Um auf einem Windows-Computer auf eine neuereAWS 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 oder höher.
  - pip, Version 18.1 oder höher.
  - AWS CLIVersion 1.17.10 oder später
    - Informationen zur Installation derAWS CLI Version 1 finden Sie unter Installation derAWS CLI Version 1.
    - Informationen zurAWS CLI Konfiguration von finden Sie unter Konfiguration vonAWS CLI.
    - Führen Sie den folgenden Befehl aus, um auf die neuesteAWS CLI Version der Version 1 zu aktualisieren.

      ```
      pip install awscli --upgrade --user
      ```

  > **Note**
  >
  > Wenn Sie die MSI-Installation derAWS CLI Version 1 unter Windows verwenden, beachten Sie Folgendes:

- Wenn die InstallationAWS CLI von Version 1 Botocore nicht installiert, versuchen Sie es mit der Python- und Pip-Installation.
- Um auf eine neuereAWS CLI Version 1 zu aktualisieren, müssen Sie den MSI-Installationsvorgang wiederholen.

- Für den Zugriff Elastic Container Registry (Amazon ECR) -Ressourcen müssen Sie die folgende Erlaubnis erteilen.
  - Amazon ECR setzt voraus, dass Benutzer die`ecr:GetAuthorizationToken` Erlaubnis über eineAWS Identity and Access Management (IAM) -Richtlinie erteilen, bevor sie sich bei einer Registrierung authentifizieren und Images aus einem Amazon ECR-Repository pushen oder pullen können. Weitere Informationen finden Sie unter Beispiele für Amazon ECR-Repository-Richtlinien und Zugriff auf ein Amazon ECR-Repository im Amazon Elastic Container Registry-Benutzerhandbuch.

Schritt 1: Abrufen desAWS 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 derAWS IoT Greengrass Core-Software enthalten die Greengrass Docker-Images Python 2.7 nicht mehr, da Python 2.7 2020 End-of-life erreicht hat 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 für Produktionsgeräte 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.

Für Schritte, die zeigen, wie Sie das`latest` Bild aus Amazon ECR abrufen, wählen Sie Ihr Betriebssystem aus:

Abrufen des Container-Abbilds (Linux)

Führen Sie die folgenden Befehle auf Ihrem Computer-Terminal aus.

1. Melden Sie sich bei derAWS 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
   > ```

3. Aktivieren Sie den symlink- und hardlink-Schutz. Wenn Sie mit dem Ausführen von AWS IoT Greengrass in einem Container experimentieren, können Sie die Einstellungen nur für den aktuellen Startvorgang 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 erforderlich ist, damit die AWS IoT Greengrass-Cloud-Bereitstellung und MQTT-Kommunikation auf Linux funktionieren. Legen Sie in der Datei `/etc/sysctl.conf` für `net.ipv4.ip_forward` "1" fest und laden Sie dann `sysctls` 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 derAWS 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
```

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 derAWS 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 bereits installiert, Sie müssen aber eine Greengrass-Gruppe und einen Greengrass-Core erstellen. Dazu gehören das Herunterladen von Zertifikaten und von der Core-Konfigurationsdatei.

- Führen Sie die Schritte unter the section called “Modul 2: Installieren vonAWS IoT GreengrassCore-Software” aus. Überspringen Sie die Schritte zum Herunterladen und Ausführen derAWS IoT Greengrass Core-Software. Die Software und ihre Laufzeitabhängigkeiten sind im Docker-Image bereits eingerichtet.

Schritt 3: Lokales Ausführen von AWS IoT Greengrass

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.

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. Ersetzen Sie es *path-to-security-files*durch den Pfad zu den Sicherheitsressourcen und ersetzen Sie *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 dieAWS 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 AWS IoT Greengrass und stellen Sie die Zertifikate und die Konfigurationsdatei mit einem Bind-Mount im Docker-Container bereit.

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. Ersetzen Sie es *`path-to-security-files`*durch den Pfad zu den Sicherheitsressourcen und ersetzen Sie *`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 dieAWS 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 AWS IoT Greengrass und stellen Sie die Zertifikate und die Konfigurationsdatei mit einem Bind-Mount im Docker-Container bereit.

   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. Ersetzen Sie es *path-to-security-files*durch den Pfad zu den Sicherheitsressourcen und ersetzen Sie *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 dieAWS 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 AWS IoT Greengrass und stellen Sie die Zertifikate und die Konfigurationsdatei mit einem Bind-Mount im Docker-Container bereit. 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 SieAWS 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 derAWS 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 “Festlegen der Standard-Containerisierung für Lambda-Funktionen in einer Gruppe”.

> **Note**
>
> Standardmäßig verwenden Lambda-Funktionen die Einstellung für die Gruppencontainerisierung. Wenn Sie die Einstellung Kein Container für eine Lambda-Funktion überschreiben, wenn sie in einem Docker-Container ausgeführtAWS IoT Greengrass wird, schlägt die Bereitstellung fehl.

Schritt 5: Lambda-Funktionen imAWS IoT Greengrass Docker-Container bereitstellen

Sie können langlebige Lambda-Funktionen für den Greengrass Docker-Container bereitstellen.

- Folgen Sie den Schritten unterthe section called “Modul 3 (Teil 1): Lambda-Funktionen aufAWS 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, mit denen sie interagieren,AWS IoT Greengrass wenn sie in einem Docker-Container ausgeführt werden.

- Folgen Sie den Schritten unterthe section called “Modul 4: Interagieren mit Client-Geräten in einemAWS IoT GreengrassGruppe”, um Client-Geräte bereitzustellen, die eine Verbindung zum Core herstellen und MQTT-Nachrichten senden.

Anhalten des AWS IoT Greengrass-Docker-Containers

Um den AWS IoT Greengrass-Docker-Container zu beenden, drücken Sie im Terminal- oder Eingabeaufforderungsfenster Strg+C. Diese Aktion wird`SIGTERM` an den Greengrass-Daemon-Prozess gesendet, um den Greengrass-Daemon-Prozess und alle Lambda-Prozesse, die durch den Daemon-Prozess gestartet wurden, herunterzufahren. Der Docker-Container wird mit dem Prozess `/dev/init` als PID 1 initialisiert, wodurch alle verbliebenen Zombieprozesse entfernen werden. Weitere Informationen finden Sie in der Referenz zu docker run.

Fehlerbehebung bei AWS IoT Greengrass in einem Docker-Container

Verwenden Sie die folgenden Informationen für die Problembehandlung mit der AWS IoT Greengrass-Ausführung in einem Docker-Container.

Fehler: Eine interaktive Anmeldung von einem Nicht-TTY-Gerät aus kann nicht durchgeführt werden.

Lösung: Dieser Fehler kann auftreten, wenn Sie den Befehl `aws ecr get-login-password` ausführen. Stellen Sie sicher, dass Sie die neuesteAWS CLI Version 2 oder Version 1 installiert haben. Wir empfehlen dieAWS CLI Version 2 zu verwenden. Weitere Informationen finden Sie unter Installieren der AWS CLI im AWS Command Line Interface-Leitfaden.

Fehler: Unbekannte Optionen: -no-include-email.

Lösung: Dieser Fehler kann auftreten, wenn Sie den Befehl `aws ecr get-login` ausführen. Stellen Sie sicher, dass die neueste AWS CLI-Version (z. B. run: `pip install awscli --upgrade --user`) installiert ist. 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 InterfaceBenutzerhandbuchAWS Command Line Interface unter Installation von unter Microsoft Windows.

Warnung: IPv4 ist deaktiviert. Das Netzwerk wird nicht funktionieren.

Lösung: Sie können diese Warnung oder eine ähnliche Nachricht erhalten, wenn Sie AWS IoT Greengrass auf einem Linux-Computer ausführen. Aktivieren Sie die IPv4-Netzwerkweiterleitung wie in diesem Schritt beschrieben. Die AWS IoT Greengrass-Cloud Bereitstellung und MQTT-Kommunikation sind nicht funktionsfähig, 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 desGetAuthorizationToken Vorgangs ist ein Fehler aufgetreten (AccessDeniedException): User: arn:aws:iam: ::user/ <account-id>is <user-name>not authorized to perform: ecr: onGetAuthorizationToken resource: *

Diese Fehlermeldung kann bei der Ausführung des`aws ecr get-login-password` Befehls auftreten, wenn Sie nicht über ausreichende Berechtigungen verfügen, um auf ein Amazon ECR-Repository zuzugreifen. Weitere Informationen finden Sie unter Beispiele für Amazon ECR-Repository-Richtlinien und Zugriff auf ein Amazon ECR-Repository im Amazon ECR-Benutzerhandbuch.

Hilfe zur allgemeinen Problembehebung für AWS IoT Greengrass finden Sie unter Fehlerbehebung.

Debuggen von 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 die AWS IoT Greengrass Docker-Container ausführen, nachdem Sie einen Bind-Mount mit dem `/greengrass/ggc/var/log`-Verzeichnis erstellt haben. 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 AWS IoT Greengrass-Docker-Container anfügen. 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 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 fürAWS 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 auf Core-Geräten zur lokalen Ausführung bereitstellen. Auf Greengrass-Kernen, auf denen Linux ausgeführt wird, können diese lokal bereitgestellten Lambda-Funktionen und -Connectors auf lokale Ressourcen zugreifen, die physisch auf dem Greengrass-Core-Gerät vorhanden sind. Um beispielsweise mit Geräten zu kommunizieren, die über Modbus oder CanBus verbunden sind, können Sie Ihre Lambda-Funktion so aktivieren, dass sie auf die serielle Schnittstelle des Kerngeräts zugreift. 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 mithilfe der AWS Management Console

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 in den 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 v1.6 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 Greengrass Lambda-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, die für Sensoren und Aktuatoren über GPIO verwendet werden (zum Beispiel `/dev/gpiomem`).
- GPUs, die verwendet werden, um Machine Learning über integrierte GPUs zu beschleunigen (zum Beispiel `/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 für die Durchführung von Machine Learning-Inferenz 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 müssenAWS IoT Greengrass Core Software v1.3 oder höher verwenden. Um Ressourcen für das /proc-Verzeichnis 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 müssenAWS IoT Greengrass Core Software v1.6 oder höher verwenden.
- Sie können nur Lesezugriff für Lambda-Funktionen zulassen, aber keinen Lese-/Schreibzugriff. Diese Zugriffsebene wird von AWS IoT Greengrass verwaltet.

- 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. Vermeiden Sie daher unbedingt Namenskonflikte, 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

EinAWS IoT Greengrass Lambda-Funktionsprozess läuft normalerweise als`ggc_user` und`ggc_group`. Sie können dem Lambda-Funktionsprozess in der lokalen Ressourcendefinition jedoch zusätzliche Dateizugriffsberechtigungen wie folgt erteilen:

- Um die Berechtigungen der Linux-Gruppe hinzuzufügen, der die Ressource gehört, verwenden Sie den`GroupOwnerSetting#AutoAddGroupOwner` Parameter oder Die Option Automatisch Dateisystemberechtigungen der Systemgruppe hinzufügen, der die Ressourcenkonsole gehört.
- Um die Berechtigungen einer anderen Linux-Gruppe hinzuzufügen, verwenden Sie den`GroupOwnerSetting#GroupOwner` Parameter oder Die Konsolenoption Andere Systemgruppe angeben, um Dateisystemberechtigungen hinzuzufügen. Der Wert `GroupOwner` wird ignoriert, wenn `GroupOwnerSetting#AutoAddGroupOwner` auf "true" gesetzt ist.

EinAWS IoT Greengrass Lambda-Funktionsprozess erbt alle Dateisystemberechtigungen von`ggc_user`und`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.

In diesem Tutorial wird beschrieben, wie Sie eine lokale Ressource erstellen und den Zugriff darauf mithilfe der AWS Command Line Interface (CLI) konfigurieren. 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, in dem die AWS Management Console verwendet wird, finden Sie unter So konfigurieren Sie den lokalen Ressourcenzugriff mithilfe der AWS Management Console.

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 Ressourcenname 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-Kerngerät. 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 in der Lambda-Umgebung. Dieser Speicherort befindet sich innerhalb des Containers, indem die Funktion ausgeführt wird.

GroupOwnerSetting: Ermöglicht Ihnen die Konfiguration zusätzlicher Gruppenberechtigungen für den Lambda-Prozess. Dies ist ein optionales Feld. Weitere Informationen finden Sie unter Dateizugriffsberechtigung des Gruppenbesitzers.

GroupOwnerSetting#AutoAddGroupOwner: Wenn „true“, fügt Greengrass den angegebenen Linux-OS-Gruppenbesitzer der Ressource automatisch 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 Berechtigungen dem Lambda-Prozess hinzugefügt werden. 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: Wenn „true“, kann der Lambda-Prozess Lesezugriff auf den `/sys` Ordner auf dem Greengrass-Kerngerät haben. Dies wird in Fällen verwendet, in denen die Greengrass-Lambda-Funktion Geräteinformationen aus lesen muss`/sys`.

Auch hier gibt `CreateFunctionDefinition` den ARN einer Funktionsdefinitionsversion zurück. Der ARN sollte in Ihrer Gruppendefinitionsversion 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"
}
```

Hinzufügen der Lambda-Funktion zur Gruppe

Fügen Sie zum Schluss die Funktion mit `CreateGroupVersion` zur Gruppe hinzu. Beispielsweise:

```
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: TestDirectory und 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/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 mithilfe der AWS Management Console

Diese Funktion ist verfügbar fürAWS IoT GreengrassCore v1.3 und höher.

Sie können Lambda-Funktionen für den sicheren Zugriff auf lokale Ressourcen des Greengrass Core-Geräts, das als Host fungiert, konfigurieren. 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 SieAWS Management Consoleum den Zugriff auf lokale Ressourcen zu konfigurieren, dieAWS IoT Greengrass-Core-Gerät. Es enthält die folgenden allgemeinen Schritte:

1. Bereitstellungspaket für die Lambda-Funktion erstellen
2. Erstellen und Veröffentlichen einer Lambda-Funktion
3. Hinzufügen der Lambda-Funktion zur Gruppe

4. Hinzufügen einer lokalen Ressource zur Gruppe
5. Hinzufügen von Abonnements zur Gruppe
6. Bereitstellen der Gruppe

Ein Tutorial, in dem die AWS Command Line Interface verwendet wird, 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 Core (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: Bereitstellungspaket für die Lambda-Funktion erstellen

In diesem Schritt erstellen Sie ein Bereitstellungspaket für eine Lambda-Funktion. Dabei handelt es sich um eine ZIP-Datei mit dem Code der Funktion und allen zugehörigen Abhängigkeiten. Zudem laden Sie das AWS IoT Greengrass Core SDK herunter, das als Abhängigkeit im Paket berücksichtigt wird.

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. Aus AWS IoT GreengrassCore-SDK Downloads-Seite, laden Sie die AWS IoT GreengrassCore SDK für Python auf Ihrem Computer.
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`file ist das Bereitstellungspaket Ihrer Lambda-Funktion. Nun können Sie eine Lambda-Funktion erstellen und das Bereitstellungspaket hochladen.

Schritt 2: Erstellen und Veröffentlichen einer Lambda-Funktion

In diesem Schritt verwenden Sie die OptionAWS Lambda-Konsole eine Lambda-Funktion erstellen und diese zur Verwendung des Bereitstellungspakets konfigurieren. Anschließend veröffentlichen Sie eine Funktionsversion und erstellen einen Alias.

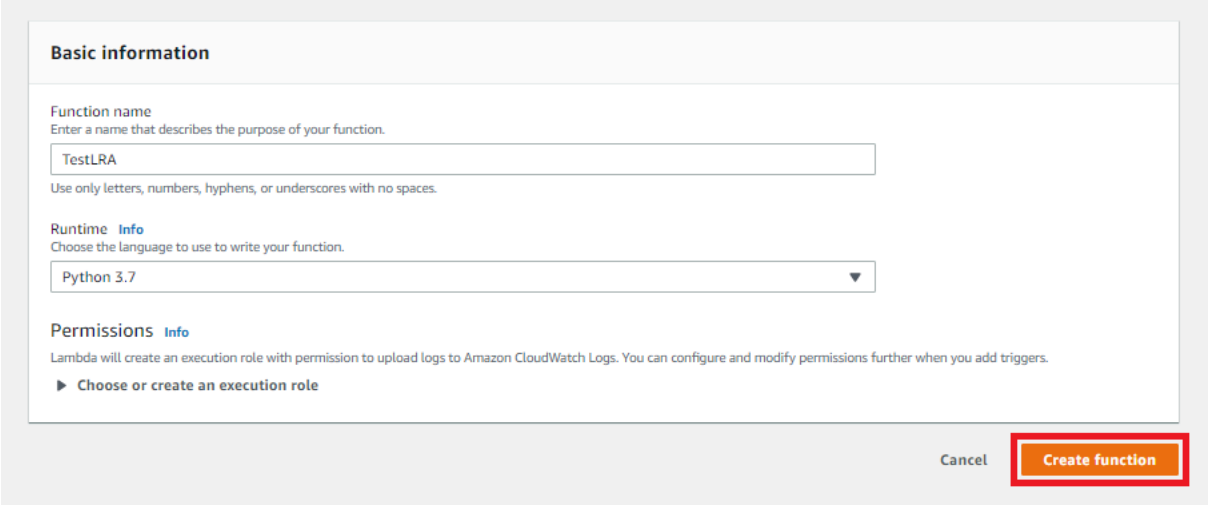
Zunächst erstellen Sie die Lambda-Funktion.

1. Wählen Sie in der AWS Management Console Services und öffnen Sie die AWS Lambda-Konsole.
2. Klicken Sie aufFunktionen.
3. Klicken Sie aufFunktion erstellenund danach auf.Author from scratch.
4. Geben Sie im Abschnitt Basic information (Basisinformationen) folgende Werte ein:

   a. Geben Sie für Function name (Funktionsname) **`TestLRA`** ein.

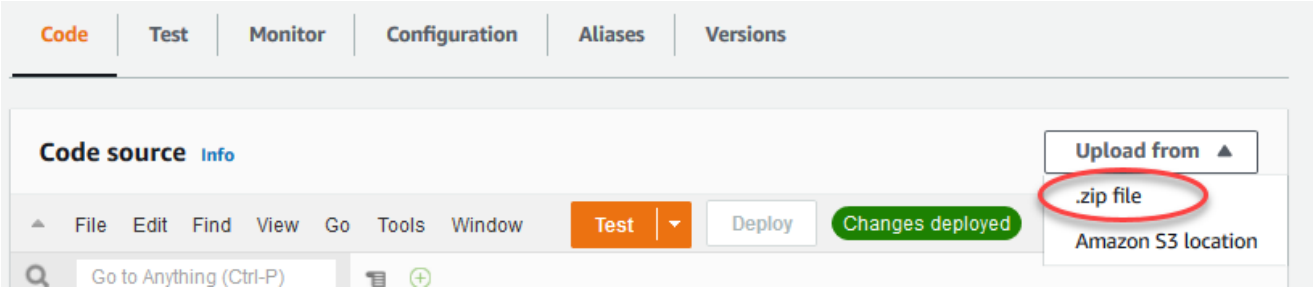
   b. Wählen Sie für Runtime (Laufzeit) die Option Python 3.7 aus.

   c. FürBerechtigungenverwenden, behalten Sie die Standardeinstellung bei. Dadurch wird eine Ausführungsrolle erstellt, die grundlegende Lambda-Berechtigungen gewährt. Diese Rolle wird nicht verwendet vonAWS IoT Greengrass.
5. Wählen Sie Create function (Funktion erstellen).



6. Laden Sie das Bereitstellungspaket Ihrer Lambda-Funktionsbereitstellung hoch und registrieren Sie

   a. Auf derCodeunterQuellcode, wählenHochladen von. Klicken Sie im Dropdown aufZIP-Datei.



b. Klicken Sie aufHochladenund danach auf`lraTestLambda.zip`Bereitstellungspaket. Wählen Sie dann Save (Speichern) aus.

c. Auf derCodefür die Funktion, unterEinstellungen zur Laufzeit, wählenBearbeitenund geben Sie 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 Save (Speichern) aus.

> **Note**
>
> DieTest-Schaltfläche aufAWS Lambda-Konsole funktioniert nicht mit dieser Funktion. DieAWS IoT GreengrassDas Core SDK enthält keine Module, die erforderlich sind, um Ihre Greengrass Lambda-Funktionen unabhängig imAWS Lambdaconsole. Diese Module (zum Beispiel`greengrass_common`) werden für die Funktionen bereitgestellt, nachdem sie in Ihrem Greengrass-Kern bereitgestellt wurden.

Als Nächstes veröffentlichen Sie die erste Version der Lambda-Funktion. Anschließend erstellen Sie einen Alias für die Version.

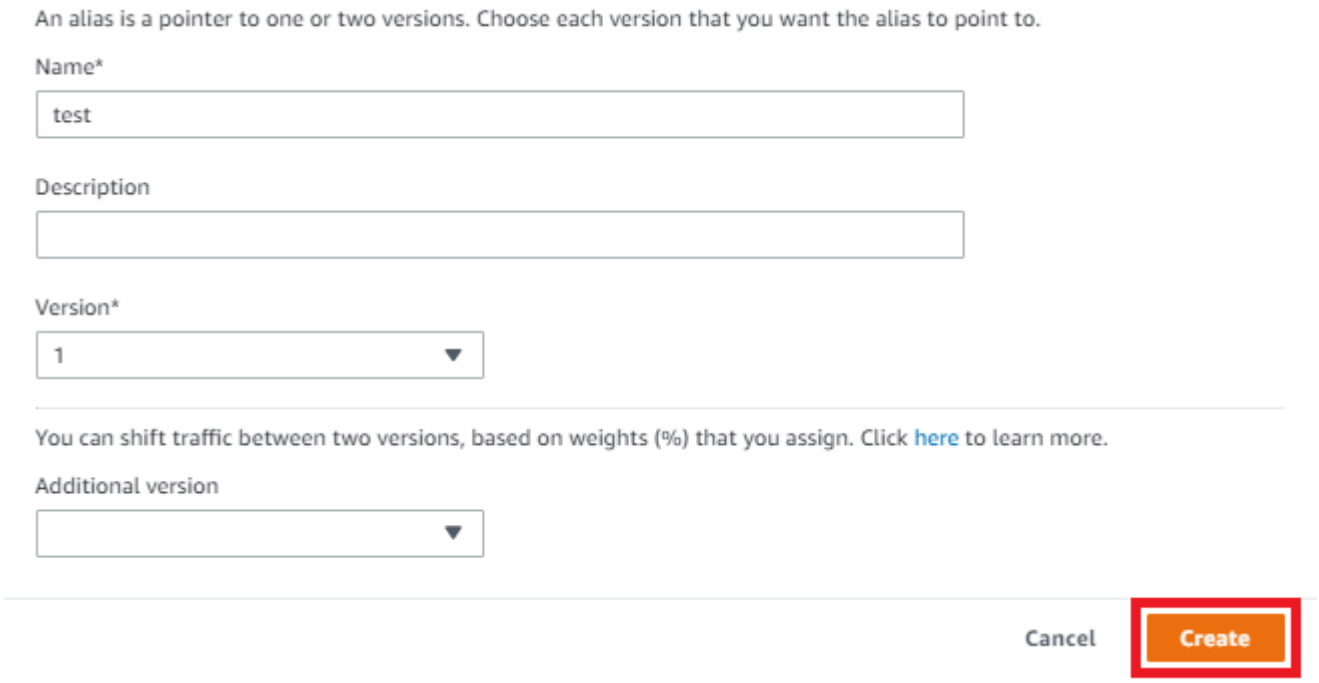
Greengrass-Gruppen können eine Lambda-Funktion nach Alias (empfohlen) oder nach Version referenzieren. Mit einem Alias lassen sich Code-Updates einfacher verwalten, da die Abonnementtabelle oder Gruppendefinition nicht geändert werden muss, wenn der Funktionscode aktualisiert wird. Stattdessen verweisen Sie einfach den 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. Auf derErstellen eines Aliasangezeigten, fürName, den Wert ein**`test`**. Geben Sie für Version 1 ein.

> **Note**
>
> AWS IoT Greengrassunterstützt keine Lambda-Aliasse für$LATESTVersionen.

11. Wählen Sie Create (Erstellen) aus.



Nun können Sie die Lambda-Funktion zur Greengrass-Gruppe hinzufügen.

Schritt 3: Hinzufügen der Lambda-Funktion zur Greengrass-Gruppe

In diesem Schritt fügen Sie die Funktion Ihrer Gruppe hinzu und konfigurieren dann den Lebenszyklus der Funktion.

Zunächst fügen Sie die Lambda-Funktion zur Greengrass-Gruppe hinzu.

1. In derAWS IoTNavigationsbereich der -Konsole unterVerwalten, erweiternGreengrass-Geräteund danach auf.Gruppen (V1).
2. Wählen Sie die Greengrass-Gruppe aus, der die Lambda-Funktion hinzugefügt werden soll.
3. Wählen Sie auf der Gruppenkonfigurationsseite die OptionLambda-FunktionenRegisterkarte.
4. UnDERMeine Lambda-Funktionenden Wert ein.Add.
5. Auf derHinzufügen einer Lambda-Funktionwählen Sie die Option aus.Lambda-Funktion. Select **`TestLRA`**.
6. Wählen Sie das SymbolVersioning der Lambda-Funktion.
7. In derKonfiguration der Lambda-FunktionAbschnitt, wählen SieSystembenutzer und GruppeundContainerisierung der Lambda-Funktion.

   Als Nächstes konfigurieren Sie den Lebenszyklus der Lambda-Funktion.
8. Wählen Sie für Timeout 30 seconds (30 Sekunden).

   > **Important**
   >
   > Lambda-Funktionen, die lokalen 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 SeiteHinzufügen einer Lambda-Funktion.

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 erteilen.

1. Wählen Sie auf der Gruppenkonfigurationsseite die OptionRessourcenRegisterkarte.
2. Zeigen Sie unter dem Verzeichnis die folgenden Dateien an.Lokale Ressourcenden Wert ein.Add.
3. Auf derHinzufügen einer lokalen Ressource-Seite die folgenden Werte an.

a. Geben Sie für Resource name (Ressourcenname) **`testDirectory`** ein.

b. Wählen Sie für Resource type (Ressourcentyp) die Option Volume aus.

c. FürLokaler 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 auf dem Dateisystem des Core-Geräts. 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. UnDERBesitzer der Systemgruppe und Dateizugriffsberechtigung, selectAutomatisch Dateisystemberechtigungen der Systemgruppe hinzufügen, zu der Ressource gehört.

   DieBesitzer der Systemgruppe und Dateizugriffsberechtigungkönnen Sie zusätzliche Dateizugriffsberechtigungen für den Lambda-Prozess erteilen. 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. Mit diesen Abonnements wird die bidirektionale Kommunikation zwischen der Lambda-Funktion und allen zugehörigenAWS IoT.

Zunächst erstellen Sie ein Abonnement für die Lambda-Funktion, damit Nachrichten an gesendet werden könnenAWS IoT.

1. Wählen Sie auf der Gruppenkonfigurationsseite die OptionAbonnementsRegisterkarte.
2. Wählen Sie Add (Hinzufügen) aus.
3. Auf derErstellen eines Abonnements-Seite die Quelle und das Ziel wie folgt:

   a. FürRessourcentyp, wählenLambda-Funktionund danach auf.TestLRA.

b. FürZieltyp, wählenServiceund danach auf.IoT Cloud.

c. FürThemenfilter, den Wert ein**LRA/test**und danach auf.Erstellen eines Abonnements.

4. Auf der Seite Subscriptions wird das neue Abonnement angezeigt.

   Als Nächstes konfigurieren Sie ein Abonnement, mit dem die Funktion aus AWS IoT aufgerufen werden kann.

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. FürRessourcentyp, wählenLambda-Funktionund danach auf.IoT Cloud.

   b. FürZieltyp, wählenServiceund danach auf.TestLRA.

   c. Wählen Sie Next (Weiter).

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: Bereitstellen derAWS IoT GreengrassGruppe

In diesem Schritt stellen Sie die aktuelle Version der Gruppendefinition bereit.

1. Stellen Sie sicher, dass dieAWS IoT Greengrasscore 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 in dem Pfad hängt von der AWS IoT Greengrass-Core-Softwareversion ab, die auf Ihrem Core-Gerät installiert ist.

b. So starten Sie den Daemon:

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

2. Klicken Sie auf der Gruppenkonfigurationsseite aufBereitstellen.

> **Note**
>
> Die Bereitstellung schlägt fehl, wenn Sie Ihre Lambda-Funktion ohne Containerisierung ausführen und versuchen, auf zugeordnete lokale Ressourcen zuzugreifen.

3. Wenn Sie dazu aufgefordert wird, aufLambda-FunktionunterSystem-Lbda-Funktionen, selectIP-Detektorund dannBearbeitenund dannAutomatisch erkennen.

   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.

   > **Note**
   >
   > Erteilen Sie bei Aufforderung die Berechtigung zum Erstellen desGreengrass-Servicerolleund assoziiere es mit deinemAWS-Kontoim aktuellenAWS-Region. Diese Rolle erlaubtAWS IoT Greengrassum auf Ihre -Ressourcen zuzugreifenAWS-Services.

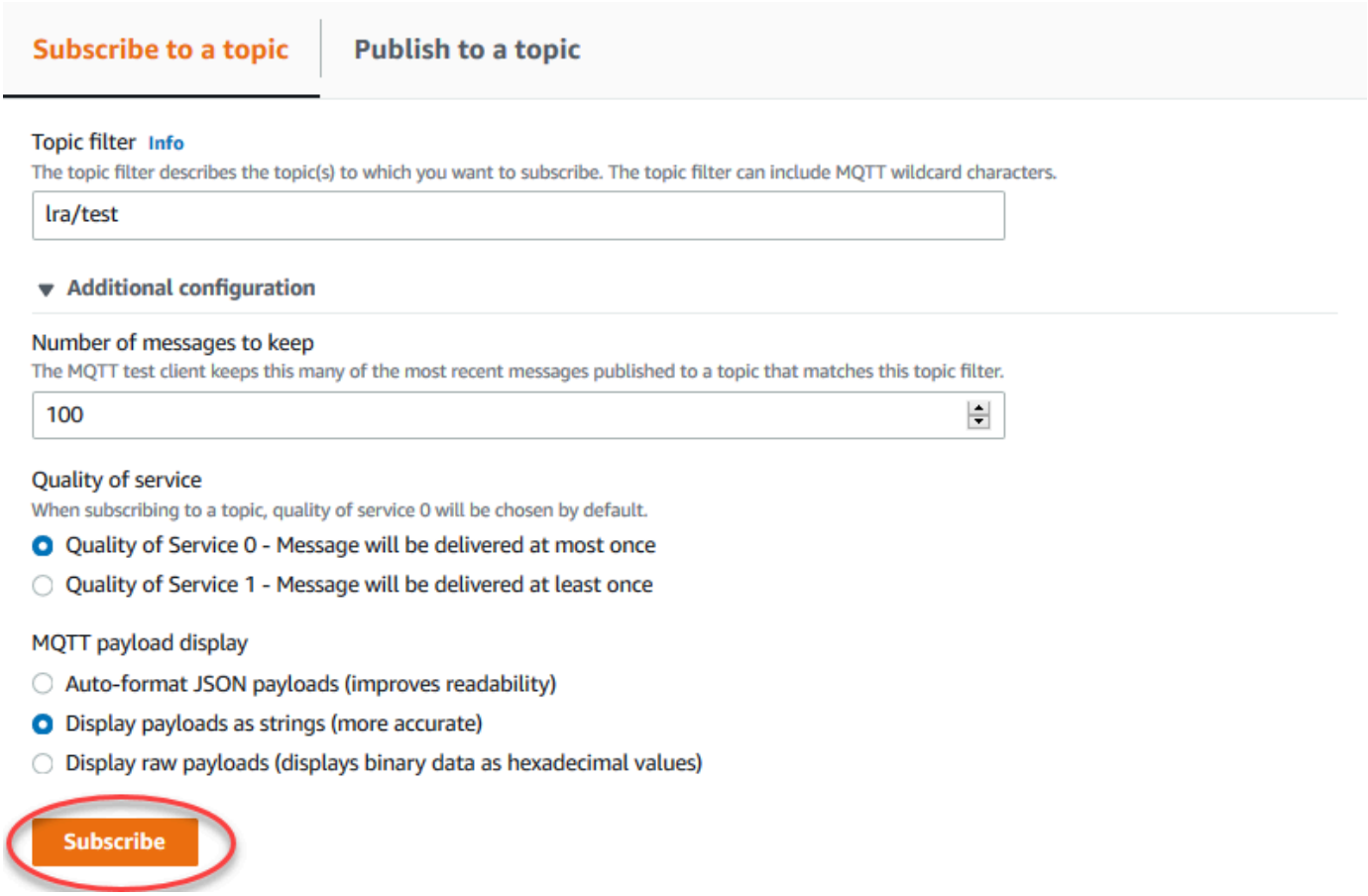
   Auf der Seite Deployments werden der Zeitstempel, die Versions-ID und der Status der Bereitstellung angegeben. Nach Abschluss ist der BereitstellungsstatusCompleted (Abgeschlossen).

   Hilfe zur Problembehebung finden Sie unter Fehlerbehebung.

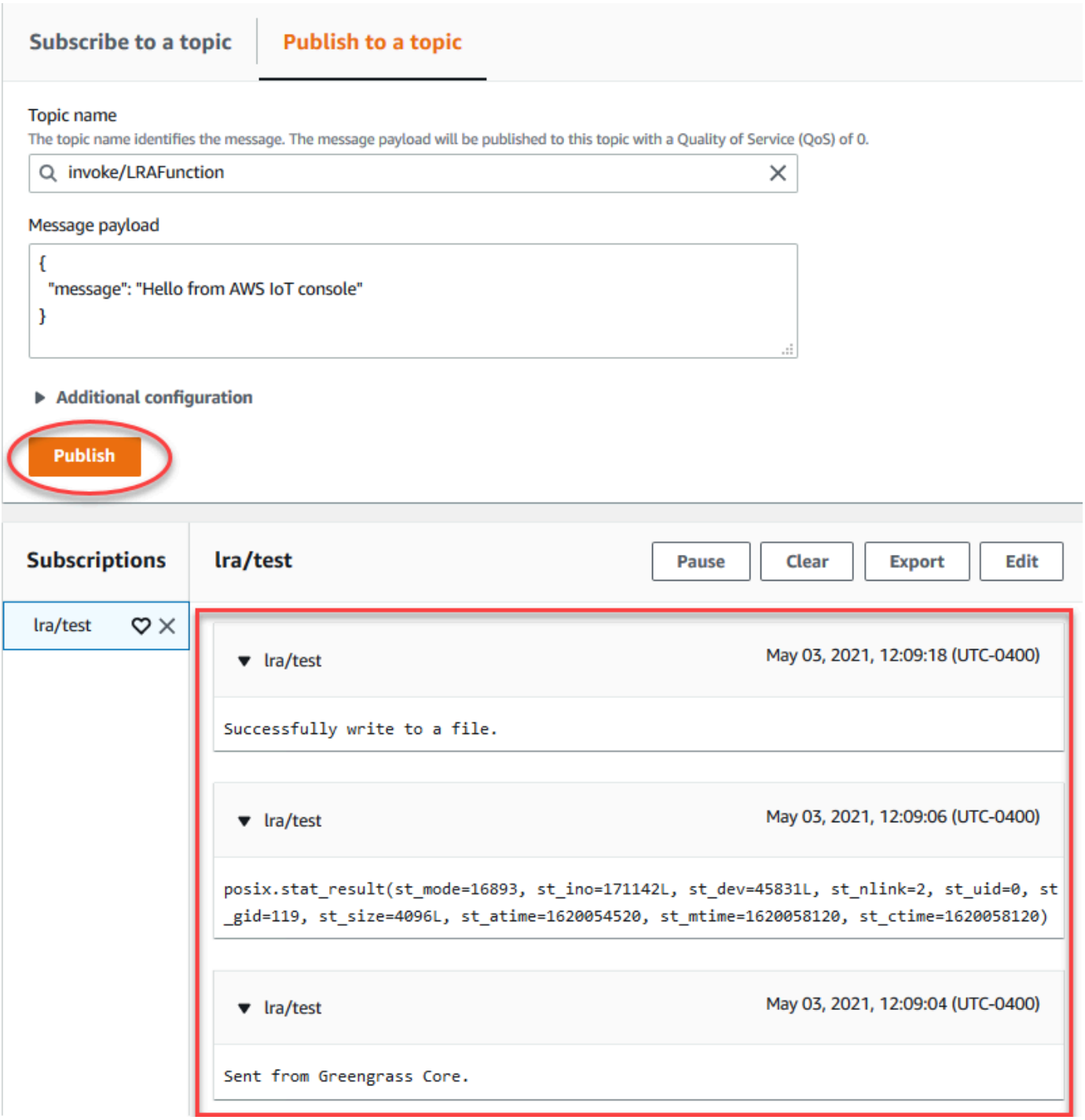
Testen des lokalen Ressourcenzugriffs

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 anAWS IoT.

1. AusAWS IoTNavigationsmenü der -Konsole unterTest, wählen-MQTT-Test-Client.
2. UnDERAbonnieren eines Themas, fürThemenfilter, den Wert ein**`LRA/test`**.
3. UnDERZusätzliche Informationen, fürMQTT-Nutzlast-Anzeige, selectZeigt Nutzlasten als Zeichenfolgen an.
4. Wählen Sie Subscribe aus. Ihre Lambda-Funktion veröffentlicht an das LRA/test-Thema.



5. UnDERVeröffentlichung für ein Thema, in derTopic-Nameeintragen**`invoke/LRAFunction`**und danach auf.Veröffentlichenum Ihre Lambda-Funktion aufzurufen. Der Test ist erfolgreich, wenn auf der Seite die drei Nachrichtennutzlasten der Funktion angezeigt werden.



Die von der Lambda-Funktion erstellte Testdatei befindet sich in der`/src/LRAtest`-Verzeichnis auf dem Greengrass Core-Gerät. Obwohl die Lambda-Funktion in eine Datei im`/dest/LRAtest`Verzeichnis geöffnet haben, 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 verfügbar fürAWS IoT GreengrassCore v1.6 oder höher.

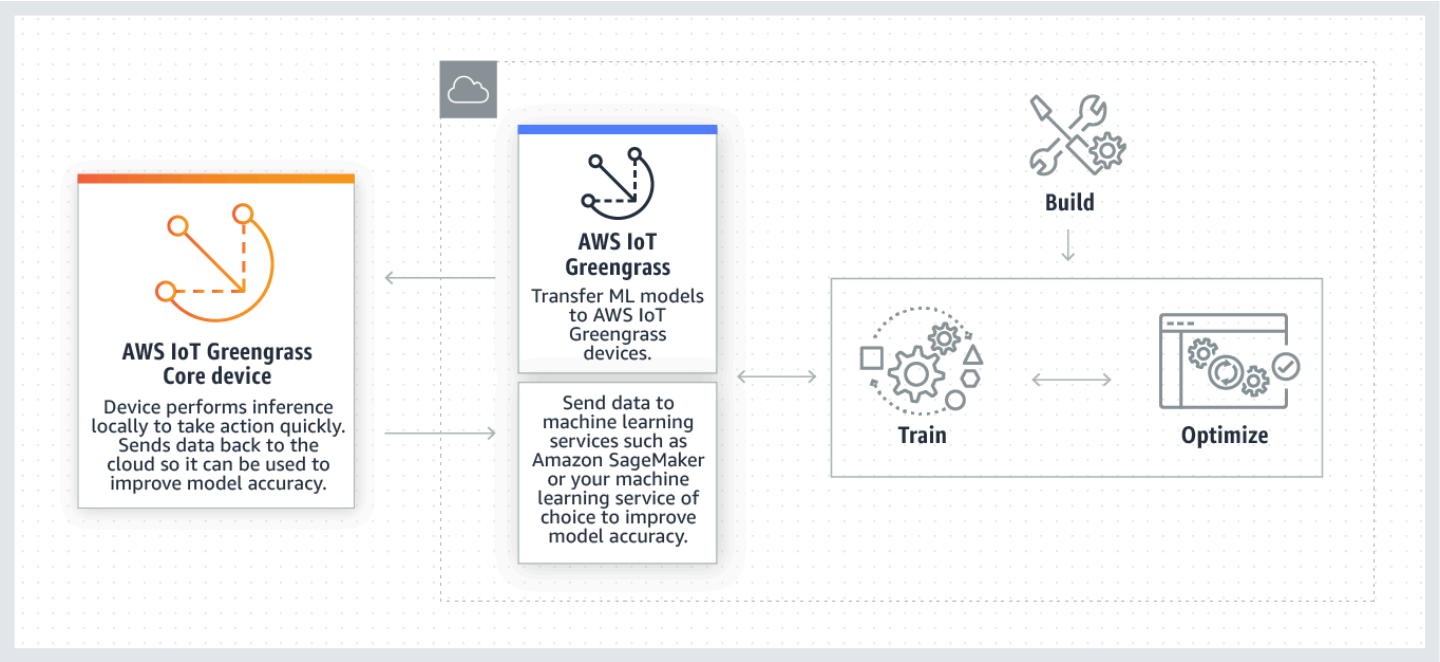
Sie können mit AWS IoT Greengrass Machine Learning (ML)-Inferenz am Edge-Standort mit lokal generierten Daten mithilfe von Cloud-geschulten Modellen durchführen. Sie können von der niedrigen Latenz und Kosteneinsparungen der Ausführung von lokaler Inferenz profitieren und trotzdem die Cloud-Rechenleistung für Schulungsmodelle und komplexe Verarbeitung nutzen.

Hinweise zur Durchführung von lokaler Inferenz finden Sie unter the section called “So konfigurieren Sie Machine Learning-Inferenz”.

Funktionsweise der AWS IoT Greengrass-ML-Inferenz

Sie können Ihre Inferenzmodelle überall schulen und lokal bereitstellenMachine Learning-RessourcenIn einer Greengrass-Gruppe greifen Sie über die Greengrass Lambda-Funktionen zu. Sie können beispielsweise Deep-Learning-Modelle in erstellen und schulenSageMakerund setze sie in deinem Greengrass-Kern ein. Anschließend können Ihre Lambda-Funktionen die lokalen Modelle verwenden, um Inferenz auf verbundene Geräte durchzuführen und neue Schulungsdaten zurück in die Cloud zu senden.

Das folgende Diagramm zeigt den AWS IoT Greengrass-ML-Inferenz-Workflow.



Die AWS IoT Greengrass-ML-Inferenz vereinfacht die einzelnen Schritte des ML-Workflows, darunter:

- Die Entwicklung und Bereitstellung von ML-Framework-Prototypen.
- Der Zugriff auf Cloud-geschulte Modelle und die deren Bereitstellung in Greengrass Core-Geräten.
- Erstellen von Inferenz-Apps, die auf Hardware-Beschleuniger (z. B. GPUs und FPGAs) als lokale Ressourcen zugreifen können.

Machine Learning-Ressourcen

Machine Learning-Ressourcen stehen für Cloud-geschulte Inferenzmodelle, die in einerAWS IoT GreengrassCore-Wert. Um Machine Learning-Ressourcen bereitzustellen, fügen Sie die Ressourcen zuerst einer Greengrass-Gruppe hinzu und definieren dann, wie Lambda-Funktionen in der Gruppe darauf zugreifen können. Während der GruppenbereitstellungAWS IoT Greengrassruft die Quellmodellpakete aus der Cloud ab und extrahiert sie in Verzeichnisse innerhalb des Lambda-Laufzeit-Namespace. Anschließend verwenden Greengrass Lambda-Funktionen die lokal bereitgestellten Modelle für die Durchführung der Inferenz.

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, dann aktualisiert AWS IoT Greengrass das lokale Modell.

Unterstützte Modellquellen

AWS IoT Greengrassunterstützt SageMaker und Amazon S3 S3-Modellquellen für Machine Learning-Ressourcen.

Die folgenden Anforderungen gelten für Modellquellen:

- S3-Eimer, in denen Ihr SageMaker Und Amazon S3 S3-Modellquellen dürfen nicht mit SSE-C verschlüsselt werden. Bei Buckets mit serverseitiger VerschlüsselungAWS IoT GreengrassDerzeit unterstützt die ML-Inferenz nur die SSE-S3- oder SSE-KMS-Verschlüsselungsoptionen. Weitere Informationen zu serverseitigen Verschlüsselungsoptionen finden Sie unterSchützen von Daten mithilfe serverseitiger VerschlüsselungimAmazon Simple Storage Service — Benutzerhandbuchaus.
- Die Namen von S3-Buckets, die Ihre speichern SageMaker Und Amazon S3 S3-Modellquellen dürfen keine Perioden (`.`) enthalten. Weitere Informationen finden Sie in der Regel über die Verwendung von virtuell gehosteten Buckets mit SSL inRegeln für die Bucket-BenennungimAmazon Simple Storage Service — Benutzerhandbuchaus.

- Service-LevelAWS-RegionFür beide muss Unterstützung verfügbar seinAWS IoT Greengrassund SageMakeraus. Derzeit istAWS IoT Greengrassunterstützt SageMaker Modelle in den folgenden Regionen:
  - USA Ost (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)
  - Europe (London)
- AWS IoT Greengrass muss über `read`-Berechtigungen für die Modellquelle, wie in den folgenden Abschnitten beschrieben, verfügen.

SageMaker

AWS IoT Greengrassunterstützt Modelle, die als gespeichert werden SageMaker Schulungsaufträge SageMaker ist ein vollständig verwalteter ML-Service, mit dem Sie Modelle mit vordefinierten oder benutzerdefinierten Algorithmen erstellen und schulen können. Weitere Informationen finden Sie unterWas ist SageMaker?imSageMaker-Entwicklerhaus.

Wenn Sie Ihre konfiguriert haben SageMaker environment vonBucket erstellendessen Name enthält`sagemaker`, dannAWS IoT Greengrasshat ausreichende Erlaubnis, auf Ihr zuzugreifen SageMaker Schulungsaufträge 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 AWS IoT Greengrass die `read`-Berechtigung für den Bucket gewähren, in dem Ihr Schulungsauftrag gespeichert ist. Betten Sie zu diesem Zweck die folgende eingebundene Richtlinie in die Greengrass-Servicerolle ein. Sie können mehrere Bucket-ARNs in einer Liste anzeigen.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::my-bucket-name"
            ]
        }
    ]
}
```

Amazon S3

AWS IoT Greengrassunterstützt Modelle, die in Amazon S3 als gespeichert werden`tar.gz`oder`.zip`Dateien.

So aktivieren SieAWS IoT GreengrassUm auf Modelle zugreifen zu können, die in Amazon S3 S3-Buckets gespeichert sind, müssen Sie gewährenAWS IoT Greengrass `read`Mit der Berechtigung für den Zugriff auf die BucketseinsVon den folgenden:

- 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 Bucket-ARNs in einer Liste anzeigen.

  ```
  {
      "Version": "2012-10-17",
      "Statement": [
          {
              "Effect": "Allow",
              "Action": [
  ```

```
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::my-bucket-name"
            ]
        }
    ]
}
```

Weitere Informationen finden Sie unterEinbetten von eingebundenen RichtlinienimIAM User Guideaus.

Voraussetzungen

Die folgenden Anforderungen gelten für die Erstellung und Verwendung von Machine Learning-Ressourcen:

- Sie müssen verwendenAWS IoT GreengrassCore v1.6 oder höher.
- Benutzerdefinierte Lambda-Funktionen können ausführen`read`oder`read and write`Operationen auf der Ressource. Berechtigungen für andere Operationen sind nicht verfügbar. Der Containerisierungsmodus von verbundenen 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 AWS IoT Greengrass verwenden.

- Amazon SageMaker Neo Deep Learning Runtime
- 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 Neo Deep Learning Runtime

Sie können das SageMaker Neo Deep Learning Runtime für die Durchführung von Inferenzen mit optimierten Machine Learning-Modellen aufAWS IoT GreengrassGeräte. Diese Modelle werden mithilfe der SageMaker Der Neo Deep Learning Compiler zur Verbesserung der Machine Learning-Inferenz-Voraussage. Weitere Informationen zur Modelloptimierung in SageMaker finden Sie imSageMaker Neo-Dokumentationaus.

> **Note**
>
> Derzeit können Sie Machine Learning-Modelle nur mit dem Neo Deep Learning Compiler optimieren. Sie können jedoch die Neo Deep Learning Runtime mit optimierten Modellen in jedem verwendenAWS-RegionwoherAWS IoT Greengrasscore wird unterstützt. Weitere Informationen finden Sie unter So konfigurieren Sie Optimized Machine Learning-Inferenz.

MXNet-Versioning

Apache MXNet stellt derzeit keine Aufwärtskompatibilität sicher, sodass Modelle, die Sie mit späteren Versionen des Frameworks schulen, möglicherweise in früheren Versionen des Frameworks nicht ordnungsgemäß funktionieren. Um Konflikte zwischen den Modellschulungs- und Modellbereitstellungsstufen zu vermeiden und eine konsistente end-to-end erleben, verwenden Sie dieselbe MXNet-Framework-Version in beiden Phasen.

MXNet auf Raspberry Pi

Greengrass Lambda-Funktionen, die auf lokale MXNet-Modelle zugreifen, müssen die folgende Umgebungsvariable festlegen:

```
MXNET_ENGINE_TYPE=NativeEngine
```

Sie können die Umgebungsvariable im Funktionscode festlegen oder zur gruppenspezifischen Konfiguration der Funktion hinzufügen. Ein Bespiel, bei dem dieses im Rahmen der Konfigurationseinstellung hinzugefügt wird, finden Sie in diesem Schritt.

> **Note**
>
> Bei einer allgemeinen Verwendung des MXNet-Frameworks, beispielsweise bei der Ausführung eines Code-Beispiels eines Drittanbieters, muss die Umgebungsvariable auf dem Raspberry Pi konfiguriert werden.

TensorFlow-Modellbereitstellungsbeschränkungen auf dem Raspberry Pi

Die folgenden Empfehlungen zur Verbesserung der Inferenzergebnisse basieren auf unseren Tests mit der 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 TensorFlow-Slim-Abbildklassifikations-Modellbibliothek.
- 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 unterInstallieren von TensorFlowim TensorFlow -Dokumentation.

Zugriff auf Machine-Learning-Ressourcen über Lambda-Funktionen

Benutzerdefinierte Lambda-Funktionen können auf Machine-Learning-Ressourcen zugreifen, um lokale Inferenzen auf dem AWS IoT Greengrass Core auszuführen. 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 Machine-Learning-Ressource im -Kern zugreifen kann, müssen Sie die Ressource an die Lambda-Funktion anhängen und Zugriffsberechtigungen definieren. Der Containerisierungsmodus der zugehörigen (oder angefügten) 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 Machine Learning-Ressource definieren. Der Ressourcenbesitzer stellt die Betriebssystemgruppe und die Berechtigungen dar, die AWS IoT Greengrass zum Herunterladen der Ressourcenartefakte verwendet. Wenn ein Ressourcenbesitzer nicht definiert ist, sind die heruntergeladenen Ressourcenartefakte nur für Root zugänglich.

- Wenn nicht containerisierte Lambda-Funktionen auf eine Machine-Learning-Ressource zugreifen, müssen Sie einen Ressourcenbesitzer definieren, da es keine Berechtigungskontrolle vom Container gibt. Nicht containerisierte Lambda-Funktionen können Berechtigungen von Ressourcenbesitzern erben und sie für den Zugriff auf die Ressource verwenden.

- Wenn nur containerisierte Lambda-Funktionen auf die Ressource zugreifen, empfehlen wir Ihnen, Berechtigungen auf Funktionsebene zu 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 den zusätzlichen Gruppen-IDs der Lambda-Funktion hinzugefügt.

Wenn eine Lambda-Funktion in der Greengrass-Gruppe so konfiguriert ist, dass sie als dieselbe Betriebssystemgruppe wie der Ressourcenbesitzer für eine Machine-Learning-Ressource ausgeführt wird, muss die Ressource an die Lambda-Funktion angehängt werden. Andernfalls schlägt die Bereitstellung fehl, da diese Konfiguration implizite Berechtigungen erteilt, die die Lambda-Funktion verwenden kann, um ohne AWS IoT Greengrass Autorisierung auf die Ressource zuzugreifen. Die Bereitstellungsvalidierungsprüfung wird übersprungen, wenn die Lambda-Funktion als Stamm ausgeführt wird (UID=0).

Wir empfehlen, eine Betriebssystemgruppe zu verwenden, die nicht von anderen Ressourcen, Lambda-Funktionen oder Dateien auf dem Greengrass-Kern verwendet wird. Die Verwendung einer freigegebenen Betriebssystemgruppe erteilt angefügten Lambda-Funktionen mehr Zugriffsberechtigungen, als sie benötigen. Wenn Sie eine freigegebene Betriebssystemgruppe verwenden, muss allen Machine-Learning-Ressourcen, die die freigegebene Betriebssystemgruppe verwenden, auch eine angefügte Lambda-Funktion zugeordnet werden. Andernfalls schlägt die Bereitstellung fehl.

Gruppenbesitzer-Berechtigungen. Die Lese- oder Lese- und Schreibberechtigung zum Hinzufügen zum Lambda-Prozess.

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 Nicht containerisierte Lambda-Funktionen 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 zum Konfigurieren von Lambda-Funktionen und -Ressourcen verwenden, ist auch die `ResourceId` Eigenschaft auf Funktionsebene erforderlich. Die -`ResourceId`Eigenschaft fügt die Machine-Learning-Ressource an die Lambda-Funktion an.

Definieren von Zugriffsberechtigungen für Lambda-Funktionen (Konsole)

In der AWS IoT Konsole definieren Sie Zugriffsberechtigungen, wenn Sie eine Machine-Learning-Ressource konfigurieren oder einer Lambda-Funktion anfügen.

Containerisierte Lambda-Funktionen

Wenn nur containerisierte Lambda-Funktionen an die Machine-Learning-Ressource angefügt sind:

- Wählen Sie Keine Systemgruppe als Ressourcenbesitzer für die Machine-Learning-Ressource aus. Dies ist die empfohlene Einstellung, wenn nur containerisierte Lambda-Funktionen auf

die Machine-Learning-Ressource zugreifen. Andernfalls erteilen Sie angefügten Lambda-Funktionen möglicherweise mehr Zugriffsberechtigungen, als sie benötigen.

Nicht containerisierte Lambda-Funktionen (erfordert CCPC v1.10 oder höher)

Wenn der Machine-Learning-Ressource nicht containerisierte Lambda-Funktionen zugeordnet sind:

- Geben Sie die Systemgruppen-ID (GID) an, die als Ressourcenbesitzer für die Machine-Learning-Ressource 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 Schreibgeschützter Zugriff oder Lese- und Schreibzugriff für die Systemgruppenberechtigungen aus.

Definieren von Zugriffsberechtigungen für Lambda-Funktionen (API)

In der AWS IoT Greengrass API definieren Sie Berechtigungen für Machine-Learning-Ressourcen in der `-ResourceAccessPolicy`Eigenschaft für die Lambda-Funktion oder der `-0wnerSetting`Eigenschaft für die Ressource.

Containerisierte Lambda-Funktionen

Wenn nur containerisierte Lambda-Funktionen an die Machine-Learning-Ressource angefügt sind:

- Definieren Sie für containerisierte Lambda-Funktionen Zugriffsberechtigungen in der `-Permission`Eigenschaft der `-ResourceAccessPolicies`Eigenschaft. Beispielsweise:

```
"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. Beispielsweise:

```
"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 Machine-Learning-Ressource zugreifen. Andernfalls erteilen Sie angefügten Lambda-Funktionen möglicherweise mehr Zugriffsberechtigungen, als sie benötigen.

Nicht containerisierte Lambda-Funktionen (erfordert CCPC v1.10 oder höher)

Wenn der Machine-Learning-Ressource nicht containerisierte Lambda-Funktionen zugeordnet sind:

- Lassen Sie für nicht containerisierte Lambda-Funktionen die Eigenschaft `Permission` in weg`ResourceAccessPolicies`. Diese Konfiguration ist erforderlich und ermöglicht es der Funktion, die Berechtigung auf Ressourcenebene zu erben. Beispielsweise:

```
"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
        }
    }
]
```

- Für containerisierte Lambda-Funktionen, die auch auf die Machine-Learning-Ressource zugreifen, lassen Sie die -`Permission`Eigenschaft in weg `ResourceAccessPolicies` oder definieren Sie eine Berechtigung, die dieselbe oder restriktiver ist wie die -Berechtigung auf Ressourcenebene. Beispielsweise:

```
"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. Beispielsweise:

```
"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 Machine-Learning-Ressourcen über Lambda-Funktionscode

Benutzerdefinierte Lambda-Funktionen verwenden plattformspezifische Betriebssystemschnittstellen, um auf Machine-Learning-Ressourcen auf einem Core-Gerät zuzugreifen.

GGC v1.10 or later

Bei containerisierten Lambda-Funktionen wird die Ressource im Greengrass-Container gemountet und steht unter dem für die Ressource definierten lokalen Zielpfad zur Verfügung. Bei nicht containerisierten Lambda-Funktionen wird die Ressource mit einem Lambda-spezifischen Arbeitsverzeichnis verknüpft und im Lambda-Prozess an die `AWS_GG_RESOURCE_PREFIX` Umgebungsvariable übergeben.

Um den Pfad zu den heruntergeladenen Artefakten einer Machine-Learning-Ressource abzurufen, hängen Lambda-Funktionen die `AWS_GG_RESOURCE_PREFIX` Umgebungsvariable an den lokalen Zielpfad an, der für die Ressource definiert ist. Bei containerisierten 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 auf Machine-Learning-Ressourcen in AWS IoT Greengrass Core v1.9 und früher 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

- InvalidMLModelOwner GroupOwnerSetting –wird in der ML-Modellressource bereitgestellt, aber GroupOwner oder GroupPermission ist nicht vorhanden

- NoContainer Die -Funktion kann die Berechtigung beim Anfügen von Machine Learning-Ressourcen nicht konfigurieren. <function-arn> bezieht sich auf Machine-Learning-Ressource <resource-id> mit Berechtigung <ro/rw> in der Ressourcenzugriffsrichtlinie.
- Funktion <function-arn> bezieht sich auf die Machine Learning-Ressource <resource-id> mit fehlender Berechtigung sowohl in ResourceAccessPolicy als auch in der Ressource OwnerSetting.
- Funktion <function-arn> bezieht sich auf die Machine Learning-Ressource <resource-id> mit der Berechtigung \"rw\", während die Einstellung GroupPermission des Ressourcenbesitzers nur \"ro\" zulässt.
- NoContainer 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

InvalidMLModelOwner GroupOwnerSetting –wird in der ML-Modellressource bereitgestellt, aber GroupOwner oder GroupPermission ist nicht vorhanden

Lösung: Sie erhalten diesen Fehler, wenn eine Machine-Learning-Ressource das -ResourceDownloadOwnerSettingObjekt enthält, aber die erforderliche - `GroupOwner` oder -`GroupPermission`Eigenschaft nicht definiert ist. Um dieses Problem zu beheben, definieren Sie die fehlende Eigenschaft.

NoContainer Die -Funktion kann die Berechtigung beim Anfügen von Machine Learning-Ressourcen nicht konfigurieren. <function-arn> bezieht sich auf Machine-Learning-Ressource <resource-id> mit Berechtigung <ro/rw> in der Ressourcenzugriffsrichtlinie.

Lösung: Sie erhalten diesen Fehler, wenn eine nicht containerisierte Lambda-Funktion Berechtigungen auf Funktionsebene für eine Machine-Learning-Ressource angibt. Nicht containerisierte Funktionen müssen Berechtigungen von den Ressourcenbesitzer-Berechtigungen erben, die für die Machine Learning-Ressource definiert sind. Um dieses Problem zu beheben, erben Sie die Berechtigungen des Ressourcenbesitzers (Konsole) oder entfernen Sie die Berechtigungen aus der Ressourcenzugriffsrichtlinie (API) der Lambda-Funktion.

Funktion <function-arn> bezieht sich auf die Machine Learning-Ressource <resource-id> mit fehlender Berechtigung sowohl in ResourceAccessPolicy als auch in der Ressource OwnerSetting.

Lösung: Sie erhalten diesen Fehler, wenn Berechtigungen für die Machine-Learning-Ressource nicht für die angehängte Lambda-Funktion oder die Ressource konfiguriert sind. Um dieses Problem zu beheben, konfigurieren Sie Berechtigungen in der -ResourceAccessPolicyEigenschaft für die Lambda-Funktion oder der -OwnerSettingEigenschaft für die -Ressource.

Funktion <function-arn> bezieht sich auf die Machine Learning-Ressource <resource-id> mit der Berechtigung \"rw\", während die Einstellung GroupPermission des Ressourcenbesitzers nur \"ro\" zulässt.

Lösung: Sie erhalten diesen Fehler, wenn die für die angefügte Lambda-Funktion definierten Zugriffsberechtigungen die für die Machine-Learning-Ressource 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 Funktion <function-arn> bezieht sich auf Ressourcen des verschachtelten Zielpfads.

Lösung: Sie erhalten diesen Fehler, wenn mehrere Machine-Learning-Ressourcen, die einer nicht containerisierten Lambda-Funktion zugeordnet 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 in , `runtime.log` wenn dieselbe Betriebssystemgruppe als „Als Identität ausführen“ 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 erteilt 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 fügen Sie die Machine-Learning-Ressource 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 Referenz

So konfigurieren Sie Machine Learning-Inferenz mit der AWS Management Console

Um die Schritte in diesem Tutorial ausführen zu können, benötigen SieAWS IoT GreengrassCore 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 dasAWS Management Consoleum eine Greengrass-Gruppe so zu konfigurieren, dass sie eine Lambda-Inferenz-App ausführt, die Kamerabilder lokal erkennt, ohne dass Daten in die Cloud gesendet werden. Die Inferenz-App greift auf das Kameramodul auf einem Raspberry Pi zu und führt Inferenz mithilfe der Open Source ausSqueezeNet-Modell

Das Tutorial enthält die folgenden allgemeinen Schritte:

1. Konfigurieren des Raspberry Pi
2. Installieren des MXNet-Frameworks
3. Erstellen eines Modellpakets
4. Erstellen und Veröffentlichen einer Lambda-Funktion
5. Hinzufügen der Lambda-Funktion zur Gruppe

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 mitAWS 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 ein 2,5A-Netzteilum 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 zuwie man die Kamera aufstellt, finden Sie unterAnschließen der Kamerain der Raspberry Pi-Dokumentation.
- Eine Greengrass-Gruppe und ein Greengrass Core. Weitere Informationen zuwie man eine Greengrass-Gruppe oder einen Greengrass-Cores erstellt, sieheErste Schritte mit AWS IoT Greengrass.

> **Note**
>
> Dieses Tutorial verwendet ein Raspberry Pi, aber AWS IoT Greengrass unterstützt andere Plattformen, wie z. B. Intel Atom und NVIDIA Jetson TX2. Beim Jetson TX2-Beispiel können Sie statische Abbilder anstelle von Abbildern verwenden, die von einer Kamera gestreamt werden. Wenn Sie das Jetson TX2-Beispiel verwenden, müssen Sie möglicherweise Python 3.6 anstelle von Python 3.7 installieren. Weitere Informationen zur Konfiguration des -Geräts finden SieDaskönnen Sie dasAWS IoT GreengrassCore-Software finden Sie unterthe section called “Einrichten anderer Geräte”.

Für Plattformen von Drittanbietern, dieAWS IoT Greengrassunterstützt nicht, Sie müssen Ihre Lambda-Funktion im nicht containerisierten Modus ausführen. Um im nicht containerisierten Modus auszuführen, müssen Sie Ihre Lambda-Funktion als Root-Benutzer ausführen. Weitere Informationen erhalten Sie unter the section called “Überlegungen bei der Auswahl der Containerisierung von Lambda-Funktionen” und the section called “Festlegen 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 des MXNet-Frameworks

Installieren Sie in diesem Schritt die 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 Installing MXNet und folgen Sie den Anweisungen zur Installation von MXNet auf dem Gerät.

   > **Note**
   >
   > Wir empfehlen, Version 1.5.0 zu installieren und MXNet für dieses Tutorial zu erstellen, um Gerätekonflikte zu vermeiden.

3. Überprüfen Sie nach der Installation von 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 Folgendes sicher NumPy ist installiert.

     ```
     sudo -u ggc_user bash -c 'python3 -c "import numpy"'
     ```

Schritt 3: Erstellen eines MXNet-Modellpakets

Erstellen Sie in diesem Schritt ein Modellpaket, das ein vortrainiertes MXNet-Modell zum Hochladen auf Amazon Simple Storage Service (Amazon S3) enthält.AWS IoT Greengrasskann ein Modellpaket von Amazon S3 verwenden, vorausgesetzt, Sie verwenden das Format tar.gz oder zip.

1. Laden Sie auf Ihrem Computer das MXNet-Beispiel für Raspberry Pi von the section called “Beispiele für Machine Learning” herunter.
2. Entpacken Sie die heruntergeladene `mxnet-py3-armv7l.tar.gz`-Datei.
3. Navigieren Sie zum Verzeichnis `squeezenet`.

   ```
   cd path-to-downloaded-sample/mxnet-py3-armv7l/models/squeezenet
   ```

Die `squeezenet.zip`-Datei in diesem Verzeichnis ist Ihr Modellpaket. Sie enthält SqueezeNet Open Source Modellartefakte für ein Bildklassifizierungsmodell. Später laden Sie dieses Modellpaket auf Amazon S3 hoch.

Schritt 4: Erstellen und Veröffentlichen einer Lambda-Funktion

Erstellen Sie in diesem Schritt ein Bereitstellungspaket für Lambda-Funktion und eine Lambda-Funktion. Veröffentlichen Sie anschließend eine Funktionsversion und erstellen einen Alias.

Erstellen Sie zunächst das Bereitstellungspaket für die Lambda-Funktion.

1. Navigieren Sie auf Ihrem Computer zum `examples`-Verzeichnis im Beispielpaket, 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 vonAWS IoT GreengrassCore-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 unterAWS IoT GreengrassCore-SDK für Pythonauf 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. VonAWS IoT-Konsole, die OptionFunktionenundFunktion.
4. Klicken Sie aufAuthor from scratchund verwenden Sie die folgenden Werte zum Erstellen der Funktion:

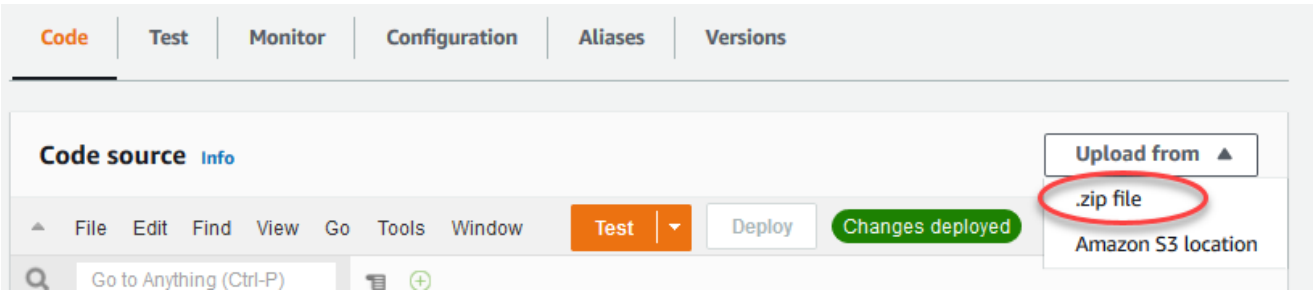
   - Geben Sie für Function name (Funktionsname) **`greengrassObjectClassification`** ein.
   - Wählen Sie für Runtime (Laufzeit) die Option Python 3.7 aus.

   FürBerechtigungenverwenden Sie die Standardeinstellung bei. Dadurch wird eine Ausführungsrolle erstellt, die grundlegende Lambda-Berechtigungen gewährt. Diese Rolle wird nicht verwendetAWS IoT Greengrass.
5. Wählen Sie Create function (Funktion erstellen).

Laden Sie nun das Bereitstellungspaket Ihrer Lambda-Funktion hoch und registrieren Sie den Handler

6. Wählen Sie Ihre Lambda-Funktion und laden Sie Ihr Bereitstellungspaket für Ihre Lambda

   a. Auf derCodeunterQuellcode, wählenHochladen von. Wählen Sie in der Dropdown-Liste die Option.Zip-Datei.



b. Klicken Sie aufHochladenund wählen Sie dann Ihre`greengrassObjectClassification.zip`Bereitstellungspaket Wählen Sie dann Save (Speichern) aus.

c. Auf derCodefür die Funktion, unterEinstellungen zur Laufzeit, wählenBearbeitenund geben Sie dann die folgenden Werte ein.

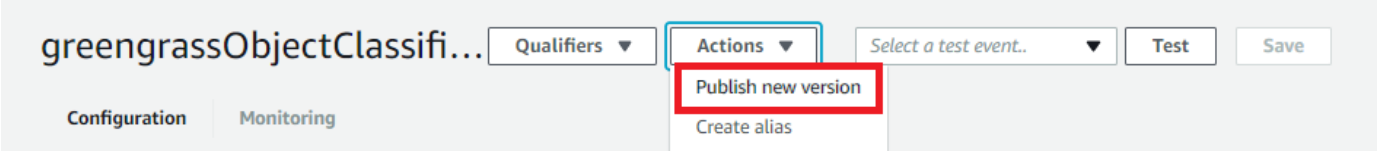
- Wählen Sie für Runtime (Laufzeit) die Option Python 3.7 aus.
- Geben Sie unter Handler **`greengrassObjectClassification.function_handler`** ein.

Wählen Sie Save (Speichern) aus.

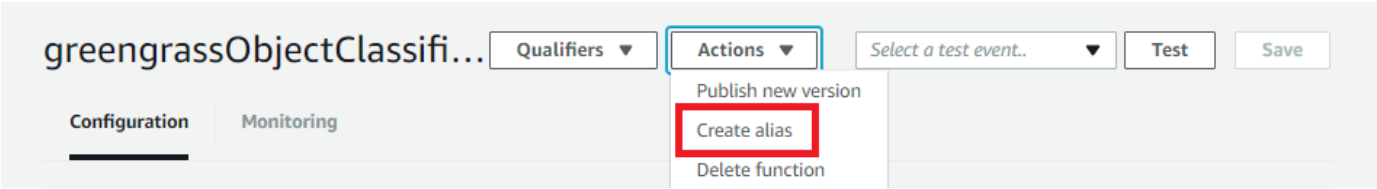
Als Nächstes veröffentlichen Sie die erste Version der Lambda-Funktion. Anschließend erstellen Sie einen Alias für die Version.

> **Note**
>
> Greengrass-Gruppen können eine Lambda-Funktion per Alias (empfohlen) oder nach Version referenzieren. Mit einem Alias lassen sich Code-Updates einfacher verwalten, da die Abonnementtabelle oder Gruppendefinition nicht geändert werden muss, wenn der Funktionscode aktualisiert wird. Stattdessen zeigen Sie einfach den 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. Auf dergreengrassObjectClassification: 1-Konfigurationsseite, von derAktionen-Menü, wählen SieErstellen eines Alias.



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 Greengrassunterstützt keine Lambda-Aliase für$LATEST-Versionen.

11. Wählen Sie Save (Speichern) aus.

Fügen Sie nun die Lambda-Funktion zur Greengrass-Gruppe hinzu.

Schritt 5: Hinzufügen der Lambda-Funktion zur Greengrass-Gruppe

In diesem Schritt fügen Sie die Lambda-Funktion der Gruppe hinzu und konfigurieren dann den Lebenszyklus und die Umgebungsvariablen.

Zunächst fügen Sie die Lambda-Funktion zur Greengrass-Gruppe hinzu.

1. In derAWS IoTNavigationsbereich der -Konsole unterVerwalten, erweiternGreengrass-Geräteund wählen Sie dann aus.Gruppen (V1).
2. Wählen Sie auf der Gruppenkonfigurationsseite die OptionLambda-Funktionender Wert ein.
3. Zeigen Sie unter dem Verzeichnis die folgenden Dateien an.Meine Lambda-FunktionenBereichAdd.
4. Für denLambda-Funktion, wählengreengrassObjectClassification.
5. Für denVersioning der Lambda-Funktion, wählenAlias: mltest.

Konfigurieren Sie dann den Lebenszyklus und die Umgebungsvariablen der Lambda-Funktion.

6. Auf derKonfiguration der Lambda-Funktiondie folgenden Änderungen vor.

> **Note**
>
> Wir empfehlen, dass Sie Ihre Lambda-Funktion ohne Containerisierung ausführen, es sei denn, Ihr Geschäftsfall erfordert dies. Dies ermöglicht den Zugriff auf die GPU und die Kamera Ihres Geräts, ohne die Geräteressourcen zu konfigurieren. Wenn Sie ohne Containerisierung laufen, müssen Sie auch Root-Zugriff auf IhreAWS IoT GreengrassLambda-Funktionen

a. So laufen Sie ohne Containerisierung aus:

- FürSystembenutzer und Gruppe, wählen**`Another user ID/group ID`**. FürSystembenutzer-IDden Wert ein.**`0`**. FürSystemgruppen-IDden Wert ein.**`0`**.

  Auf diese Weise kann Ihre Lambda-Funktion als root ausgeführt werden. Weitere Informationen zum Ausführen als root finden Sie unterthe section called “Festlegen der Standardzugriffsidentität für Lambda-Funktionen in einer Gruppe”.

  > **Tip**
  >
  > Sie müssen auch Ihre`config.json`-Datei, um Root-Zugriff auf Ihre Lambda-Funktion zu gewähren. Die -Prozedur finden Sie unterthe section called “Ausführen einer Lambda-Funktion als Root”.

- FürContainerisierung der Lambda-Funktion, wählenKein Container.

  Weitere Informationen zur Ausführung ohne Containerisierung finden Sie unterthe section called “Überlegungen bei der Auswahl der Containerisierung von Lambda-Funktionen”.

- Geben Sie in Timeout (Zeitüberschreitung) **`10 seconds`** ein.
- FürPinned, wählenWahr.

  Weitere Informationen finden Sie unter the section called “Lebenszyklus-Konfiguration”.

b. Um stattdessen im Containermodus zu laufen:

> **Note**
>
> Wir empfehlen, nicht im Containermodus auszuführen, es sei denn, Ihr Geschäftsfall erfordert dies.

- FürSystembenutzer und Gruppe, wählenGruppen-Standardnamen.
- FürContainerisierung der Lambda-Funktion, wählenGruppen-Standardnamen.
- Geben Sie in Memory Limit (Speicherlimit) **`96 MB`** ein.
- Geben Sie in Timeout (Zeitüberschreitung) **`10 seconds`** ein.
- FürPinned, wählenWahr.

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 als Wert den WertNaiveEngine.

   > **Note**
   >
   > Bei Ihren eigenen benutzerdefinierten Lambda-Funktionen können Sie die Umgebungsvariable auch in Ihrem Funktionscode festlegen.

8. Übernehmen Sie die Standardwerte für alle anderen Eigenschaften und wählen SieRessource Lambda-Funktion.

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. Dies ermöglicht es der Lambda-Funktion, auf die Ressourcen auf dem Core-Gerät zuzugreifen.

> **Note**
>
> Wenn Sie im nicht containerisierten Modus laufen,AWS IoT Greengrasskann 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 OptionRessourcender Wert ein.
2. In derLokale RessourcenBereichRessource „Lokale Ressource“.
3. Auf derEine lokalen Ressourceverwenden Sie die folgenden Werte:

   - Geben Sie für Resource name (Ressourcenname) **`videoCoreSharedMemory`** ein.
   - Wählen Sie in Resource type (Ressourcentyp) die Option Device (Gerät) aus.
   - FürLokaler Gerätepfadden 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.
   - FürEigentümer der Systemgruppe und Dateizugriffsberechtigungen, wählenAutomatisch Dateisystemberechtigungen der Systemgruppe hinzufügen, zu der die Ressource gehört.

     DieEigentümer der Systemgruppe und Dateizugriffsberechtigungenkönnen Sie zusätzliche Dateizugriffsberechtigungen für den Lambda-Prozess erteilen. Weitere Informationen finden Sie unter Dateizugriffsberechtigung des Gruppenbesitzers.
4. Als nächstes fügen Sie eine lokale Geräteressource für die Kameraschnittstelle hinzu.
5. Klicken Sie aufRessource „Lokale Ressource“.
6. Auf derEine lokalen Ressourceverwenden Sie die folgenden Werte:

   - Geben Sie für Resource name (Ressourcenname) **`videoCoreInterface`** ein.
   - Wählen Sie in Resource type (Ressourcentyp) die Option Device (Gerät) aus.
   - FürLokaler Gerätepfadden Wert ein.**`/dev/vchiq`**.
   - FürEigentümer der Systemgruppe und Dateizugriffsberechtigungen, wählenAutomatisch Dateisystemberechtigungen der Systemgruppe hinzufügen, zu der die Ressource gehört.

7. Wählen Sie unten auf der SeiteRessource hinzufügen.

Fügen Sie nun das Inferenz-Modell als Machine Learning-Ressource hinzu. Dieser Schritt umfasst das Hochladen`squeezenet.zip`Modellpaket nach Amazon S3.

1. Auf derRessourcenfür Ihre Gruppe, unter demMaschinelles Lernen-BereichRessource für Machine Learning.
2. Auf derEine Ressource für maschinelles Lernen hinzufügen-SeiteRessourcennameden Wert ein.**`squeezenet_model`**.
3. FürModellschulung, wählenVerwenden Sie ein in S3 gespeichertes Modell, z. B. ein mit Deep Learning Compiler optimiertes Modell.
4. FürS3einen Pfad ein, in dem der S3-Bucket gespeichert wird.
5. Wählen Sie Browse S3 (S3 durchsuchen). Auf diese Weise wird eine neue Registerkarte in der Amazon-S3-Konsole geöffnet.
6. Laden Sie auf der Registerkarte der Amazon-S3-Konsole die Option`squeezenet.zip`Datei in einen S3-Bucket. Weitere Informationen finden Sie unterWie lade ich Dateien und Ordner in einen S3-Bucket hoch?imAmazon Simple Storage Service — Benutzerhandbuch.

   > **Note**
   >
   > Um den Zugriff auf den S3-Bucket zu ermöglichen, muss der Bucket-Name die Zeichenfolge enthalten**`greengrass`**und der -Bucket muss sich in der gleichen -Region befinden, für die SieAWS IoT Greengrass. Wählen Sie einen eindeutigen Namen (wie z. B.: **`greengrass-bucket-`*`user-id`*`-`*`epoch-time`***). Verwenden Sie keinen Punkt (`.`) im Bucket-Namen.

7. Auf derAWS IoT Greengrassauf der -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 Namespace der Lambda-Laufzeit. Wenn Sie die Gruppe bereitstellen, ruft AWS IoT Greengrass das Quellmodellpaket ab und extrahiert dann den Inhalt in das angegebene Verzeichnis. Die Beispiel-Lambda-Funktion für dieses Tutorial ist bereits so konfiguriert, dass sie diesen Pfad verwendet (im`model_path`-Variable)

9. UNEigentümer der Systemgruppe und Dateizugriffsberechtigungen, wählenKeine Systemgruppe.
10. Wählen Sie Add resource (Ressource hinzufügen) aus.

benutzen SageMaker -geschulte Modelle

Dieses Tutorial verwendet ein Modell, das in Amazon S3 gespeichert ist, das Sie jedoch einfach verwenden können SageMaker -Modell auch. DieAWS IoT GreengrassKonsole ist eingebaut SageMaker -Integration, sodass Sie diese Modelle nicht manuell auf Amazon S3 hochladen müssen. Anforderungen und Einschränkungen für die Nutzung SageMaker Modelle finden Sie unterthe section called “Unterstützte Modellquellen”.

Um ein SageMaker -Modell:

- FürModellschulung, wählenVerwenden Sie ein in geschultes ModellAWS SageMakerund wählen Sie dann den Namen des Schulungsauftrags des Modells aus.
- FürZielpfadden 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. Mit diesem Abonnement kann die Lambda-Funktion Prognoseergebnisse senden anAWS IoTindem Sie in einem MQTT-Thema veröffentlichen.

1. Wählen Sie auf der Gruppenkonfigurationsseite die OptionAbonnementsund wählen Sie dann aus.Abo hinzufügen.
2. Auf derEinzelheiten zum Abonnementdie Quelle und das Ziel wie folgt:

   a. In :Ressourcentyp, wählenLambda-Funktionund wählen Sie dann aus.greengrassObjectClassification.

   b. In :Zieltyp, wählenServiceund wählen Sie dann aus.IoT Cloud.
3. In :Themenfilterden Wert ein.**`hello/world`**und wählen Sie dann aus.Abonnement.

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, Ressourcen und Abonnementkonfigurationen, die Sie hinzugefügt haben.

1. Stellen Sie sicher, dassAWS IoT Greengrasscore 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'
   ```

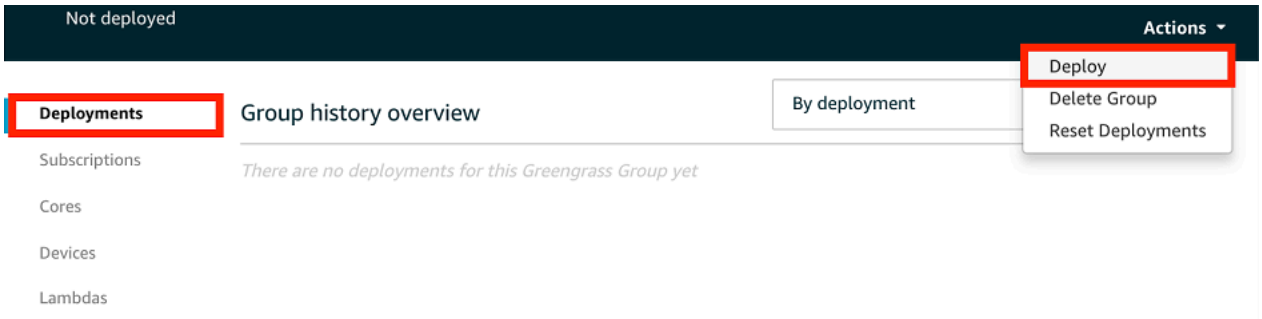
   Wenn die Ausgabe einen `root`-Eintrag für `/greengrass/ggc/packages/1.11.6/bin/daemon` enthält, dann wird der Daemon ausgeführt.

   > **Note**
   >
   > Die Version in dem Pfad hängt von der AWS IoT Greengrass-Core-Softwareversion ab, die auf Ihrem Core-Gerät installiert ist.

   b. So starten Sie den Daemon:

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

2. Wählen Sie auf der Gruppenkonfigurationsseite die OptionBereitstellen.



3. In derLambda-Funktionenunter der RegisterkarteSystem-Lambda-FunktionenAbschnitt, wählen SieIP-Detektorund wähleBearbeiten.
4. In derEinstellungen für IP-Detektor bearbeitenwählen Sie aus.Automatisches Erkennen und Überschreiben von MQTT-Broker-.

5. Wählen Sie Save (Speichern) 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.

   > **Note**
   >
   > Erteilen Sie bei Aufforderung die Berechtigung zum Erstellen desGreengrass-Servicerolleund assoziiere es mit deinemAWS-Kontoim aktuellenAWS-Region. Diese Rolle erlaubtAWS IoT Greengrassum auf Ihre Ressourcen inAWS-Services.

   Auf der Seite Deployments werden der Zeitstempel, die Versions-ID und der Status der Bereitstellung angegeben. Nach abgeschlossener Bereitstellung sollte der Status wie folgt lautenCompleted (Abgeschlossen).

   Weitere Informationen über Bereitstellungen finden Sie unterBereitstellen von AWS IoT Greengrass-Gruppen. Hilfe zur Problembehebung finden Sie unter Fehlerbehebung.

Schritt 9: Testen der Inferenzanwendung

Nun können Sie prüfen, ob die Bereitstellung korrekt konfiguriert ist. Um zu testen, abonnieren Sie dashello/worldund zeigen Sie 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.

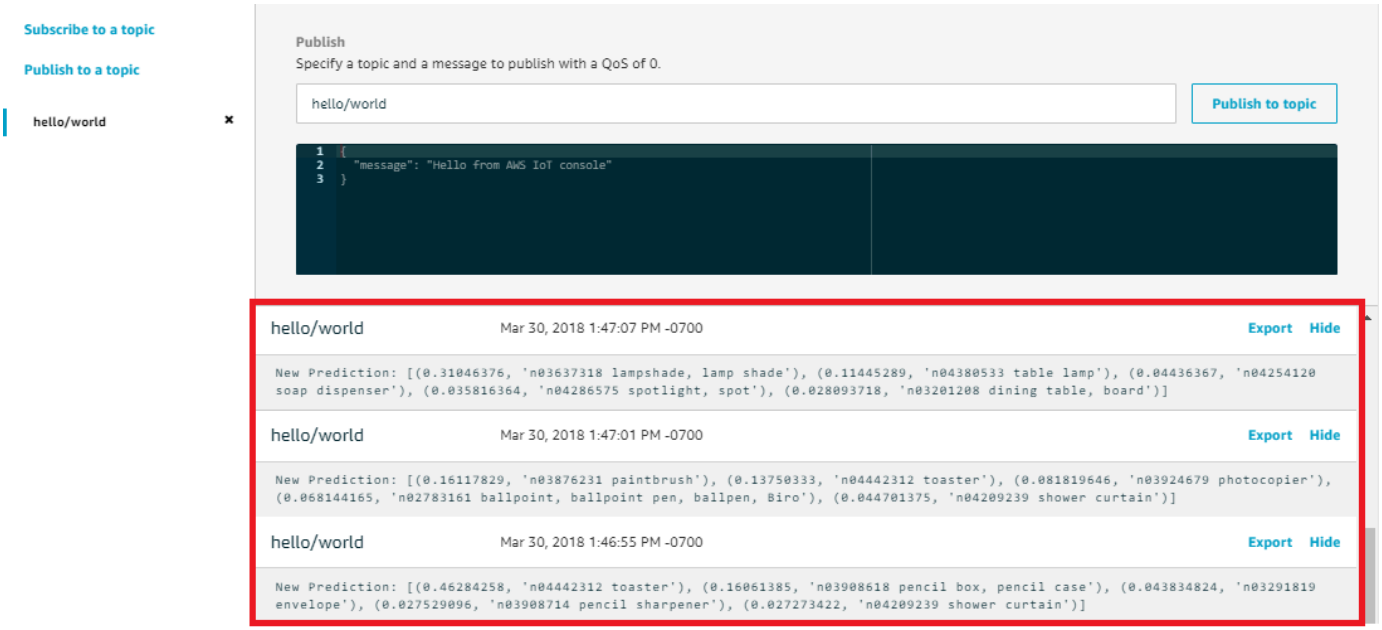
1. In derAWS IoT-Konsole, unterTest, wählenMTT-Test-Client.
2. Verwenden Sie für Subscriptions (Abonnements) die folgenden Werte:

   - Verwenden Sie als Abonnementthema „hello/world“.

- UNZusätzliche Konfiguration, fürMQTT-Nutzlast-Anzeige, wählenZeigt Nutzlasten als Zeichenfolgen an.

3. Wählen Sie Subscribe aus.

   Wenn der Test erfolgreich ist, werden die Nachrichten von 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.



Fehlerbehebung 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`. Der Zugriff auf AWS IoT Greengrass-Protokolle erfordert Root-Berechtigungen.

   ```
   sudo su
   cd /greengrass/ggc/var/log
   ```

2. 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”.

AuspackenFehlerruntime.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, umneu verpackenedas Modell:

```
tar -zcvf model.tar.gz ./model
```

Zum 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 wird

1. Listen Sie den Inhalt des bereitgestellten Lambdain/`lambda`-Verzeichnis 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: Erstellen und Veröffentlichen einer Lambda-Funktion 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 wird die PID in der zweiten Spalte der Zeile für den Lambda-Laufzeitprozess angezeigt.

2. Geben Sie den Lambda-Laufzeit-Namespace ein Stellen Sie sicher, den *`pid`*-Wert des Platzhalters zu ersetzen, bevor Sie den Befehl ausführen.

   > **Note**
   >
   > Dieses Verzeichnis und sein Inhalt befinden sich im Namespace der Lambda-Laufzeitumgebung, daher sind sie nicht in einem regulären Linux-Namespace sichtbar.

   ```
   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 Greengrassbietet weitere Lambda-Funktionen, mit denen Sie lokale Inferenz ausprobieren können. Sie finden das Beispielpaket im Ordner der vorkompilierten Bibliotheken, den Sie unter the section called “Installieren des MXNet-Frameworks” heruntergeladen haben.

Konfigurieren eines Intel Atom

Um dieses Tutorial auf einem Intel Atom-Gerät auszuführen, müssen Sie Quellabbilder bereitstellen, die Lambda-Funktion konfigurieren und eine weitere lokale Geräteressource 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 für die Bildklassifizierung herunter. 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 Bereitstellungspaket der Lambda-Funktion enthalten, das Sie in hochgeladen habenthe section called “Erstellen und Veröffentlichen einer Lambda-Funktion”.

   > **Note**
   >
   > Wenn Sie AWS DeepLens verwenden, können Sie die integrierte Kamera verwenden oder Ihre eigene Kamera mounten, um Inferenzen auf aufgenommenen Abbildern anstelle von statischen Abbildern durchzuführen. 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 unterAktualisieren derAWS DeepLensGerätimAWS DeepLensEntwicklerhandbuch.

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 so konfiguriert, dass es Python 3 mit AWS IoT Greengrass verwendet. 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 “Hinzufügen der Lambda-Funktion zur Gruppe”.

   > **Note**
   >
   > Wir empfehlen, dass Sie Ihre Lambda-Funktion ohne Containerisierung ausführen, es sei denn, Ihr Geschäftsfall erfordert dies. Dies ermöglicht den Zugriff auf die GPU

und die Kamera Ihres Geräts, ohne die Geräteressourcen zu konfigurieren. Wenn Sie ohne Containerisierung laufen, müssen Sie auch Root-Zugriff auf IhreAWS IoT GreengrassLambda-Funktionen

a. So laufen Sie ohne Containerisierung aus:

   - FürSystembenutzer und Gruppe, wählen**`Another user ID/group ID`**. FürSystembenutzer-IDden Wert ein.**`0`**. FürSystemgruppen-IDden Wert ein.**`0`**.

     Auf diese Weise kann Ihre Lambda-Funktion als root ausgeführt werden. Weitere Informationen zum Ausführen als root finden Sie unterthe section called “Festlegen der Standardzugriffsidentität für Lambda-Funktionen in einer Gruppe”.

     > **Tip**
     >
     > Sie müssen auch Ihre`config.json`-Datei, um Root-Zugriff auf Ihre Lambda-Funktion zu gewähren. Die -Prozedur finden Sie unterthe section called “Ausführen einer Lambda-Funktion als Root”.

   - FürContainerisierung der Lambda-Funktion, wählenKein Container.

     Weitere Informationen zur Ausführung ohne Containerisierung finden Sie unterthe section called “Überlegungen bei der Auswahl 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.
   - UNPinned, wählenWahr.
   - UNZusätzliche Parameter, fürLesezugriff auf das /sys-Verzeichnis, wählenEnabled.
   - Wählen Sie für Lambda-Lebenszyklus die Option Diese Funktion langlebig einstellen und unbegrenzt ausführen aus.

b. Um stattdessen im Containermodus zu laufen:

   > **Note**
   >
   > Wir empfehlen, nicht im Containermodus auszuführen, es sei denn, Ihr Geschäftsfall 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.
- FürPinned, wählenWahr.
- UNZusätzliche Parameter, fürLesezugriff auf das /sys-Verzeichnis, wählenEnabled.

4. Wenn es im Containermodus ausgeführt wirddie erforderliche lokale Geräteressource hinzu, um Zugriff auf die GPU Ihres Geräts zu gewähren.

> **Note**
>
> Wenn Sie im nicht containerisierten Modus laufen,AWS IoT Greengrasskann auf die GPU Ihres Geräts zugreifen, ohne Geräteressourcen zu konfigurieren

   a. Wählen Sie auf der Gruppenkonfigurationsseite die OptionRessourcender Wert ein.
   b. Klicken Sie aufRessource „Lokale Ressource“.
   c. Definieren Sie die Ressource:

      - Geben Sie für Resource name (Ressourcenname) **`renderD128`** ein.
      - FürResource type (Ressourcentyp), wählenLokales Gerät.
      - Geben Sie im Feld Device path (Gerätepfad) **`/dev/dri/renderD128`** ein.
      - FürEigentümer der Systemgruppe und Dateizugriffsberechtigungen, wählenAutomatisch Dateisystemberechtigungen der Systemgruppe hinzufügen, zu der die Ressource gehört.
      - FürZugehörigkeit zu Lambda-Funktion,Lese- und Schreibzugriffzu Ihrer Lambda-Funktion

Konfigurieren eines NVIDIA Jetson TX2

Um dieses Tutorial auf einem NVIDIA Jetson TX2 auszuführen, stellen Sie Quellabbilder bereit und konfigurieren Sie die Lambda-Funktion. Bei Verwendung der GPU müssen Sie zusätzlich lokale Geräteressourcen hinzufügen.

1. Stellen Sie sicher, dass Ihr Jetson-Gerät konfiguriert ist, damit 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 Installing MXNet on a Jetson und folgen Sie den Anweisungen zur Installation von MXNet auf dem Jetson-Gerät.

   > **Note**
   >
   > Wenn Sie MXNet aus der Quelle erstellen möchten, folgen Sie den Anweisungen zum Erstellen der freigegebenen Bibliothek. Bearbeiten Sie die folgenden Einstellungen in Ihrer `config.mk`-Datei für das Arbeiten mit einem Jetson TX2-Gerät:
   >
   > - 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 für die Bildklassifizierung herunter. 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 ist im Bereitstellungspaket der Lambda-Funktion enthalten, das Sie in hochgeladen habenthe section called “Erstellen und Veröffentlichen einer Lambda-Funktion”.

4. Erstellen Sie einen Symlink von Python 3.7 zu Python 3.6, um Python 3 mit AWS IoT Greengrass zu 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 “Hinzufügen der Lambda-Funktion zur Gruppe”.

> **Note**
>
> Wir empfehlen, dass Sie Ihre Lambda-Funktion ohne Containerisierung ausführen, es sei denn, Ihr Geschäftsfall erfordert dies. Dies ermöglicht den Zugriff auf die GPU und die Kamera Ihres Geräts, ohne die Geräteressourcen zu konfigurieren. Wenn Sie ohne Containerisierung laufen, müssen Sie auch Root-Zugriff auf IhreAWS IoT GreengrassLambda-Funktionen

a. So laufen Sie ohne Containerisierung aus:

- FürSystembenutzer und Gruppe, wählen**`Another user ID/group ID`**. FürSystembenutzer-IDden Wert ein.**`0`**. FürSystemgruppen-IDden Wert ein.**`0`**.

  Auf diese Weise kann Ihre Lambda-Funktion als root ausgeführt werden. Weitere Informationen zum Ausführen als root finden Sie unterthe section called “Festlegen der Standardzugriffsidentität für Lambda-Funktionen in einer Gruppe”.

  > **Tip**
  >
  > Sie müssen auch Ihre`config.json`-Datei, um Root-Zugriff auf Ihre Lambda-Funktion zu gewähren. Die -Prozedur finden Sie unterthe section called “Ausführen einer Lambda-Funktion als Root”.

- FürContainerisierung der Lambda-Funktion, wählenKein Container.

  Weitere Informationen zur Ausführung ohne Containerisierung finden Sie unterthe section called “Überlegungen bei der Auswahl der Containerisierung von Lambda-Funktionen”.

- UNZusätzliche Parameter, fürLesezugriff auf das /sys-Verzeichnis, wählenEnabled.
- UNUmgebungsvariablen, fügen Sie die folgenden Schlüssel-Wert-Paare zu Ihrer Lambda-Funktion hinzu. Hiermit wird AWS IoT Greengrass für die Verwendung des MXNet-Frameworks konfiguriert.

Schlüssel	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 Containermodus zu laufen:

> **Note**
>
> Wir empfehlen, nicht im Containermodus auszuführen, es sei denn, Ihr Geschäftsfall erfordert dies.

- Erhöhen Sie den Wert für Speicherlimit. Verwenden Sie 500 MB für die CPU oder mindestens 2000 MB für die GPU.
- UNZusätzliche Parameter, fürLesezugriff auf das /sys-Verzeichnis, wählenEnabled.
- UNUmgebungsvariablen, fügen Sie die folgenden Schlüssel-Wert-Paare zu Ihrer Lambda-Funktion hinzu. Hiermit wird AWS IoT Greengrass für die Verwendung des MXNet-Frameworks konfiguriert.

Schlüssel	Wert
PATH	/usr/local/cuda/bin:$PATH
MXNET_HOME	$HOME/mxnet/
PYTHONPATH	$MXNET_HOME/python:$PYTHONPATH
CUDA_HOME	/usr/local/cuda

Schlüssel	Wert
LD_LIBRARY_PATH	$LD_LIBRARY_PATH:${CUDA_HOME}/lib64

7. Wenn es im Containermodus ausgeführt wirddie 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 nicht containerisierten Modus laufen,AWS IoT Greengrasskann auf die GPU Ihres Geräts zugreifen, ohne Geräteressourcen zu konfigurieren

   Für jede Ressource:

   - Wählen Sie in Resource type (Ressourcentyp) die Option Device (Gerät) aus.
   - FürEigentümer der Systemgruppe und Dateizugriffsberechtigungen, wählenAutomatisch Dateisystemberechtigungen der Systemgruppe hinzufügen, zu der die Ressource gehört.

Name	Gerätepfad
nvhost-strg	/dev/nvhost-strg
nvhost-gpu	/dev/nvhost-gpu
nvhost-ctrl-gpu	/nvhost-ctrl-gpu
nvhost-dbg-gpu	/nvhost-dbg-gpu
nvhost-prof-gpu	/nvhost-prof-gpu
nvmap	/dev/nvmap
nvhost-vic	/dev/nvhost-vic

Name	Gerätepfad
tegra_dc_ctrl	/dev/tegra_dc_ctrl

8. Wenn es im Containermodus ausgeführt wirddie folgende lokale Volume-Ressource hinzu, um Zugriff auf die Kamera Ihres Geräts zu gewähren. Folgen Sie dem Verfahren unter the section called “Hinzufügen von Ressourcen zur Gruppe”.

   > **Note**
   >
   > Wenn Sie im nicht containerisierten Modus laufen,AWS IoT Greengrasskann auf die Kamera Ihres Geräts zugreifen, ohne Volume-Ressourcen zu konfigurieren.

   - Wählen Sie für Resource type (Ressourcentyp) die Option Volume aus.
   - FürEigentümer der Systemgruppe und Dateizugriffsberechtigungen, wählenAutomatisch Dateisystemberechtigungen der Systemgruppe hinzufügen, zu der die Ressource gehört.

Name	Quellpfad	Zielpfad
shm	/dev/shm	/dev/shm
tmp	/tmp	/tmp

So konfigurieren Sie optimierte Machine Learning-Inferenz mit der AWS Management Console

Um die Schritte in diesem Tutorial ausführen zu können, müssen SieAWS IoT GreengrassCore v1.10 oder höher.

Sie können das SageMaker Neo-Deep-Learning-Compiler zur Optimierung der Prognoseeffizienz nativer Inferenzmodelle für Machine Learning in Tensorflow, Apache MXNet, PyTorch, ONNX und XGBoost-Frameworks für eine kleinere Stellfläche und schnellere Leistung. Sie können dann das optimierte Modell herunterladen und die SageMaker Neo Deep Learning Runtime und stellen Sie sie in IhremAWS IoT GreengrassGeräte für schnellere Rückschlüsse.

In diesem Tutorial wird beschrieben, wie SieAWS Management Consoleum eine Greengrass-Gruppe für die Ausführung eines Lambda-Inferenzbeispiels auszuführen, das Kamerabilder lokal erkennt, ohne dass Daten in die Cloud gesendet werden. Das Inferenzbeispiel greift auf das Kameramodul auf einem Raspberry Pi zu. In diesem Tutorial laden Sie ein vorkonfiguriertes Modell herunter, das von Resnet-50 trainiert und im Neo-Deep-Learning-Compiler optimiert wird. Anschließend führen Sie mit dem Modell eine lokale Bildklassifikation auf Ihrem AWS IoT Greengrass-Gerät durch.

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. Hinzufügen der Lambda-Funktion zur Gruppe
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 mitAWS 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 ein 2,5A-Netzteilum 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 ein Raspberry Pi, aber AWS IoT Greengrass unterstützt andere Plattformen, wie z. B. 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. Weitere Informationen zum Konfigurieren des Geräts, damit Sie die AWS IoT Greengrass Core-Software installieren können, finden Sie unter the section called “Einrichten anderer Geräte”.
> Für Plattformen von Drittanbietern, dieAWS IoT Greengrassunterstützt nicht, Sie müssen Ihre Lambda-Funktion im nicht containerisierten Modus ausführen. Um im nicht containerisierten Modus auszuführen, müssen Sie Ihre Lambda-Funktion als Root-Benutzer ausführen. Weitere Informationen erhalten Sie unter the section called “Überlegungen bei der Auswahl der Containerisierung von Lambda-Funktionen” und the section called “Festlegen 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 Amazon-Programm SageMaker Neo Deep Learning Runtime

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 Folgendes sicher NumPy ist installiert.

     ```
     sudo -u ggc_user bash -c 'python3 -c "import numpy"'
     ```

Schritt 3: Erstellen Sie eine Inferenz-Lambda-Funktion

Erstellen Sie in diesem Schritt ein Bereitstellungspaket für Lambda-Funktionen 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 Beispielpaket 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 vonAWS IoT GreengrassKern-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 unterAWS IoT GreengrassKern-SDK für Pythonauf GitHub.

3. Komprimieren Sie den Inhalt des `examples`-Verzeichnisses in eine Datei namens `optimizedImageClassification.zip`. Dies ist das Bereitstellungspaket.

```
cd path-to-downloaded-sample/dlr-py3-armv7l/examples
zip -r optimizedImageClassification.zip .
```

Das Bereitstellungspaket enthält den Funktionscode und die Abhängigkeiten. Dazu gehört auch der Code, der die -Python-APIs der Neo-Deep-Learning-Laufzeit aufruft, um die Inferenz mit den Deep-Learning-Compiler

> **Note**
>
> Stellen Sie sicher, dass sich die `.py`-Dateien und -Abhängigkeiten im Stammverzeichnis befinden.

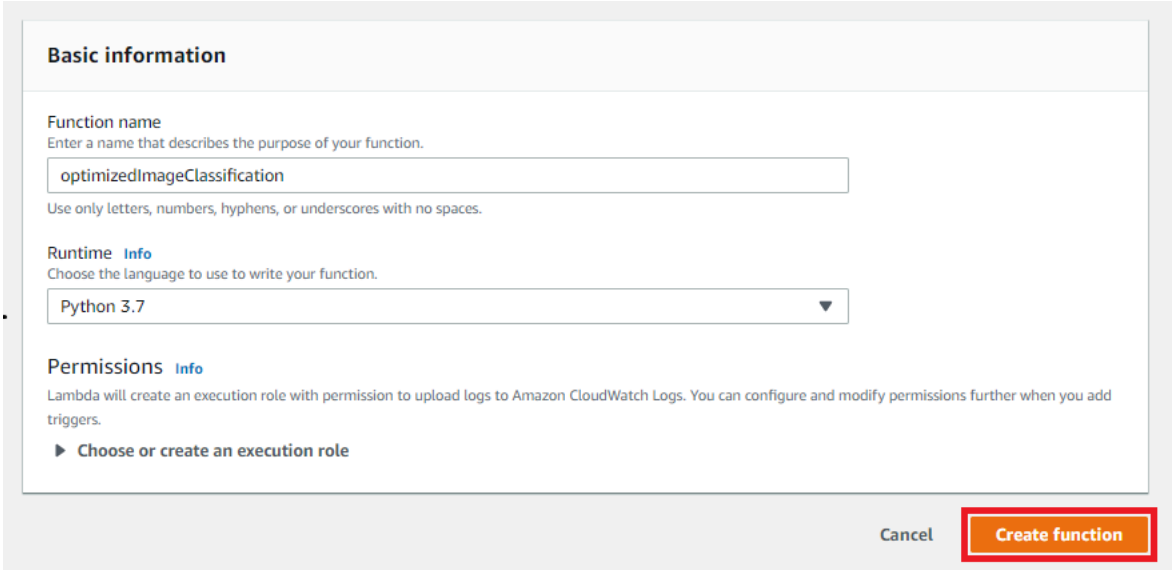
4. Fügen Sie nun die Lambda-Funktion zur Greengrass-Gruppe hinzu.

   Wählen Sie auf der Seite Lambda-KonsoleFunktionenund wähleFunktion erstellen.

5. Klicken Sie aufAuthor from scratchund verwenden Sie die folgenden Werte zum Erstellen der Funktion:

   - Geben Sie für Function name (Funktionsname) **`optimizedImageClassification`** ein.
   - Wählen Sie für Runtime (Laufzeit) die Option Python 3.7 aus.

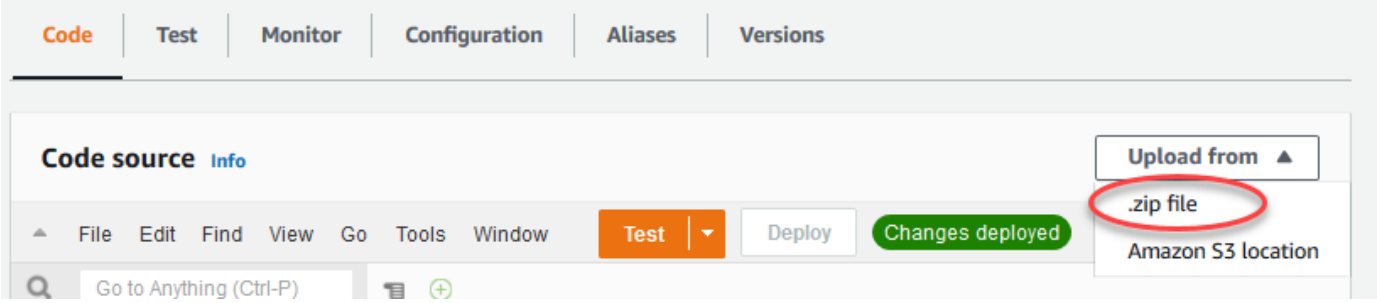
   FürBerechtigungenverwenden Sie die Standardeinstellung bei. Dadurch wird eine Ausführungsrolle erstellt, die grundlegende Lambda-Berechtigungen gewährt. Diese Rolle wird nicht verwendet vonAWS IoT Greengrass.



6. Wählen Sie Create function (Funktion erstellen).

Laden Sie nun das Bereitstellungspaket für Lambda-Funktionen hoch und registrieren Sie den Handler

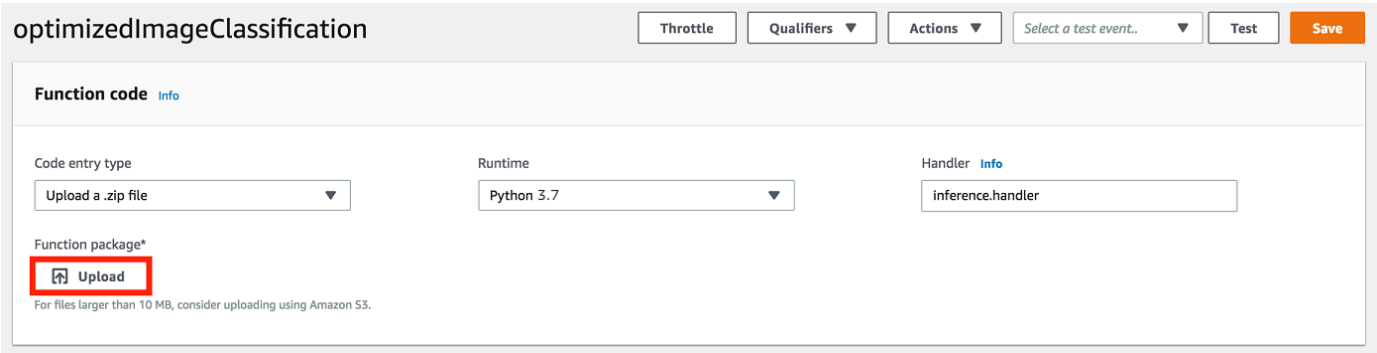
1. Auf derCodeunterQuellcode, wählenHochladen von. Wählen Sie aus der Dropdown-Liste die Option.Zip-Datei.



2. Wählen Sie Ihre`optimizedImageClassification.zip`Bereitstellungspaket, und wählen Sie dannSave.
3. Auf derCodefür die Funktion, unterRuntime-Einstellungen, wählenBearbeiten, 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.

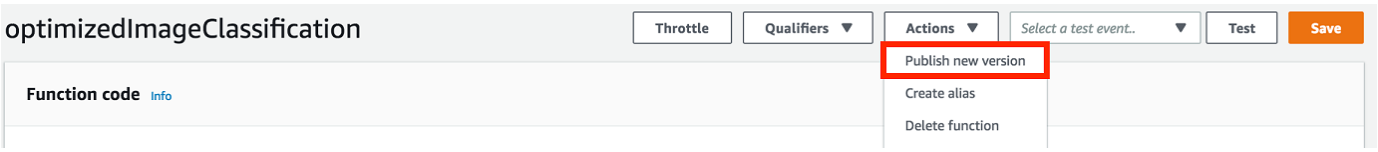
   Wählen Sie Save (Speichern) aus.



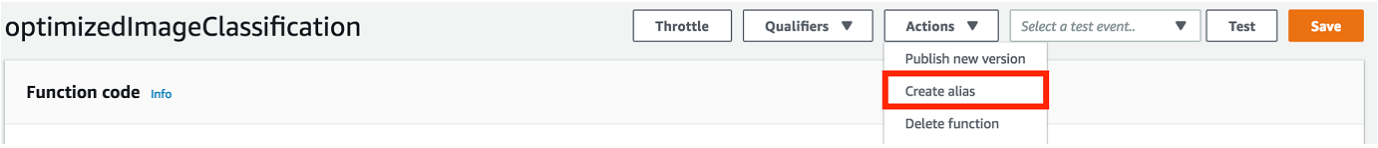
Als Nächstes veröffentlichen Sie die erste Version der Lambda-Funktion. Anschließend erstellen Sie einen Alias für die Version.

> **Note**
>
> Greengrass-Gruppen können eine Lambda-Funktion per Alias (empfohlen) oder nach Version referenzieren. Mit einem Alias lassen sich Code-Updates einfacher verwalten, da die Abonnementtabelle oder die Gruppendefinition nicht geändert werden muss, wenn der Funktionscode aktualisiert wird. Stattdessen verweisen Sie einfach den 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. Auf deroptimizedImageClassification: 1-Konfigurationsseite, von derAktionen-Menü, AuswahlErstellen eines Alias.



4. Geben Sie auf der Seite Create a new alias folgende Werte an:
   - Geben Sie unter Name **`mlTestOpt`** ein.
   - Geben Sie in Version (Version) **`1`** ein.

   > **Note**
   >
   > AWS IoT Greengrassunterstützt keine Lambda-Aliase für$LATESTVersionen.

5. Wählen Sie Create (Erstellen) aus.

Fügen Sie nun die Lambda-Funktion zur Greengrass-Gruppe hinzu.

Schritt 4: Hinzufügen der Lambda-Funktion zur Greengrass-Gruppe

In diesem Schritt fügen Sie die Lambda-Funktion der Gruppe hinzufügen und dann den Lebenszyklus konfigurieren.

Zunächst fügen Sie die Lambda-Funktion zur Greengrass-Gruppe hinzu.

1. In derAWS IoTNavigationsbereich der -Konsole unterVerwalten, vektorGreengrass-Geräteund wählen Sie und anschließend ausGruppen (V1).
2. Wählen Sie auf der Gruppenkonfigurationsseite die OptionLambda-Funktionenund wählen SieAdd.
3. Wählen Sie das SymbolLambda-Funktionund selectoptimizedImageClassification.
4. Auf derVersioning der Lambda-Funktion, wählen Sie den Alias für die Version, die Sie veröffentlicht haben.

Als Nächstes konfigurieren Sie den Lebenszyklus der Lambda-Funktion.

1. In derKonfiguration der Lambda-Funktiondie folgenden Änderungen vor.

   > **Note**
   >
   > Wir empfehlen, dass Sie Ihre Lambda-Funktion ohne Containerisierung ausführen, sofern es Ihr Business Case nicht erfordert. Dies ermöglicht den Zugriff auf die GPU und die Kamera Ihres Geräts, ohne die Geräteressourcen zu konfigurieren. Wenn Sie ohne Containerisierung laufen, müssen Sie auch Root-Zugriff auf IhreAWS IoT GreengrassLambda-Funktionen

   a. So laufen Sie ohne Containerisierung aus:

      - FürSystembenutzer und Gruppe, wählen**`Another user ID/group ID`**. FürSystembenutzer-IDden Wert ein.**`0`**. FürSystemgruppen-IDden Wert ein.**`0`**.

        Dadurch kann Ihre Lambda-Funktion als root ausgeführt werden. Weitere Informationen zur Ausführung als Root finden Sie unterthe section called “Festlegen der Standardzugriffsidentität für Lambda-Funktionen in einer Gruppe”.

> **Tip**
>
> Sie müssen auch Ihre`config.json`-Datei, um Root-Zugriff auf Ihre Lambda-Funktion zu gewähren. Die -Prozedur finden Sie unterthe section called “Ausführen einer Lambda-Funktion als Root”.

- FürContainerisierung der Lambda-Funktion, wählenKein Container.

  Weitere Informationen zum Ausführen ohne Containerisierung finden Sie unterthe section called “Überlegungen bei der Auswahl der Containerisierung von Lambda-Funktionen”.
- Geben Sie in Timeout (Zeitüberschreitung) **`10 seconds`** ein.
- FürPinned, wählenWahr.

  Weitere Informationen finden Sie unter the section called “Lebenszyklus-Konfiguration”.
- UNDERZusätzliche Parameter, fürLesezugriff auf das /sys-Verzeichnis, wählenEnabled.

b. Um stattdessen im Containermodus zu laufen:

> **Note**
>
> Wir empfehlen, nicht im Containermodus auszuführen, es sei denn, Ihr Geschäftsfall erfordert dies.

- FürSystembenutzer und Gruppe, wählenVerwenden Sie Gruppenstandard.
- FürContainerisierung der Lambda-Funktion, wählenVerwenden Sie Gruppenstandard.
- Geben Sie in Memory Limit (Speicherlimit) **`1024 MB`** ein.
- Geben Sie in Timeout (Zeitüberschreitung) **`10 seconds`** ein.
- FürPinned, wählenWahr.

  Weitere Informationen finden Sie unter the section called “Lebenszyklus-Konfiguration”.
- UNDERZusätzliche Parameter, fürLesezugriff auf das /sys-Verzeichnis, wählenEnabled.

2. Klicken Sie aufHinzufügen einer Lambda-Funktion.

Schritt 5: Hinzufügen einer SageMaker Neo-optimierte Modellressource für die Greengrass-Gruppe

Erstellen Sie in diesem Schritt eine Ressource für das optimierte ML-Inferenzmodell und laden Sie sie in einen Amazon S3 Bucket hoch. Suchen Sie dann das von Amazon S3 hochgeladene Modell in derAWS IoT Greengrass-Konsole und ordnen die neu erstellte Ressource mit der Lambda-Funktion zu. Dies ermöglicht es der Funktion, auf ihre Ressourcen auf dem Core-Gerät zuzugreifen.

1. Navigieren Sie auf Ihrem Computer zum `resnet50`-Verzeichnis im Beispielpaket, 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 Beispielpaket verwenden. Weitere Informationen finden Sie unter the section called “Konfigurieren 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. Auf der Gruppenkonfigurationsseite fürAWS IoT GreengrassGruppe aus und wählen Sie dieRessourcen-Registerkarte. 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. FürModel-Quelle, wählenVerwenden Sie ein in S3 gespeichertes Modell, z. B. ein mit Deep Learning Compiler optimiertes Modell.
5. UNDERVEKTOR, wählenDurchsuchen von S3.

> **Note**
>
> Derzeit optimiert SageMaker Modelle werden automatisch in Amazon S3 gespeichert. Sie können Ihr optimiertes Modell mit dieser Option in Ihrem Amazon S3 Bucket finden. Weitere Informationen zur Modelloptimierung in SageMaker, finden Sie unterSageMaker Neo Dokumentation.

6. Wählen Sie Upload a model (Ein Modell hochladen).
7. Laden Sie auf der Registerkarte der Amazon S3 -Konsole Ihre ZIP-Datei in einen Amazon S3 Bucket hoch. Weitere Informationen finden Sie unterWie lade ich Dateien und Ordner in einen S3-Bucket hoch?imAmazon 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. In derAWS IoT Greengrassauf der Registerkarte der -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. In :Zielpfadden Wert ein.**`/ml_model`**.

Dies ist das Ziel für das lokale Modell im Lambda-Laufzeit-Namespace. Wenn Sie die Gruppe bereitstellen, ruft AWS IoT Greengrass 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. Wenn es im Containermodus ausgeführt wird:

    a. UNDEREigentümer der Systemgruppe und Dateizugriffsberechtigungen, wählenGeben Sie die Systemgruppe und Berechtigungen an.

    b. Klicken Sie aufSchreibgeschützter Zugriffund wählen Sie dannRessource hinzufügen.

Schritt 6: Hinzufügen einer Kamerageräte-Ressource zur Greengrass-Gruppe

Erstellen Sie in diesem Schritt eine Ressource für das Kameramodul und ordnen Sie es der Lambda-Funktion zu. Dies ermöglicht es der Lambda-Funktion, auf die Ressource auf dem Core-Gerät zuzugreifen.

> **Note**
>
> Wenn Sie im nicht containerisierten Modus laufen,AWS IoT Greengrasskann auf die GPU und Kamera Ihres Geräts zugreifen, ohne diese Geräteressource zu konfigurieren.

1. Wählen Sie auf der Gruppenkonfigurationsseite die OptionRessourcen-Registerkarte.
2. Auf derLokale Ressourcen-Registerkarte aus.Hinzufügen einer lokalen Ressource.
3. Auf derHinzufügen einer lokalen Ressourcedie folgenden Werte:

   - Geben Sie für Resource name (Ressourcenname) **`videoCoreSharedMemory`** ein.
   - Wählen Sie in Resource type (Ressourcentyp) die Option Device (Gerät) aus.
   - FürLokaler Gerätepfadden Wert ein.**`/dev/vcsm`**.

     Der Gerätepfad ist der lokale absolute Pfad der Geräteressource. Dieser Pfad kann nur auf ein zeichenorientiertes Gerät oder Blockgerät unter `/dev` verweisen.

- FürEigentümer der Systemgruppe und Dateizugriffsberechtigungen, wählenAutomatisch Dateisystemberechtigungen der Systemgruppe hinzufügen, zu der die Ressource gehört.

  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 SeiteRessource hinzufügen.
5. FromRessourceneine weitere lokale Ressource erstellen, indem SieAddund verwenden Sie die folgenden Werte:

   - Geben Sie für Resource name (Ressourcenname) **`videoCoreInterface`** ein.
   - Wählen Sie in Resource type (Ressourcentyp) die Option Device (Gerät) aus.
   - FürLokaler Gerätepfadden Wert ein.**`/dev/vchiq`**.
   - FürEigentümer der Systemgruppe und Dateizugriffsberechtigungen, wählenAutomatisch Dateisystemberechtigungen der Systemgruppe hinzufügen, zu der die Ressource gehört.
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. Mit diesen Abonnements kann die Lambda-Funktion Prognoseergebnisse anAWS IoTindem Sie in einem MQTT-Thema.

1. Wählen Sie auf der Gruppenkonfigurationsseite die OptionAbonnementsund wählen Sie und wählen Sie dann ausAbo hinzufügen.
2. Auf derErstellen eines Abonnementsdie Quelle und das Ziel wie folgt:

   a. In :Ressourcentyp, wählenLambda-Funktionund wählen Sie und anschließend ausoptimizedImageClassification.

   b. In :Zieltyp, wählenServiceund wählen Sie und anschließend ausIoT Cloud.

   c. In derThemenfilterden Wert ein.**`/resnet-50/predictions`**und wählen Sie und anschließend ausErstellen eines Abonnements.
3. Hinzufügen eines zweiten Abonnements. Wählen Sie das SymbolAbonnements-Registerkarte aus.Abo hinzufügen, und konfigurieren Sie die Quelle und das Ziel wie folgt:

   a. In :Ressourcentyp, wählenServicesund wählen Sie und anschließend ausIoT Cloud.

b. In :Zieltyp, wählenLambda-Funktionund wählen Sie und anschließend ausoptimizedImageClassification.

c. In derThemenfilterden Wert ein.**`/resnet-50/test`**und wählen Sie und anschließend ausErstellen eines Abonnements.

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, Ressourcen und Abonnementkonfigurationen, die Sie hinzugefügt haben.

1. Stellen Sie sicher, dass dieAWS IoT Greengrasscore 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. So starten Sie den Daemon:

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

2. Wählen Sie auf der Gruppenkonfigurationsseite die OptionBereitstellen.
3. Auf derLambda-Funktionen, wählen SieIPdetektorund wähleBearbeiten.
4. FromEinstellungen für IP-Detektor bearbeitenund wählen Sie aus.Automatisches Erkennen und Überschreiben von MQTT-Broker-und wähleSave.

   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.

> **Note**
>
> Erteilen Sie bei Aufforderung die Berechtigung zum Erstellen desGreengrass-Servicerolleund assoziiere es mit deinemAWS-Kontoim aktuellenAWS-Region. Diese Rolle erlaubtAWS IoT Greengrassum auf Ihre -Ressourcen zuzugreifenAWS-Services.

Auf der Seite Deployments werden der Zeitstempel, die Versions-ID und der Status der Bereitstellung angegeben. Nach abgeschlossener Bereitstellung sollte der StatusCompleted (Abgeschlossen).

Weitere Informationen zu Bereitstellungen finden Sie unterBereitstellen von AWS IoT Greengrass-Gruppen. 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. Dies löst die Lambda-Funktion aus, um ein Foto mit Ihrem Raspberry Pi aufzunehmen und die Inferenz auf das aufgenommene Bild auszuführen.

> **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. Auf derAWS IoTKonsolen-Startseite, unterTest, wählenMQTT-Test-Client.
2. FürAbonnements, wählenAbonnieren eines Themas. Verwenden Sie die folgenden Werte. Übernehmen Sie für die verbleibenden Optionen die Standardwerte.

- Geben Sie für Abonnementthema **`/resnet-50/predictions`** ein.
- UNDERZusätzliche Konfiguration, fürMQTT-Nutzlast-Anzeige, wählenZeigt Nutzlasten als Zeichenfolgen an.

3. Wählen Sie Subscribe aus.
4. Klicken Sie aufVeröffentlichung für ein Themaden Wert ein.**`/resnet-50/test`**wie derTopic-Nameund wähleVeröffentlichen.
5. Wenn der Test erfolgreich ist, veranlasst die veröffentlichte Nachricht die Raspberry Pi-Kamera, ein Bild aufzunehmen. Eine Meldung der Lambda-Funktion erscheint unten auf der Seite. 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 Quellabbilder bereitstellen, die Lambda-Funktion konfigurieren und eine weitere lokale Geräteressource 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 die Bildklassifizierung durchzuführen. 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 Bereitstellungspaket für Lambda-Funktionen enthalten, das Sie in hochgeladen habenthe section called “Erstellen Sie eine Inferenz-Lambda-Funktion”.

   > **Note**
   >
   > Wenn Sie AWS DeepLens verwenden, können Sie die integrierte Kamera verwenden oder Ihre eigene Kamera mounten, um Inferenzen auf aufgenommenen Abbildern

anstelle von statischen Abbildern durchzuführen. 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 unterAktualisieren IhrerAWS DeepLensGerätimAWS DeepLensEntwicklerhandbuch.

2. Bearbeiten Sie die Konfiguration der Lambda-Funktion. Folgen Sie dem Verfahren unter the section called “Hinzufügen der Lambda-Funktion zur Gruppe”.

> **Note**
>
> Wir empfehlen, dass Sie Ihre Lambda-Funktion ohne Containerisierung ausführen, sofern es Ihr Business Case nicht erfordert. Dies ermöglicht den Zugriff auf die GPU und die Kamera Ihres Geräts, ohne die Geräteressourcen zu konfigurieren. Wenn Sie ohne Containerisierung laufen, müssen Sie auch Root-Zugriff auf IhreAWS IoT GreengrassLambda-Funktionen

   a. So laufen Sie ohne Containerisierung aus:

      - FürSystembenutzer und Gruppe, wählen**`Another user ID/group ID`**. FürSystembenutzer-IDden Wert ein.**`0`**. FürSystemgruppen-IDden Wert ein.**`0`**.

        Dadurch kann Ihre Lambda-Funktion als root ausgeführt werden. Weitere Informationen zur Ausführung als Root finden Sie unterthe section called “Festlegen der Standardzugriffsidentität für Lambda-Funktionen in einer Gruppe”.

        > **Tip**
        >
        > Sie müssen auch Ihre`config.json`-Datei, um Root-Zugriff auf Ihre Lambda-Funktion zu gewähren. Die -Prozedur finden Sie unterthe section called “Ausführen einer Lambda-Funktion als Root”.

      - FürContainerisierung der Lambda-Funktion, wählenKein Container.

        Weitere Informationen zum Ausführen ohne Containerisierung finden Sie unterthe section called “Überlegungen bei der Auswahl 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.
- FürPinned, wählenWahr.
- UNDERZusätzliche Parameter, fürLesezugriff auf das /sys-Verzeichnis, wählenEnabled.

b. Um stattdessen im Containermodus zu laufen:

> **Note**
>
> Wir empfehlen, nicht im Containermodus auszuführen, es sei denn, Ihr Geschäftsfall 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.
- FürPinned, wählenWahr.
- UNDERZusätzliche Parameter, fürLesezugriff auf das /sys-Verzeichnis, wählenEnabled.

3. Fügen Sie Ihrer Neo-optimierten Modellressource der Gruppe hinzu. Laden Sie die Modellressourcen in das `resnet50`-Verzeichnis des Beispielpakets 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 Containermodus ausgeführt wirddie erforderliche lokale Geräteressource hinzu, um Zugriff auf die GPU Ihres Geräts zu gewähren.

> **Note**
>
> Wenn Sie im nicht containerisierten Modus laufen,AWS IoT Greengrasskann auf die GPU Ihres Geräts zugreifen, ohne Geräteressourcen zu konfigurieren

a. Wählen Sie auf der Gruppenkonfigurationsseite die OptionRessourcen-Registerkarte.

b. In derLokale Ressourcenund wählen Sie aus.Hinzufügen einer lokalen Ressource.

c. Definieren Sie die Ressource:

- Geben Sie für Resource name (Ressourcenname) **`renderD128`** ein.
- Wählen Sie in Resource type (Ressourcentyp) die Option Device (Gerät) aus.
- FürLokaler Gerätepfadden Wert ein.**`/dev/dri/renderD128`**.
- FürEigentümer der Systemgruppe und Dateizugriffsberechtigungen, wählenAutomatisch Dateisystemberechtigungen der Systemgruppe hinzufügen, zu der die Ressource gehört.

Konfigurieren eines NVIDIA Jetson TX2

Um dieses Tutorial auf einem NVIDIA Jetson TX2 auszuführen, stellen Sie Quellabbilder bereit, konfigurieren Sie die Lambda-Funktion und fügen Sie weitere lokale Geräteressourcen hinzu.

1. Stellen Sie sicher, dass Ihr Jetson-Gerät konfiguriert ist, damit Sie die AWS IoT Greengrass Core-Software installieren und die GPU für Inferenzzwecke verwenden können. Weitere Informationen zur Konfiguration des Geräts finden Sie unter the section called “Einrichten anderer Geräte”. Um die GPU für Inferenzzwecke auf einem NVIDIA Jetson TX2 zu verwenden, 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 die Bildklassifizierung durchzuführen. 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 ist im Bereitstellungspaket für Lambda-Funktionen enthalten, das Sie in hochgeladen habenthe section called “Erstellen Sie eine Inferenz-Lambda-Funktion”.

> **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 “Hinzufügen der Lambda-Funktion zur Gruppe”.

> **Note**
>
> Wir empfehlen, dass Sie Ihre Lambda-Funktion ohne Containerisierung ausführen, sofern es Ihr Business Case nicht erfordert. Dies ermöglicht den Zugriff auf die GPU und die Kamera Ihres Geräts, ohne die Geräteressourcen zu konfigurieren. Wenn Sie ohne Containerisierung laufen, müssen Sie auch Root-Zugriff auf IhreAWS IoT GreengrassLambda-Funktionen

   a. So laufen Sie ohne Containerisierung aus:

      - FürRun as (Ausführen als), wählen**`Another user ID/group ID`**. FürUIDden Wert ein.**`0`**. FürGUIDden Wert ein.**`0`**.

        Dadurch kann Ihre Lambda-Funktion als root ausgeführt werden. Weitere Informationen zur Ausführung als Root finden Sie unterthe section called “Festlegen der Standardzugriffsidentität für Lambda-Funktionen in einer Gruppe”.

        > **Tip**
        >
        > Sie müssen auch Ihre`config.json`-Datei, um Root-Zugriff auf Ihre Lambda-Funktion zu gewähren. Die -Prozedur finden Sie unterthe section called “Ausführen einer Lambda-Funktion als Root”.

      - FürContainerisierung der Lambda-Funktion, wählenKein Container.

        Weitere Informationen zum Ausführen ohne Containerisierung finden Sie unterthe section called “Überlegungen bei der Auswahl 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.
- FürPinned, wählenWahr.
- UNDERZusätzliche Parameter, fürLesezugriff auf das /sys-Verzeichnis, wählenEnabled.

b. Um stattdessen im Containermodus zu laufen:

> **Note**
>
> Wir empfehlen, nicht im Containermodus auszuführen, es sei denn, Ihr Geschäftsfall 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.
- FürPinned, wählenWahr.
- UNDERZusätzliche Parameter, fürLesezugriff auf das /sys-Verzeichnis, wählenEnabled.

4. Fügen Sie Ihrer Neo-optimierten Modellressource der Gruppe hinzu. Laden Sie die Modellressourcen in das `resnet18`-Verzeichnis des Beispielpakets 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 es im Containermodus ausgeführt wirddie erforderlichen lokalen Geräteressourcen hinzu, um Zugriff auf die GPU Ihres Geräts zu gewähren.

> **Note**
>
> Wenn Sie im nicht containerisierten Modus laufen,AWS IoT Greengrasskann auf die GPU Ihres Geräts zugreifen, ohne Geräteressourcen zu konfigurieren

a. Wählen Sie auf der Gruppenkonfigurationsseite die OptionRessourcen-Registerkarte.

b. In derLokale Ressourcenund wählen Sie aus.Hinzufügen einer lokalen Ressource.

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.
- FürEigentümer der Systemgruppe und Dateizugriffsberechtigungen, wählenAutomatisch Dateisystemberechtigungen der Systemgruppe hinzufügen, zu der die Ressource gehört.

Name	Gerätepfad
nvhost-strg	/dev/nvhost-strg
nvhost-gpu	/dev/nvhost-gpu
nvhost-ctrl-gpu	/vvvektornvhost-ctrl-gpu
nvhost-dbg-gpu	/vvvektornvhost-dbg-gpu
nvhost-prof-gpu	/vvvektornvhost-prof-gpu
nvmap	/dev/nvmap
nvhost-vic	/dev/nvhost-vic
tegra_dc_ctrl	/dev/tegra_dc_ctrl

6. Wenn es im Containermodus ausgeführt wirddie folgende lokale Volume-Ressource hinzu, um Zugriff auf die Kamera Ihres Geräts zu gewähren. Folgen Sie dem Verfahren unter the section called “Hinzufügen einer Neo-optimierten Modellressource zur Gruppe”.

   > **Note**
   >
   > Wenn Sie im nicht containerisierten Modus laufen,AWS IoT Greengrasskann auf Ihre Gerätekamera zugreifen, ohne Geräteressourcen zu konfigurieren.

   - Wählen Sie für Resource type (Ressourcentyp) die Option Volume aus.
   - FürEigentümer der Systemgruppe und Dateizugriffsberechtigungen, wählenAutomatisch Dateisystemberechtigungen der Systemgruppe hinzufügen, zu der die Ressource gehört.

Name	Quellpfad	Zielpfad
shm	/dev/shm	/dev/shm
tmp	/tmp	/tmp

7. 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.

8. 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:

   - FürAbonnements, wählenAbonnieren eines Themas. 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.

Fehlerbehebung 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`. Der Zugriff auf AWS IoT Greengrass-Protokolle erfordert Root-Berechtigungen.

   ```
   sudo su
   cd /greengrass/ggc/var/log
   ```

2. Check`runtime.log`für Fehler.

   ```
   cat system/runtime.log | grep 'ERROR'
   ```

   Sie können auch im 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”.

Sicherstellen, dass die Lambda-Funktion erfolgreich bereitgestellt wird

1. Listen Sie den Inhalt des bereitgestellten Lambdas in`/lambda`-Verzeichnis. 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 wird die PID in der zweiten Spalte der Zeile für den Lambda-Laufzeitprozess angezeigt.

2. Geben Sie den Lambda-Laufzeit-Namespace ein Stellen Sie sicher, den *`pid`*-Wert des Platzhalters zu ersetzen, bevor Sie den Befehl ausführen.

   > **Note**
   >
   > Dieses Verzeichnis und sein Inhalt befinden sich im Lambda-Laufzeit-Namespace, daher sind sie nicht in einem regulären Linux-Namespace sichtbar.

   ```
   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 `ml_model` lautet, müssen Sie ihn hier ersetzen.

   ```
   cd /ml_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 für die Geräte, die für Ihre Lambda-Funktion erforderlich sind, Geräteressourcen erstellen.

Nächste Schritte

Betrachten Sie als Nächstes weitere optimierte Modelle. Weitere Informationen finden Sie in der SageMaker Neo-Dokumentation.

Verwalten von Daten-Streams auf dem AWS IoT Greengrass Core

AWS IoT GreengrassDer Stream-Manager macht es einfacher und zuverlässiger, hochvolumige IoT-Daten in dasAWS Cloudaus. Der Stream-Manager verarbeitet Daten-Streams lokal und exportiert diese inAWS Cloudautomatisch. Diese Funktion lässt sich in gängige Edge-Szenarien integrieren, z. B. Inferenz für maschinelles Lernen (ML), bei denen Daten lokal verarbeitet und analysiert werden, bevor sie in dasAWS Cloudoder lokale Lagerziele.

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 dasAWS Cloudaus.

Sie können automatische Exporte in dieAWS Cloudzur Lagerung oder Weiterverarbeitung und Analyse. Stream Manager unterstützt das Exportieren in FolgendesAWS CloudZiele.

- Kanäle inAWS IoT Analyticsaus.AWS IoT Analyticsermöglicht es Ihnen, erweiterte Analysen Ihrer Daten durchzuführen, um Geschäftsentscheidungen zu treffen und Modelle für maschinelles Lernen zu verbessern. Weitere Informationen finden Sie unterWas ist ?AWS IoT Analytics?imAWS IoT Analytics-Benutzerhandbuchaus.
- Streams in Kinesis Data Streams Kinesis Data Streams wird häufig verwendet, um Daten mit hohem Volumen zu aggregieren und in ein Data Warehouse oder einen Map-Reduce-Cluster zu laden. Weitere Informationen finden Sie unterWas ist Amazon Kinesis Data Streams?imEntwicklerhandbuch für Amazon Kinesisaus.
- Komponenteneigenschaften inAWS IoT SiteWiseaus.AWS IoT SiteWiseermöglicht es Ihnen, Daten von Industrieanlagen skalierbar zu sammeln, zu organisieren und zu analysieren.

Weitere Informationen finden Sie unterWas ist ?AWS IoT SiteWise?imAWS IoT SiteWise-Benutzerhandbuchaus.

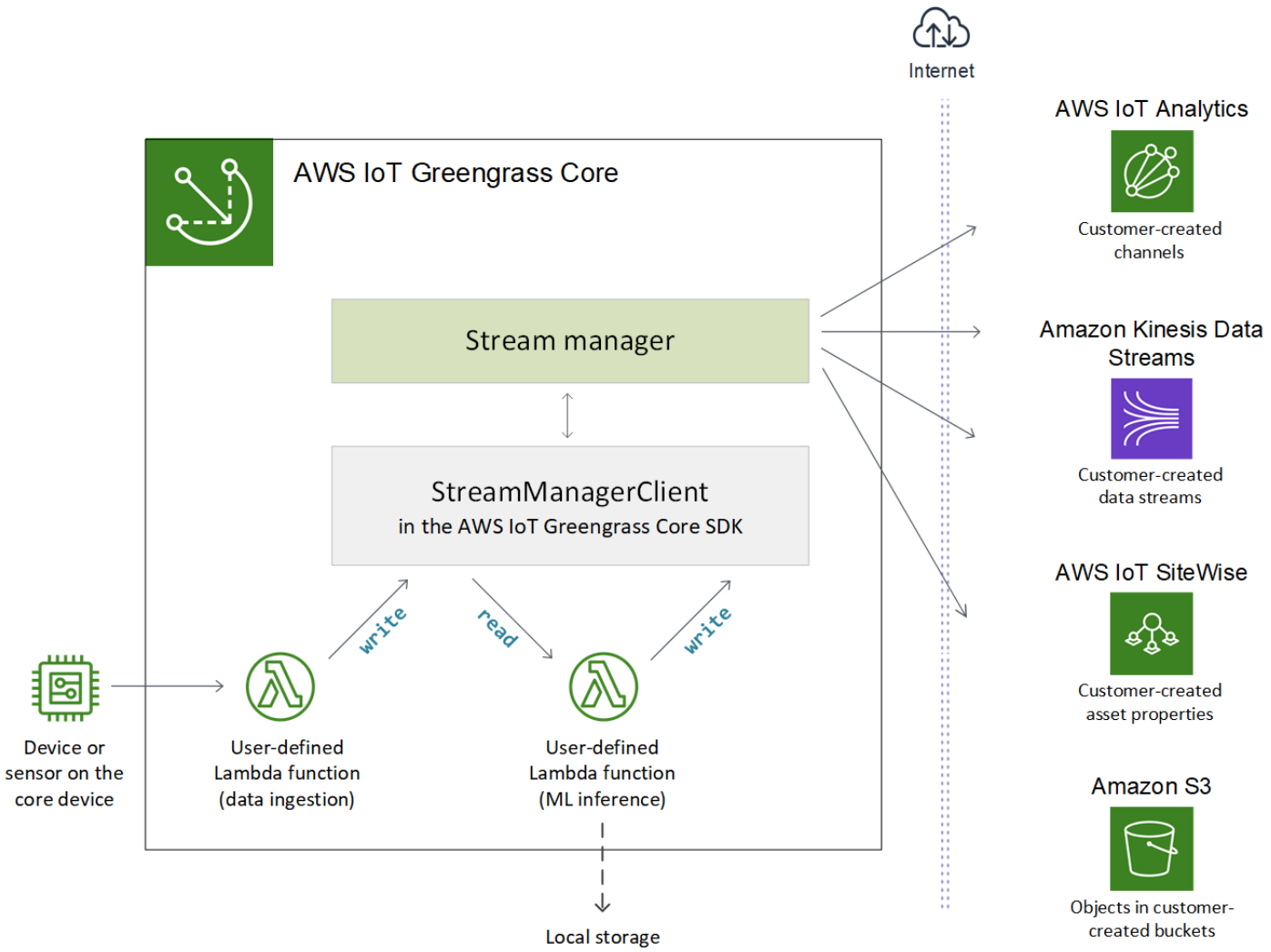
- Objekte in Amazon S3. Sie können Amazon S3 verwenden, um große Datenmengen zu speichern und abzurufen. Weitere Informationen finden Sie unterWas ist Amazon S3?imAmazon Simple Storage Service – Entwicklerhandbuchaus.

Stream-Management-Workflow

Ihre IoT-Anwendungen interagieren mit dem Stream-Manager über dasAWS IoT GreengrassCore-SDK In einem einfachen Workflow verbraucht eine benutzerdefinierte Lambda-Funktion, die auf dem Greengrass-Kern ausgeführt wird, IoT-Daten wie Temperatur- und Druckmetriken in Zeitreihen. Die Lambda-Funktion könnte die Daten filtern oder komprimieren und dann dasAWS IoT GreengrassCore-SDK, um die Daten in einen Stream im Stream-Manager zu schreiben. Der Stream-Manager kann den Stream in dieAWS Cloudautomatisch, basierend auf den für den Stream definierten Richtlinien. 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 darin schreiben. Diese lokalen Lambda-Funktionen können Streams lesen und darin 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 den Stream-Manager zu verwenden, konfigurieren Sie zunächst Stream-Manager-Parameter, um Laufzeiteinstellungen auf Gruppenebene zu definieren, die für alle Streams im Greengrass-Kern gelten. Diese anpassbaren Einstellungen ermöglichen es Ihnen, zu steuern, wie der Stream-Manager Streams basierend auf Ihren geschäftlichen Anforderungen und Umgebungsbeschränkungen speichert, verarbeitet und exportiert. 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 normalerweise benutzerdefinierte Lambda-Funktionen, die verwenden`StreamManagerClient`imAWS IoT GreengrassCore-SDK, um Streams zu erstellen und mit ihnen zu interagieren. Während der Stream-Erstellung definiert die Lambda-Funktion Richtlinien pro Stream, z. B. Exportzielbereiche, Priorität und Persistenz. Für weitere Informationen, einschließlich Code-Snippets für`StreamManagerClient`Operationen, siehethe section called “Verwenden von StreamManagerClient um mit Streams zu arbeiten”aus.

Informationen zu Tutorials, die einen einfachen Workflow konfigurieren, finden Sie unterthe section called “Exportieren von Daten-Streams (Konsole)”oderthe section called “Exportieren von Daten-Streams (CLI)”aus.

Voraussetzungen

Für die Verwendung des Stream-Managers gelten folgende Anforderungen:

- Sie müssen verwendenAWS IoT GreengrassCore-Software v1.10 oder höher, wobei der Stream-Manager aktiviert ist. Weitere Informationen finden Sie unter the section called “Konfigurieren des -Stream-Managers”.

  Der Stream-Manager wird auf nicht unterstützt OpenWrt -Verteilungen.
- 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.

- Der Stream-Manager benötigt zusätzlich zu Ihrer AWS IoT Greengrass Core-Basissoftware mindestens 70 MB RAM. Ihr gesamter Speicherbedarf hängt von Ihrer Arbeitslast ab.

- Benutzerdefinierte Lambda-Funktionen müssen dieAWS IoT GreengrassCore-SDKum mit dem Stream-Manager zu interagieren. DieAWS IoT GreengrassDas 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-Funktion -Laufzeit entspricht und fügen sie in das Bereitstellungspaket Ihrer Lambda-Funktion ein.

  > **Note**
  >
  > DieAWS IoT GreengrassCore-SDK für Python benötigt Python 3.7 oder höher und hat andere Paketabhängigkeiten. Weitere Informationen finden Sie unterErstellen eines Bereitstellungspakets für die Lambda-Funktion (Konsole)oderErstellen eines Bereitstellungspakets für die Lambda-Funktion (CLI)aus.

- Wenn du definierstAWS CloudExportziele für einen Stream, müssen Sie Ihre Exportziele erstellen und Zugriffsberechtigungen in der Greengrass-Gruppenrolle erteilen. 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 SiteWiseKomponenteneigenschaften”
  - the section called “Amazon S3 S3-Objekte”

  Sie sind verantwortlich für die Aufrechterhaltung dieserAWS CloudRessourcen schätzen.

Datensicherheit

Beachten Sie bei der Verwendung des Stream-Managers die folgenden Sicherheitsüberlegungen.

Lokale Datensicherheit

AWS IoT Greengrass verschlüsselt keine Streamdaten, die sich im Ruhezustand befinden oder lokal zwischen Komponenten auf dem Core-Gerät übertragen werden.

- Daten im Ruhezustand. Streamdaten werden lokal in einem Speicherverzeichnis im Greengrass-Kern gespeichert. Für die Datensicherheit verwendet AWS IoT Greengrass Unix-Dateiberechtigungen und Volldatenträgerverschlüsselung, 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, löscht AWS IoT Greengrass das vorherige Speicherverzeichnis oder dessen Inhalt nicht.

- Daten, die lokal übertragen werdenaus.AWS IoT Greengrassverschlüsselt Streamdaten bei der lokalen Übertragung auf dem Kern zwischen Datenquellen, Lambda-Funktionen,AWS IoT GreengrassCore-SDK und Stream-Manager.

- Daten, die in das übertragen werdenAWS Cloudaus. Vom Stream-Manager exportierte Datenströme in dieAWS Cloudverwenden StandardAWSVerschlüsselung des Dienstclients mit Transport Layer Security (TLS)

Weitere Informationen finden Sie unter the section called “Datenverschlüsselung”.

Client-Authentifizierung

Stream-Manager-Clients verwenden dasAWS IoT GreengrassCore-SDK für die Kommunikation mit dem Stream-Manager Wenn die Clientauthentifizierung 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 “Verwenden von StreamManagerClient um mit Streams zu arbeiten”
- the section called “Exportieren von Konfigurationen für unterstützteAWS CloudDestinationen”
- the section called “Exportieren von Daten-Streams (Konsole)”
- the section called “Exportieren von Daten-Streams (CLI)”

Konfigurieren des AWS IoT Greengrass-Stream-Managers

Auf derAWS IoT Greengrasscore, Stream-Manager kann 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 dasAWS IoT-KonsoleAWS IoT GreengrassAPI zum Konfigurieren der Stream-Manager-Einstellungen. 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. Diese IoT-Anwendungen sind in der Regel benutzerdefinierte Lambda-Funktionen. Weitere Informationen finden Sie unter the section called “Verwenden von 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”.

MinimumAWS IoT Greengrass-Co-Version: 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`.

MinimumAWS IoT Greengrass-Co-Version: 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 gesteuertAWS IoT GreengrassCo-SDK. Dieser Parameter bestimmt, welche Clients das aufrufen könnenAWS IoT Greengrass-SDK zum Arbeiten mit Streams. Weitere Informationen finden Sie unter the section called “Client-Authentifizierung”.

Gültige Werte sind `true` oder `false`. Der Standardwert ist `true` (empfohlen).

- `true`aus. Erlaubt nur Greengrass Lambda-Funktionen als Clients. Lambda-Funktionsclients verwenden internAWS IoT GreengrassKernprotokolle zur Authentifizierung mit demAWS IoT GreengrassCo-SDK.
- `false`aus. Ermöglicht jeden Prozess, der auf demAWS IoT Greengrasscore ein Kunde zu sein. Setzen Sie diese Einstellung nicht auf `false`, es sei denn, Ihr Geschäftsfall erfordert dies. Stellen Sie diesen Wert beispielsweise auf`false`nur wenn Nicht—Lambda-Prozesse auf dem

Core-Gerät direkt mit dem Stream-Manager kommunizieren müssen, z. B.Docker-Containerauf dem Core.

MinimumAWS IoT Greengrass-Co-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.

MinimumAWS IoT Greengrass-Co-Version: 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.

MinimumAWS IoT Greengrass-Co-Version: 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.

MinimumAWS IoT Greengrass-Co-Version: 1.10.0

Schreibgeschützte Dateiverzeichnisse

Parametername: `STREAM_MANAGER_READ_ONLY_DIRS`

Eine durch Komma getrennte Liste mit absoluten Pfaden zu den Verzeichnissen außerhalb des Root-Dateisystems, die Eingabedateien speichern. Der Stream-Manager liest und lädt die Dateien in Amazon S3 hoch und hängt die Verzeichnisse schreibgeschützt ein. Weitere Informationen zum Exportieren nach Amazon S3 finden Sie unter.the section called “Amazon S3 S3-Objekte”aus.

Verwenden Sie diesen Parameter nur, wenn die folgenden Bedingungen zutreffen:

- Das Eingabedateiverzeichnis für einen Stream, der nach Amazon S3 exportiert, befindet sich an einem der folgenden Speicherorte:
  - Eine andere Partition als das Root-Dateisystem.
  - UNDER`/tmp`im Root-Dateisystem.
- DieStandard-Containerisierungseigenschaftender Greengrass-GruppeGreengrass-Containeraus.

Beispielwert: `/mnt/directory-1,/mnt/directory-2,/tmp`

MinimumAWS IoT Greengrass-Co-Version: 1.11.0

Mindestgröße für mehrteilige Uploads

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. Der Stream-Manager verwendet diese Einstellung und die Größe der Eingabedatei, um zu bestimmen, wie Daten in einer mehrteiligen PUT-Anforderung gebündelt werden. Der Standard und Mindestwert ist`5242880`Byte (5 MB).

> **Note**
>
> Der Stream-Manager verwendet die`sizeThresholdForMultipartUploadBytes`-Eigenschaft, 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.

MinimumAWS IoT Greengrass-Co-Version: 1.11.0

Konfigurieren der Stream-Manager-Einstellungen (Konsole)

Sie können dasAWS IoT-Konsole für die folgenden Verwaltungsaufgaben:

- 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 unterthe section called “Exportieren von Daten-Streams (Konsole)”aus.

> **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, dass Sie die Speichergröße auf mindestens 128000 KB einstellen.

So prüfen Sie, ob der Stream-Manager aktiviert ist (Konsole)

1. In derAWS IoTNavigationsbereich der -Konsole unterVerwaltenErweitern SieGreengrass-GeräteKlicken Sie aufGruppen (V1)aus.
2. Wählen Sie die Zielgruppe aus.
3. Wählen Sie das SymbolLambdaaus.

4. UNDERSystem von Lambda-FunktionenSELECTStream-Manager, und wähleBearbeitenaus.
5. Überprüfen Sie den Status „Aktiviert“ oder „De Alle konfigurierten benutzerdefinierten Stream-Manager-Einstellungen werden ebenfalls angezeigt.

So aktivieren oder deaktivieren Sie den Stream-Manager während der Gruppenerstellung (Konsole)

1. In derAWS IoTNavigationsbereich der -Konsole unterVerwaltenErweitern SieGreengrass-GeräteKlicken Sie aufGruppen (V1)aus.
2. Wählen Sie Create Group. Ihre Auswahl auf der nächsten Seite bestimmt, wie Sie den Stream-Manager für die Gruppe konfigurieren.
3. Führen Sie dieNennen Sie Ihre Gruppeund wähle einGreengrass-Kernseiten.
4. Wählen Sie Create group (Gruppe erstellen) aus.
5. Wählen Sie auf der Gruppenkonfigurationsseite die Option aus.Lambda-Funktionen, wählen SieStream-Manager, und wähleBearbeitenaus.

   - Um den Stream-Manager mit Standardeinstellungen zu aktivieren, wählen SieMit Standardeinstellungen aktivierenaus.

   - Um den Stream-Manager mit benutzerdefinierten Einstellungen zu aktivieren, wählen Sie Einstellungen anpassen aus.
     1. Auf derKonfigurieren des Stream-Managers-SeiteAktivieren mit benutzerdefinierten Einstellungenaus.
     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, damit AWS IoT Greengrass die Standardwerte verwenden kann.

   - Um den Stream-Manager zu deaktivieren, wählen SieDeaktivieren vonaus.
     1. Wählen Sie auf der Seite Stream-Manager konfigurieren die Option Deaktivieren aus.

6. Wählen Sie Save (Speichern) aus.

7. Fahren Sie mit den verbleibenden Seiten fort, um Ihre Gruppe zu erstellen.
8. Auf derClient-Geräte, laden Sie Ihre Sicherheitsressourcen herunter, überprüfen Sie die Informationen und wählen Sie dannFinishaus.

> **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. In derAWS IoTNavigationsbereich der -Konsole unterVerwaltenErweitern SieGreengrass-GeräteKlicken Sie aufGruppen (V1)aus.
2. Wählen Sie die Zielgruppe aus.
3. Wählen Sie das SymbolLambdaaus.
4. UNDERSystem von Lambda-FunktionenSELECTStream-Manager, und wähleBearbeitenaus.
5. Überprüfen Sie den Status „Aktiviert“ oder „De Alle konfigurierten benutzerdefinierten Stream-Manager-Einstellungen werden ebenfalls angezeigt.

So ändern Sie die Einstellungen des Stream-Managers (Konsole)

1. In derAWS IoTNavigationsbereich der -Konsole unterVerwaltenErweitern SieGreengrass-GeräteKlicken Sie aufGruppen (V1)aus.
2. Wählen Sie die Zielgruppe aus.
3. Wählen Sie das SymbolLambdaaus.
4. UNDERSystem von Lambda-FunktionenSELECTStream-Manager, und wähleBearbeitenaus.
5. Überprüfen Sie den Status „Aktiviert“ oder „De Alle konfigurierten benutzerdefinierten Stream-Manager-Einstellungen werden ebenfalls angezeigt.
6. Wählen Sie Save (Speichern) aus.

Konfigurieren der Stream Manager-Einstellungen (CLI)

In derAWS CLI, benutze das System`GGStreamManager`Lambda—Funktion zum Konfigurieren des Stream-Managers. Lambda-Funktionen sind Komponenten desAWS IoT Greengrass-Co-Software-Sof Für den Stream-Manager und einige andere Lambda-Systemfunktionen können Sie die Greengrass-Funktionalität konfigurieren, indem Sie die entsprechenden`Function`und`FunctionDefinitionVersion`-Objekte in der Greengrass-Gruppe. 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 das verwendenAWS CLI, aber Sie können auch dasAWS IoT GreengrassAPI direkt oder verwenden Sie eineAWSSDK.

- Überprüfen, ob der Stream-Manager aktiviert ist
- Aktivieren, Deaktivieren oder Konfigurieren des Stream-Managers

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 unterthe section called “Exportieren von Daten-Streams (CLI)”aus.

> **Tip**
>
> Um zu sehen, ob der Stream-Manager aktiviert ist und von Ihrem Core-Gerät 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)

Der Stream-Manager ist aktiviert, wenn die bereitgestellte Funktionsdefinitionsversion das System enthält`GGStreamManager`Lambda-Funktion Zur Überprüfung gehen Sie folgendermaßen vor:

1. Rufen Sie die IDs der Greengrass-Zielgruppen und die Gruppenversion ab. Bei diesem Verfahren wird davon ausgegangen, dass es sich um die neueste Gruppe 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 inAWS IoTconsole. Die Gruppen-ID wird auf der Seite Einstellungen der Gruppe angezeigt. Gruppenversions-IDs werden in der Gruppe angezeigtBereitstellungenRegisterkarteneigenschaften.

2. Kopieren Sie die `LatestVersion`Werte und `Id` aus der Zielgruppe in die Ausgabe.
3. Rufen Sie die neueste Gruppenversion ab.

   - Ersetzen Sie *`group-id`* durch die kopierte `Id`.
   - Ersetzen*`latest-group-version-id`*mit dem`LatestVersion`das du kopiert hast.

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

4. Rufen Sie die IDs der Funktionsdefinition und der Funktionsdefinitionsversion aus dem `FunctionDefinitionVersionArn` in der Ausgabe ab.

   - Die ID der Funktionsdefinition ist die GUID nach dem`functions`-Segment im Amazon-Ressourcennamen (ARN).
   - 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*function-definition-id*mit der ID der Funktionsdefinition.
- Ersetzen*function-definition-version-id*mit der ID der Funktionsdefinitionsversion.

```
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.

So aktivieren, deaktivieren oder konfigurieren Sie den Stream-Manager (CLI)

In derAWS CLI, benutze das System`GGStreamManager`Lambda—Funktion zum Konfigurieren des Stream-Managers. 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**
>
> Für den`FunctionConfiguration`-Eigenschaften kennen Sie vielleicht Folgendes:
>
> - `MemorySize`ist mit Standardeinstellungen auf 4194304 KB (4 GB) eingestellt. Sie können diesen Wert jederzeit ändern. Wir empfehlen Ihnen, einzustellen`MemorySize`auf mindestens 128000 KB.
> - 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 den Stream-Manager mit benutzerdefinierten Werten für die Parameter Speicherverzeichnis, Serverport und Größe des Threadpools.

```
{
    "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 Greengrassverwendet Standardwerte fürStream-Manager-Parameterdie 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 Parameter der Mindestgröße für mehrteilige 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"
}
```

So aktivieren, deaktivieren oder konfigurieren Sie den Stream-Manager (CLI)

1. Rufen Sie die IDs der Greengrass-Zielgruppen und die Gruppenversion ab. Bei diesem Verfahren wird davon ausgegangen, dass es sich um die neueste Gruppe 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 inAWS IoTconsole. Die Gruppen-ID wird auf der Seite Einstellungen der Gruppe angezeigt. Gruppenversions-IDs werden in der Gruppe angezeigtBereitstellungenRegisterkarteneigenschaften.

2. Kopieren Sie die `LatestVersion`Werte und `Id` aus der Zielgruppe in die Ausgabe.
3. Rufen Sie die neueste Gruppenversion ab.

   - Ersetzen Sie *`group-id`* durch die kopierte `Id`.
   - Ersetzen*`latest-group-version-id`*mit dem`LatestVersion`das du kopiert hast.

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

4. Kopieren Sie das`CoreDefinitionVersionArn`und alle anderen Versions-ARNs aus der Ausgabe, außer`FunctionDefinitionVersionArn`aus. 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/bcfc6b49-beb0-4396-b703-6dEXAMPLEcu5/
versions/0f7337b4-922b-45c5-856f-1aEXAMPLEsf6
```

> **Note**
>
> Sie können auch eine Funktionsdefinition mit dem`create-function-definition`und kopieren Sie dann die ID aus der Ausgabe.

6. Fügen Sie der Funktionsdefinition eine Funktionsdefinitionsversion hinzu.

   - Ersetzen*`function-definition-id`*mit dem`Id`die Sie für die -Funktionsdefinition kopiert haben.

- In der`functions`-Array enthält alle anderen Funktionen, die Sie auf dem Greengrass-Core zur Verfügung stellen 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 das`GGStreamManager`-Funktion im`functions`Array. 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 Einstellungen des Stream-Managers sind optional, es sei denn, Sie möchten die Standardwerte ändern. Dieses Beispiel zeigt, wie Sie Umgebungsvariablen verwenden, 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**
>
> Für den`FunctionConfiguration`-Eigenschaften kennen Sie vielleicht Folgendes:
>
> - `MemorySize`ist mit Standardeinstellungen auf 4194304 KB (4 GB) eingestellt. Sie können diesen Wert jederzeit ändern. Wir empfehlen Ihnen, einzustellen`MemorySize`auf mindestens 128000 KB.
> - 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, in der die die Lambda-Systemfunktion enthalten ist.

   - Ersetzen Sie *`group-id`* durch die `Id` für die Gruppe.
   - Ersetzen*`core-definition-version-arn`*mit dem`CoreDefinitionVersionArn`die Sie aus der neuesten Gruppe kopiert haben.
   - Ersetzen*`function-definition-version-arn`*mit dem`Arn`die Sie für die neue Funktionsdefinitionsversion kopiert haben.
   - Ersetzen Sie die ARNs für andere Gruppenkomponenten (zum Beispiel `SubscriptionDefinitionVersionArn` oder `DeviceDefinitionVersionArn`), die Sie aus der neuesten Gruppe 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.

    - Ersetzen Sie *`group-id`* durch die kopierte `Id` für die Gruppe.
    - Ersetzen*`group-version-id`*mit dem`Version`die Sie für die neue Gruppe kopiert haben.

```
aws greengrass create-deployment \
--group-id group-id \
```

```
--group-version-id group-version-id \
--deployment-type NewDeployment
```

Verwenden Sie dieses Verfahren, wenn Sie die Einstellungen des Stream-Managers später erneut bearbeiten möchten. Stellen Sie sicher, dass Sie eine Funktionsdefinitionsversion mit dem`GGStreamManager`funktionieren mit der aktualisierten Konfiguration. Die Gruppenversion muss auf alle ARNs der Komponentenversion verweisen, die Sie auf dem Core bereitstellen möchten. Änderungen werden wirksam, nachdem die Gruppe bereitgestellt wurde.

Weitere Informationen finden Sie auch unter

- Verwalten von Daten-Streams
- the section called “Verwenden von StreamManagerClient um mit Streams zu arbeiten”
- the section called “Exportieren von Konfigurationen für unterstützteAWS CloudDestinationen”
- the section called “Exportieren von Daten-Streams (Konsole)”
- the section called “Exportieren von Daten-Streams (CLI)”

Verwenden von StreamManagerClient um mit Streams zu arbeiten

Benutzerdefinierte Lambda-Funktionen, die auf demAWS IoT Greengrasscore kann die`StreamManagerClient`-Objekt imAWS IoT GreengrassCore-SDKum Streams zu erstellenStream-Managerund interagiere dann mit den Streams. Wenn eine Lambda-Funktion einen Stream erstellt, definiert sie dieAWS CloudZiele, Priorisierung sowie 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**
>
> Typischerweise sind Clients des Stream-Managers benutzerdefinierte Lambda-Funktionen. Wenn Ihr Geschäftsfall dies erfordert, können Sie zulassen, dass nicht auf dem Greengrass Core ausgeführte -Prozesse (z. B. ein Docker-Container) mit dem Stream-Manager interagieren. Weitere Informationen finden Sie unter the section called “Client-Authentifizierung”.

Die Snippets in diesem Thema zeigen, wie Kunden anrufen`StreamManagerClient`Methoden zum Arbeiten mit Streams. Um Implementierungsdetails zu den Methoden und ihren Argumenten zu erhalten, verwenden Sie die Links zur SDK-Referenz, die nach jedem Snippet aufgeführt sind. Tutorials, die eine vollständige Python-Lambda-Funktion enthalten, finden Sie unterthe section called “Exportieren von Daten-Streams (Konsole)”oderthe section called “Exportieren von Daten-Streams (CLI)”aus.

Ihre Lambda-Funktion sollte instanziieren`StreamManagerClient`außerhalb des Funktionshandlers. 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 instanziieren, 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 “Aktualisieren eines Nachrichten-Streams”
- the section called “Löschen eines Nachrichten-Streams”

Erstellen eines Nachrichten-Streams

Um einen Stream zu erstellen, ruft eine benutzerdefinierte Lambda-Funktion die create-Methode auf und übergibt`MessageStreamDefinition`-Objekt. Dieses Objekt gibt den eindeutigen Namen für den Stream an und definiert, wie der Stream-Manager neue Daten verarbeiten soll, wenn die maximale Stream-Größe erreicht ist. Sie können mit `MessageStreamDefinition` seinen Datentypen (z. B. `ExportDefinition`, `StrategyOnFull`, und `Persistence`) andere Stream-Eigenschaften definieren. Dazu zählen:

- Das ZielAWS IoT Analytics, Kinesis Data Streams,AWS IoT SiteWiseund Amazon S3-Destinationen für automatische Exporte. Weitere Informationen finden Sie unter the section called “Exportieren von Konfigurationen für unterstützteAWS CloudDestinationen”.
- Export-Priorität. Stream-Manager exportiert Streams mit höherer Priorität vor Streams mit niedrigerer Priorität.
- Maximale Stapelgröße und Stapelintervall fürAWS IoT Analytics, Kinesis Data Streams undAWS IoT SiteWiseZielen. 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:

- MessageStreamDefinitionim Java SDK
- MessageStreamDefinitionim SDK Node.js
- MessageStreamDefinitionim 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 nicht stabil oder wird nicht für den Einsatz in Produktionsumgebungen unterstützt.

Nachdem ein Stream erstellt wurde, können Ihre Lambda-FunktionenAnhängen von Nachrichtenzum Stream um Daten für den Export zu senden undLesen von Nachrichtenaus dem Stream zur lokalen Verarbeitung. Die Anzahl der Streams, die Sie erstellen, hängt von Ihren Hardwarefunktionen und Ihrem Geschäftsfall ab. Eine Strategie besteht darin, einen Stream für jeden Zielkanal in zu

erstellenAWS IoT Analyticsoder Kinesis-Datenstream, obwohl Sie mehrere Ziele für einen Stream definieren können. Ein Stream hat eine dauerhafte Lebensdauer.

Voraussetzungen

Dieser Vorgang hat folgende Anforderungen:

- MinimumAWS IoT GreengrassCore-Version: 1.10.0
- MinimumAWS IoT GreengrassCore-SDK-Version: Python: 1.5.0 | Java: 1.4.0 | Node.js: 1.6.0

> **Note**
>
> Erstellen von Streams mit einemAWS IoT SiteWiseoder Amazon S3 S3-Exportziel hat die folgenden Anforderungen:
>
> - MinimumAWS IoT GreengrassCore-Version: 1.11.0
> - MinimumAWS IoT GreengrassCore-SDK-Version: 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 im`MessageStreamDefinition`und untergeordnete 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.
```

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:CreateMessageStream|MessageStreamDefinition

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        await client.createMessageStream(
            new MessageStreamDefinition()
                .withName("StreamName") // Required.
                .withMaxSize(268435456)  // Default is 256 MB.
                .withStreamSegmentSize(16777216)  // Default is 16 MB.
                .withTimeToLiveMillis(null)  // By default, no TTL is enabled.
                .withStrategyOnFull(StrategyOnFull.OverwriteOldestData)  //
 Required.
                .withPersistence(Persistence.File)  // Default is File.
                .withFlushOnWrite(false)  // Default is false.
                .withExportDefinition(  // Optional. Choose where/how the stream is
 exported to the AWS 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.
});
```

Node.js SDK-Referenz:CreateMessageStream|MessageStreamDefinition

Weitere Informationen zum Konfigurieren von Exportzielen finden Sie unterthe section called “Exportieren von Konfigurationen für unterstützteAWS CloudDestinationen”aus.

Anhängen einer Nachricht

Um Daten zum Export an den Stream-Manager zu senden, hängen Ihre Lambda-Funktionen die Daten an den Ziel-Stream an. Das Exportziel bestimmt den Datentyp, der an diese Methode übergeben werden soll.

Voraussetzungen

Dieser Vorgang hat die folgenden Anforderungen:

- MinimumAWS IoT GreengrassCore-Version: 1.10.0
- MinimumAWS IoT GreengrassCore-SDK-Version: Python: 1.5.0 | Java: 1.4.0 | Node.js: 1.6.0

> **Note**
>
> Nachrichten an ein anhängenAWS IoT SiteWiseoder Amazon S3 S3-Exportziel hat die folgenden Anforderungen:
>
> - MinimumAWS IoT GreengrassCore-Version: 1.11.0
> - MinimumAWS IoT GreengrassCore-SDK-Version: Python: 1.6.0 | Java: 1.5.0 | Node.js: 1.7.0

Beispiele

AWS IoT Analyticsoder Exportieren von Kinesis Data Streams

Das folgende Snippet fügt eine Nachricht an den Stream namens `StreamName` an. FürAWS IoT Analyticsoder Kinesis Data Streams Destinationen, Ihre Lambda-Funktionen hängen einen Datenblock an.

Dieser Snippet hat die folgenden Anforderungen:

- MinimumAWS IoT GreengrassCore-Version: 1.10.0
- MinimumAWS IoT GreengrassCore-SDK-Version: 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.
```

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 SiteWiseExportieren von -

Das folgende Snippet fügt eine Nachricht an den Stream namens `StreamName` an. FürAWS IoT SiteWiseDestinationen, Ihre Lambda-Funktionen hängen eine serialisierte`PutAssetPropertyValueEntry`-Objekt. Weitere Informationen finden Sie unter the section called “Exportieren in AWS IoT SiteWise”.

> **Note**
>
> Wenn Sie Daten an sendenAWS IoT SiteWisemüssen Ihre Daten die Anforderungen des`BatchPutAssetPropertyValue`Aktion Weitere Informationen finden Sie unter BatchPutAssetPropertyValue in der AWS IoT SiteWise-API-Referenz.

Dieser Snippet hat die folgenden Anforderungen:

- MinimumAWS IoT GreengrassCore-Version: 1.11.0
- MinimumAWS IoT GreengrassCore-SDK-Version: 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.
```

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 fügt eine Exportaufgabe an den Stream mit dem Namen an`StreamName`aus. Für Amazon S3 S3-Destinationen hängen Ihre Lambda-Funktionen eine serialisierte`S3ExportTaskDefinition`-Objekt, das Informationen zur Quelleingabedatei und zum Ziel-Amazon S3-Objekt 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”.

Dieser Snippet hat die folgenden Anforderungen:

- MinimumAWS IoT GreengrassCore-Version: 1.11.0
- MinimumAWS IoT GreengrassCore-SDK-Version: 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.
```

SDK-Referenz:append_message|s3ExportTaskDefinition

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    // Append an Amazon S3 export task definition and print the sequence number.
    S3ExportTaskDefinition s3ExportTaskDefinition = new S3ExportTaskDefinition()
        .withBucket("BucketName")
        .withKey("KeyName")
        .withInputUrl("URLToFile");
```

```
    long sequenceNumber = client.appendMessage("StreamName",
 ValidateAndSerialize.validateAndSerializeToJsonBytes(s3ExportTaskDefinition));
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java-SDK-Referenz:AppendMessage|s3ExportTaskDefinition

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
     // Append an Amazon S3 export task definition and print the sequence number.
     const taskDefinition = new S3ExportTaskDefinition()
        .withBucket("BucketName")
        .withKey("KeyName")
        .withInputUrl("URLToFile");
        const sequenceNumber = await client.appendMessage("StreamName",
 util.validateAndSerializeToJsonBytes(taskDefinition)));
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Node.js SDK-Referenz:AppendMessage|s3ExportTaskDefinition

Lesen von Nachrichten

Lesen Sie Nachrichten aus einem Stream.

Voraussetzungen

Dieser Vorgang hat die folgenden Anforderungen:

- MinimumAWS IoT GreengrassCore-Version: 1.10.0

- MinimumAWS IoT GreengrassCore-SDK-Version: 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_messsage_count to be
 fulfilled. By default, this is 0, which immediately returns the messages or an
 exception.
        )
    )
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

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_messsage_count to
 be fulfilled. By default, this is 0, which immediately returns the messages or an
 exception.
                    .withReadTimeoutMillis(Duration.ofSeconds(5L).toMillis())
    );
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

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

Rufen Sie die Liste der Streams im Stream-Manager ab.

Voraussetzungen

Dieser Vorgang hat die folgenden Anforderungen:

- MinimumAWS IoT GreengrassCore-Version: 1.10.0
- MinimumAWS IoT GreengrassCore-SDK-Version: 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.
```

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

Rufen Sie Metadaten zu einem Stream ab, einschließlich Definition, Größe und Exportstatus des Streams.

Voraussetzungen

Dieser Vorgang hat die folgenden Anforderungen:

- MinimumAWS IoT GreengrassCore-Version: 1.10.0
- MinimumAWS IoT GreengrassCore-SDK-Version: 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.
```

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.
```

```
});
```

Node.js SDK-Referenz:describeMessageStream

Aktualisieren eines Nachrichten-Streams

Aktualisieren Sie die Eigenschaften eines vorhandenen Streams. Möglicherweise möchten Sie einen Stream aktualisieren, wenn sich Ihre Anforderungen ändern, nachdem der Stream erstellt wurde. Zum Beispiel:

- Fügen Sie ein neues hinzuExportieren der Konfigurationfür einAWS CloudZiel.
- Erhöhen Sie die maximale Größe eines Streams, um zu ändern, wie Daten exportiert oder gespeichert werden. Beispielsweise kann die Stream-Größe in Kombination mit Ihrer Strategie für vollständige Einstellungen dazu führen, dass Daten gelöscht oder abgelehnt werden, bevor der Stream-Manager sie verarbeiten kann.
- Pausieren und fortsetzen Sie Exporte, z. B. wenn Exportaufgaben lange laufen und Sie Ihre Upload-Daten rationieren möchten.

Ihre Lambda-Funktionen folgen diesem hochrangigen Prozess, um einen Stream zu aktualisieren:

1. Holen Sie sich die Beschreibung des Streams.
2. Aktualisieren Sie die Zieleigenschaften auf dem entsprechenden`MessageStreamDefinition`und untergeordnete Objekte.
3. Übergeben Sie das aktualisierte`MessageStreamDefinition`aus. 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

Dieser Vorgang hat die folgenden Anforderungen:

- MinimumAWS IoT GreengrassCore-Version: 1.11.0

- MinimumAWS IoT GreengrassCore-SDK-Version: Python: 1.6.0 | Java: 1.5.0 | Node.js: 1.7.0

Beispiele

Das folgende Snippet aktualisiert den Stream mit dem Namen`StreamName`aus. 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.
```

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.
});
```

Node.js SDK-Referenz:updateMessageStream|MessageStreamDefinition

Einschränkungen für das Aktualisieren von Streams

Beim Aktualisieren von Streams gelten die folgenden Einschränkungen. Sofern in der folgenden Liste nicht angegeben, werden Aktualisierungen sofort wirksam.

- Sie können die Persistenz eines Streams nicht aktualisieren. Um dieses Verhalten zu ändern,Löschen des StreamsundErstellen eines Streamsdas definiert die neue Persistenzrichtlinie.
- Sie können die maximale Größe eines Streams nur unter den folgenden Bedingungen aktualisieren:
  - Die maximale Größe muss größer oder gleich der aktuellen Stream-Größe sein. Um diese Informationen zu finden,Beschreiben des Streamsund überprüfen Sie dann den Speicherstatus der zurückgegebenen`MessageStreamInfo`-Objekt.
  - Die maximale Größe muss größer oder gleich der Segmentgröße des Streams sein.
- Sie können die Stream-Segmentgröße auf einen Wert aktualisieren, der kleiner als die maximale Größe des Streams ist. Die aktualisierte Einstellung gilt für neue Segmente.

- Aktualisierungen der Time to Live (TTL) -Eigenschaft gelten für neue Anfügevorgänge. Wenn Sie diesen Wert verringern, löscht der Stream-Manager möglicherweise auch vorhandene Segmente, die die TTL überschreiten.
- Aktualisierungen der Strategie für vollständige Eigenschaft gelten für neue Anfügevorgänge. Wenn Sie die Strategie zum Überschreiben der ältesten Daten festlegen, überschreibt der Stream-Manager möglicherweise auch vorhandene Segmente basierend auf der neuen Einstellung.
- Aktualisierungen der Eigenschaft „Flush on Write“ gelten für neue Nachrichten.
- Aktualisierungen für 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 einen eindeutigen Bezeichner für die neue Exportkonfiguration an.
  - Um eine Exportkonfiguration zu löschen, lassen Sie die Exportkonfiguration weg.
- BisaktualisierenDie Startsequenznummer einer Exportkonfiguration in einem Stream müssen Sie einen Wert angeben, der kleiner als die letzte Sequenznummer ist. Um diese Informationen zu finden,Beschreiben des Streamsund überprüfen Sie dann den Speicherstatus der zurückgegebenen`MessageStreamInfo`-Objekt.

Löschen eines Nachrichten-Streams

Löscht einen Stream. Wenn Sie einen Stream löschen, werden alle gespeicherten Daten für den Stream von der Festplatte gelöscht.

Voraussetzungen

Dieser Vorgang hat folgende Anforderungen:

- MinimumAWS IoT GreengrassCore-Version: 1.10.0
- MinimumAWS IoT GreengrassCore-SDK-Version: 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.
```

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.
```

```
});
```

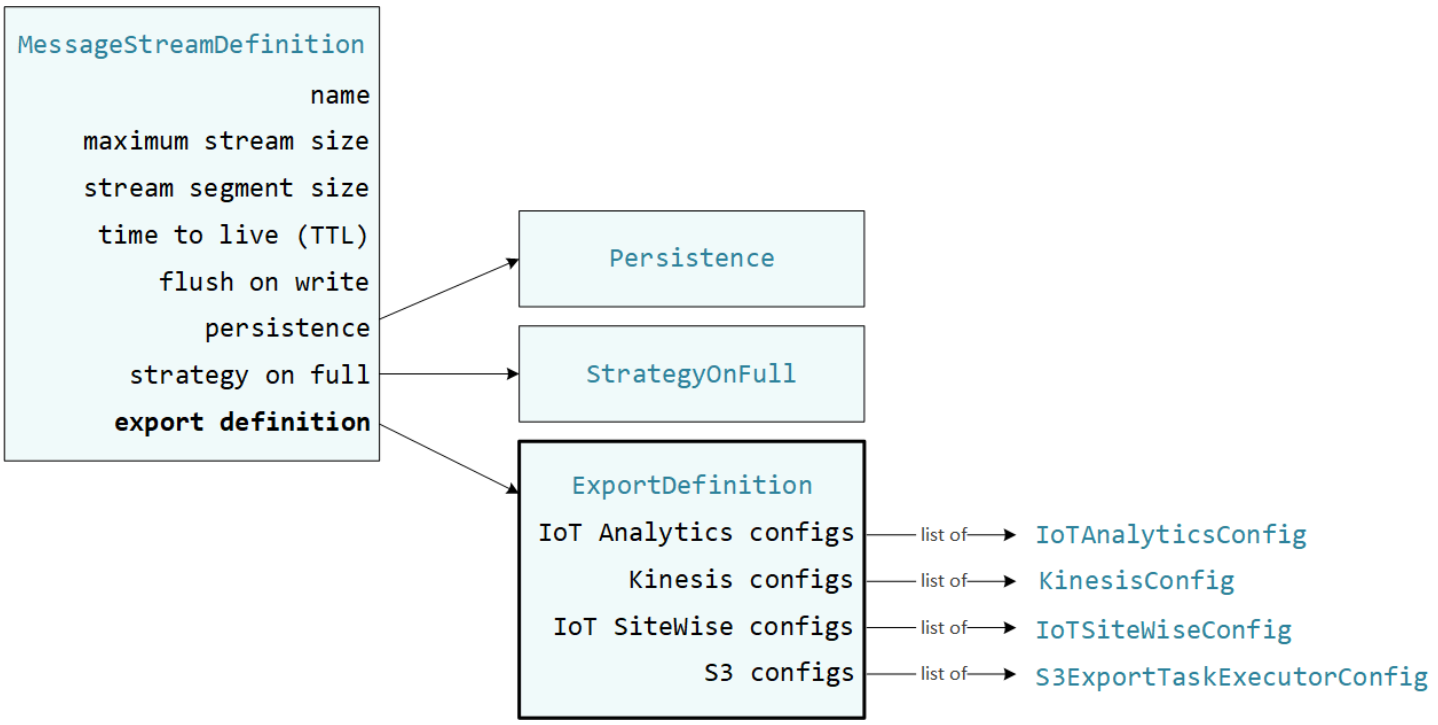
Node.js SDK-Referenz:deleteMessageStream

Weitere Informationen finden Sie auch unter

- Verwalten von Daten-Streams
- the section called “Konfigurieren des -Stream-Managers”
- the section called “Exportieren von Konfigurationen für unterstützteAWS CloudDestinationen”
- the section called “Exportieren von Daten-Streams (Konsole)”
- the section called “Exportieren von Daten-Streams (CLI)”
- `StreamManagerClient`imAWS IoT GreengrassSDK-Referenz:
  - Python
  - Java
  - Node.js

Exportieren von Konfigurationen für unterstützteAWS CloudDestinationen

Benutzerdefinierte Lambda-Funktionen verwenden`StreamManagerClient`imAWS IoT GreengrassCore-SDK zur Interaktion mit dem Stream-Manager. Wenn eine Lambda-Funktionerstellt einen Streamoderaktualisiert einen Stream, es passiert ein`MessageStreamDefinition`-Objekt, das Stream-Eigenschaften darstellt, einschließlich der Exportdefinition. Die`ExportDefinition`object enthält die Exportkonfigurationen, die für den Stream definiert sind. Stream Manager verwendet diese Exportkonfigurationen, um zu bestimmen, wo und wie der Stream exportiert werden soll.



Sie können null oder mehr Exportkonfigurationen in einem Stream definieren, einschließlich mehrerer Exportkonfigurationen für einen einzelnen Zieltyp. Sie können beispielsweise einen Stream in zwei exportierenAWS IoT Analytics-Kanäle und ein Kinesis-Data-Stream.

Bei fehlgeschlagenen Exportversuchen versucht der Stream-Manager kontinuierlich, Daten in dieAWS Cloudin Intervallen von bis zu fünf Minuten. Die Anzahl der Wiederholungsversuche hat kein maximales Limit.

> **Note**
>
> `StreamManagerClient`Bietet auch ein Zielziel, mit dem Sie Streams auf einen HTTP-Server exportieren können. Dieses Ziel dient nur zu Testzwecken. Es ist nicht stabil oder wird nicht für die Verwendung in Produktionsumgebungen unterstützt.

UnterstützteAWS CloudDestinationen

- AWS IoT Analytics-Kanäle
- Amazon Kinesis Kinesis-Datenströme
- AWS IoT SiteWiseKomponenteneigenschaften
- Amazon S3 S3-Objekte

Sie sind dafür verantwortlich, diese beizubehaltenAWS CloudRessourcen schätzen.

AWS IoT Analytics-Kanäle

Stream Manager unterstützt automatische Exporte nachAWS IoT Analyticsaus.AWS IoT Analyticsermöglicht es Ihnen, erweiterte Analysen Ihrer Daten durchzuführen, um Geschäftsentscheidungen zu treffen und Modelle für maschinelles Lernen zu verbessern. Weitere Informationen finden Sie unterWas ist ?AWS IoT Analytics?imAWS IoT Analytics-Benutzerhandbuchaus.

In derAWS IoT GreengrassCore SDK, Ihre Lambda-Funktionen nutzen das`IoTAnalyticsConfig`um die Exportkonfiguration für diesen Zieltyp zu definieren. Weitere Informationen finden Sie in der SDK-Referenz für Ihre Zielsprache:

- IotAnalyticsConfigim Python SDK
- IotAnalyticsConfigim Java SDK
- IotAnalyticsConfigim SDK Node.js

Voraussetzungen

Dieses Exportziel hat die folgenden Anforderungen:

- Zielkanäle inAWS IoT Analyticsin derselbenAWS-KontoundAWS-Regionals Greengrass-Gruppe.
- Diethe section called “Greengrass-Gruppenrolle.”muss das erlauben`iotanalytics:BatchPutMessage`Berechtigung zum Zielen von Kanälen. Zum Beispiel:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iotanalytics:BatchPutMessage"
            ],
            "Resource": [
                "arn:aws:iotanalytics:region:account-id:channel/channel_1_name",
                "arn:aws:iotanalytics:region:account-id:channel/channel_2_name"
            ]
        }
```

```
    ]
}
```

Sie können granularen oder bedingten Zugriff gewähren, etwa mit einem Platzhaltern*Benennungsschema. Weitere Informationen finden Sie unterHinzufügen und Entfernen von IAM-RichtlinienimIAM User Guideaus.

Exportieren in AWS IoT Analytics

So erstellen Sie einen Stream, der nachAWS IoT Analytics, Ihre Lambda-FunktionenErstellen eines -Streamsmit einer Exportdefinition, die eine oder mehrere enthält`IoTAnalyticsConfig`-Objekte. Dieses Objekt definiert Exporteinstellungen wie den Zielkanal, die Stapelgröße, das Stapelintervall und die Priorität.

Wenn Ihre Lambda-Funktionen Daten von Geräten empfangen, werden sieAnhängen von Nachrichtendie einen Datenblob zum Zielstream enthalten.

Dann exportiert der Stream Manager die Daten basierend auf den Batch-Einstellungen und der Priorität, die in den Exportkonfigurationen des Streams definiert sind.

Amazon Kinesis Kinesis-Datenströme

Der Stream-Manager unterstützt automatische Exporte zu Amazon Kinesis Data Streams. Kinesis Data Streams wird häufig verwendet, um Daten mit hohem Volumen zu aggregieren und in ein Data Warehouse oder einen Map-Reduce-Cluster zu laden. Weitere Informationen finden Sie unterWas ist Amazon Kinesis Data Streams?imAmazon Kinesis Developer Guideaus.

In derAWS IoT GreengrassCore SDK, Ihre Lambda-Funktionen nutzen das`KinesisConfig`um die Exportkonfiguration für diesen Zieltyp zu definieren. Weitere Informationen finden Sie in der SDK-Referenz für Ihre Zielsprache:

- KinesisConfigim Python SDK
- KinesisConfigim Java SDK
- KinesisConfigim SDK Node.js

Voraussetzungen

Dieses Exportziel hat die folgenden Anforderungen:

- Zielstreams in Kinesis Data Streams müssen im selben seinAWS-KontoundAWS-Regionals Greengrass-Gruppe.
- Diethe section called “Greengrass-Gruppenrolle.”muss das erlaubenkinesis:PutRecordsBerechtigung zum Zielen von Datenströmen. Zum Beispiel:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecords"
            ],
            "Resource": [
                "arn:aws:kinesis:region:account-id:stream/stream_1_name",
                "arn:aws:kinesis:region:account-id:stream/stream_2_name"
            ]
        }
    ]
}
```

  Sie können granularen oder bedingten Zugriff auf Ressourcen gewähren, etwa mit einem Platzhaltern*Benennungsschema. Weitere Informationen finden Sie unterHinzufügen und Entfernen von IAM-RichtlinienimIAM User Guideaus.

Exportieren in Kinesis Data Streams

Um einen Stream zu erstellen, der in Kinesis Data Streams exportiert, funktioniert Ihr Lambda-FunktionenErstellen eines -Streamsmit einer Exportdefinition, die eine oder mehrere enthältKinesisConfig-Objekte. Dieses Objekt definiert Exporteinstellungen wie den Zieldatenstrom, die Stapelgröße, das Stapelintervall und die Priorität.

Wenn Ihre Lambda-Funktionen Daten von Geräten empfangen, werden sieAnhängen von Nachrichtendie einen Datenblob zum Zielstream enthalten. Dann exportiert der Stream Manager die Daten basierend auf den Batch-Einstellungen und der Priorität, die in den Exportkonfigurationen des Streams definiert sind.

Der Stream-Manager generiert eine eindeutige, zufällige UUID als Partitionsschlüssel für jeden auf Amazon Kinesis hochgeladenen Datensatz.

AWS IoT SiteWiseKomponenteneigenschaften

Stream Manager unterstützt automatische Exporte nachAWS IoT SiteWiseaus.AWS IoT SiteWiseMit können Sie Daten von Industrieanlagen skalierbar sammeln, organisieren und analysieren. Weitere Informationen finden Sie unterWas ist ?AWS IoT SiteWise?imAWS IoT SiteWise-Benutzerhandbuchaus.

In derAWS IoT GreengrassCore SDK, Ihre Lambda-Funktionen nutzen das`IoTSiteWiseConfig`um die Exportkonfiguration für diesen Zieltyp zu definieren. Weitere Informationen finden Sie in der SDK-Referenz für Ihre Zielsprache:

- iotsiteWiseConfigim Python SDK
- iotsiteWiseConfigim Java SDK
- iotsiteWiseConfigim SDK Node.js

> **Note**
>
> AWSbietet auchthe section called “IoT SiteWise”, eine vorgefertigte Lösung, die Sie mit OPC-UA-Quellen verwenden können.

Voraussetzungen

Dieses Exportziel hat die folgenden Anforderungen:

- Zielobjekte inAWS IoT SiteWisein derselbenAWS-KontoundAWS-Regionals Greengrass-Gruppe.

  > **Note**
  >
  > Eine Liste der Regionen, dieAWS IoT SiteWiseunterstützt, sieheAWS IoT SiteWise-Endpunkte und -KontingenteimAWS– Allgemeine Referenzaus.

- Diethe section called “Greengrass-Gruppenrolle.”muss das erlaube`iotsitewise:BatchPutAssetPropertyValue`Berechtigung

zur Ausrichtung von Asset-Eigenschaften. Die folgende Beispielrichtlinie verwendet`iotsitewise:assetHierarchyPath`Bedingungsschlüssel zum Gewähren von Zugriff auf ein Ziel-Root-Asset und seine untergeordneten Elemente Sie können das`Condition`Von der Richtlinie aus, um Zugriff auf alle Ihre zu ermöglichen AWS IoT SiteWiseVermögenswerte oder geben Sie ARNs einzelner Vermögenswerte an.

```
{
    "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 granularen oder bedingten Zugriff gewähren, etwa mit einem Platzhaltern*Benennungsschema. Weitere Informationen finden Sie unterHinzufügen und Entfernen von IAM-RichtlinienimIAM User Guideaus.

Wichtige Sicherheitsinformationen finden Sie unter BatchPutAssetPropertyValue AutorisierungimAWS IoT SiteWise-Benutzerhandbuchaus.

Exportieren in AWS IoT SiteWise

So erstellen Sie einen Stream, der nach AWS IoT SiteWise, Ihre Lambda-FunktionenErstellen eines -Streamsmit einer Exportdefinition, die eine oder mehrere enthält`IoTSiteWiseConfig`-Objekte. Dieses Objekt definiert Exporteinstellungen wie Stapelgröße, Stapelintervall und Priorität.

Wenn Ihre Lambda-Funktionen Asset-Eigenschaftsdaten von Geräten erhalten, hängen sie Nachrichten an, die die Daten enthalten, an den Zielstream an. Nachrichten werden JSON-

serialisiert`PutAssetPropertyValueEntry`-Objekte, die Eigenschaftswerte für eine oder mehrere Asset-Eigenschaften enthalten. Weitere Informationen finden Sie unterAnhängen einer NachrichtzumAWS IoT SiteWiseExportieren von -Zielen

> **Note**
>
> Wenn Sie Daten an`AWS IoT SiteWise`, müssen Ihre Daten die Anforderungen des`BatchPutAssetPropertyValue`Aktion Weitere Informationen finden Sie unter BatchPutAssetPropertyValue in der AWS IoT SiteWise-API-Referenz.

Dann exportiert der Stream Manager die Daten basierend auf den 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. Zum Beispiel:

- Legen Sie für nahezu Echtzeit-Exporte die Einstellungen für niedrige Chargengröße und Intervall fest und hängen Sie die Daten an den Stream an, wenn sie empfangen werden.
- Um das Batching zu optimieren, Bandbreitenbeschränkungen zu mindern oder Kosten zu minimieren, können Ihre Lambda-Funktionen timestamp-quality-value (TQV) Datenpunkte, die für eine einzelne Asset-Eigenschaft empfangen wurden, bevor die Daten an den Stream angehängt werden. Eine Strategie besteht darin, Einträge für bis zu 10 verschiedene Eigenschaft-Asset-Kombinationen oder Eigenschaftsaliase in einer Nachricht zu stapeln, anstatt mehr als einen Eintrag für dieselbe Eigenschaft zu senden. Dies hilft dem Stream-Manager, innerhalb zu bleibenAWS IoT SiteWiseQuotenaus.

Amazon S3 S3-Objekte

Der Stream-Manager unterstützt automatische Exporte nach Amazon S3. Mit Amazon S3 können Sie große Datenmengen speichern und abrufen. Weitere Informationen finden Sie unterWas ist Amazon S3?imAmazon Simple Storage Service – Entwicklerhandbuchaus.

In derAWS IoT GreengrassCore SDK, Ihre Lambda-Funktionen nutzen das`S3ExportTaskExecutorConfig`um die Exportkonfiguration für diesen Zieltyp zu definieren. Weitere Informationen finden Sie in der SDK-Referenz für Ihre Zielsprache:

- s3ExportTaskExecutorConfigim Python SDK
- s3ExportTaskExecutorConfigim Java SDK
- s3ExportTaskExecutorConfigim SDK Node.js

Voraussetzungen

Dieses Exportziel hat die folgenden Anforderungen:

- Amazon S3 S3-Zielbuckets müssen im selben seinAWS-Kontoals Greengrass-Gruppe.
- Wenn das SymbolStandardcontainerisierungFür die Greengrass-Gruppe istGreengrass-Containereinstellen, müssen SieSTREAM_MANAGER_READ_ONLY_DIRSParameter zur Verwendung eines Eingabedateiverzeichnisses unter`/tmp`oder befindet sich nicht im Root-Dateisystem.
- Wenn eine Lambda-Funktion in läuftGreengrass-Containermode schreibt Eingabedateien in das Eingabedateiverzeichnis, Sie müssen eine lokale Volume-Ressource für das Verzeichnis erstellen und das Verzeichnis mit Schreibberechtigungen in den Container einhängen. Dies stellt sicher, dass die Dateien in das Root-Dateisystem geschrieben und außerhalb des Containers sichtbar sind. Weitere Informationen finden Sie unter Zugreifen auf lokale Ressourcen.
- Diethe section called “Greengrass-Gruppenrolle.”muss die folgenden Berechtigungen für die Ziel-Buckets zulassen. Zum Beispiel:

```
{
    "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/*"
```

```
            ]
        }
    ]
}
```

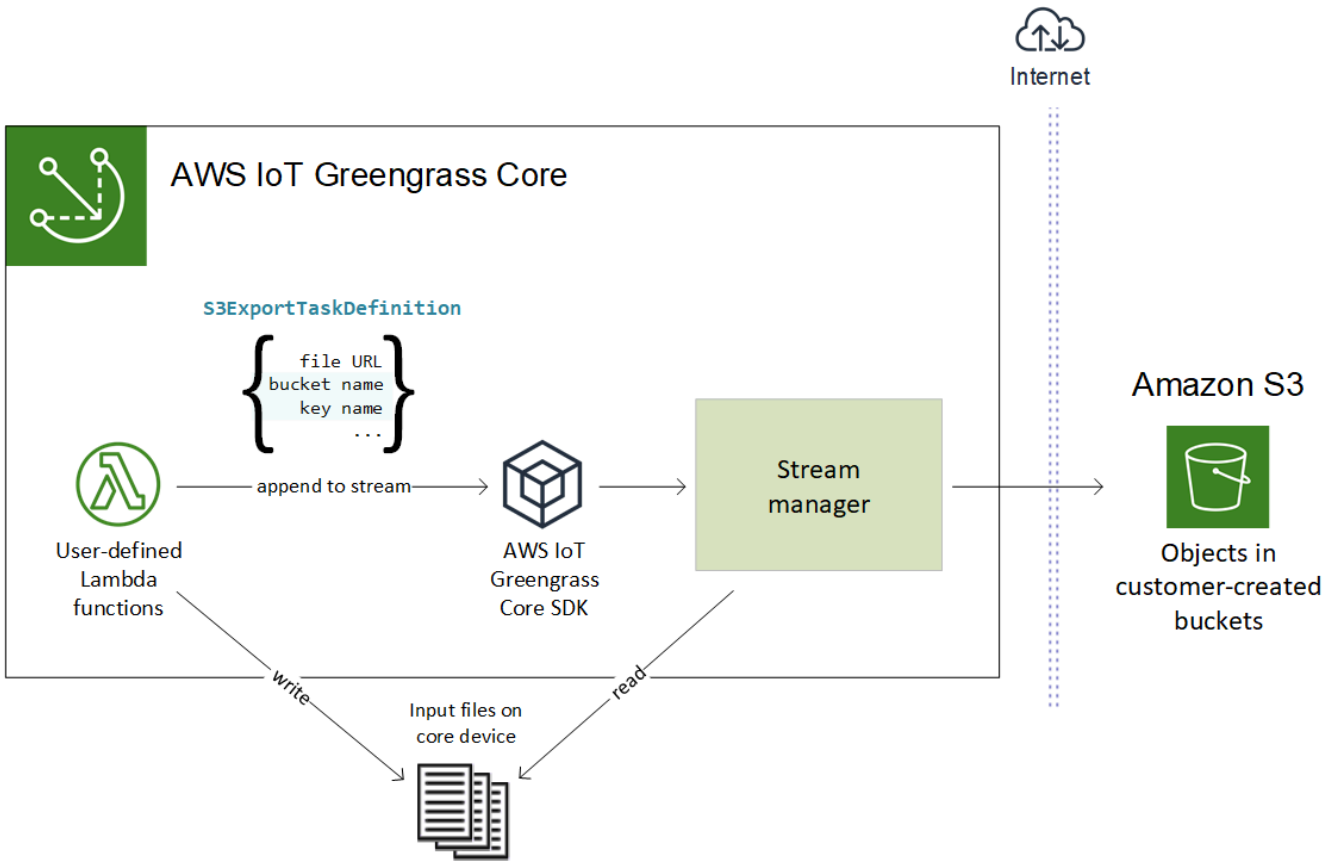
Sie können granularen oder bedingten Zugriff gewähren, etwa mit einem Platzhaltern*Benennungsschema. Weitere Informationen finden Sie unterHinzufügen und Entfernen von IAM-RichtlinienimIAM User Guideaus.

Exportieren nach Amazon S3

Um einen Stream zu erstellen, der nach Amazon S3 exportiert, verwenden Ihre Lambda-Funktionen die`S3ExportTaskExecutorConfig`-Objekt, um die Exportrichtlinie zu konfigurieren. Die Richtlinie definiert Exporteinstellungen wie den Schwellenwert für den mehrteiligen Upload und die Priorität. Bei Amazon S3 S3-Exporten 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 Zielobjekt von Amazon S3. Der Stream-Manager führt Aufgaben in der Reihenfolge aus, in der sie an den Stream angehängt werden.

> **Note**
>
> Der Ziel-Bucket muss in Ihrem bereits vorhanden seinAWS-Kontoaus. Wenn ein Objekt für den angegebenen Schlüssel nicht vorhanden ist, erstellt der Stream-Manager das Objekt für Sie.

Dieser Workflow auf hoher Ebene ist im folgenden Diagramm dargestellt.



Der Stream-Manager verwendet die Eigenschaft Multipart-Upload-Schwellenwert,Minimale TeilegrößeEinstellung und Größe der Eingabedatei, um zu bestimmen, wie Daten hochgeladen werden sollen. Der Mehrpart-Upload-Schwellenwert muss größer oder gleich der minimalen Teilegröße sein. Wenn Sie Daten parallel hochladen möchten, können Sie mehrere Streams erstellen.

Die Schlüssel, die Ihre Amazon S3 S3-Zielobjekte angeben, können gültig enthaltenJava DateTimeFormatterStrings in`!{timestamp:`*`value`*`}`Platzhalter. Sie können diese Zeitstempel-Platzhalter verwenden, um Daten in Amazon S3 basierend auf dem Zeitpunkt zu partitionieren, zu dem die Eingabedateidaten hochgeladen wurden. Der folgende Schlüsselname wird beispielsweise in einen Wert wie`my-key/2020/12/31/data.txt`aus.

```
my-key/!{timestamp:YYYY}/!{timestamp:MM}/!{timestamp:dd}/data.txt
```

> **Note**
>
> Wenn Sie den Exportstatus für einen Stream überwachen möchten, erstellen Sie zuerst einen Statusstream und konfigurieren Sie dann den Exportstream, um ihn zu verwenden. Weitere Informationen finden Sie unter the section called “Überwachen von Exportaufgaben”.

Verwalten von Eingabedaten

Sie können Code erstellen, den IoT-Anwendungen verwenden, um den Lebenszyklus der Eingabedaten zu verwalten. Der folgende Beispielworkflow 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**
   >
   > Um festzustellen, ob Sie den Zugriff auf das Eingabedateiverzeichnis konfigurieren müssen, lesen Sie dieSTREAM_MANAGER_READ_ONLY_DIRS-Parameter. Der Vorgang, in dem der Stream-Manager ausgeführt wird, erbt alle Dateisystemberechtigungen desStandardzugriffsidentitätfür die Gruppe. Der Stream-Manager muss zum Zugriff auf die Eingabedateien berechtigt sein. Sie können das`chmod(1)`-Befehl, um die Berechtigung der Dateien gegebenenfalls zu ändern.

2. Eine Lambda-Funktion scannt das Verzeichnis undhängt eine Exportaufgabezum Zielstream, wenn eine neue Datei erstellt wird. Die Aufgabe ist ein JSON-serialisierte`S3ExportTaskDefinition`-Objekt, das die URL der Eingabedatei, den Amazon S3 S3-Ziel-Bucket und Schlüssel sowie optionale Benutzermetadaten angibt.
3. Der Stream-Manager liest die Eingabedatei und exportiert die Daten in der Reihenfolge der angehängten Aufgaben nach Amazon S3. Der Ziel-Bucket muss in Ihrem bereits vorhanden seinAWS-Kontoaus. Wenn ein Objekt für den angegebenen Schlüssel nicht vorhanden ist, erstellt der Stream-Manager das Objekt für Sie.
4. Die Lambda-Funktionliest Nachrichtenaus einem Statusstream, um den Exportstatus zu überwachen. Nachdem Exportaufgaben abgeschlossen sind, kann die Lambda-Funktion die

entsprechenden Eingabedateien löschen. Weitere Informationen finden Sie unter the section called "Überwachen von Exportaufgaben".

Überwachen von Exportaufgaben

Sie können Code erstellen, mit dem IoT-Anwendungen den Status Ihrer Amazon S3 S3-Exporte überwachen. Ihre Lambda-Funktionen müssen einen Status-Stream erstellen und dann den Exportstream so konfigurieren, dass Statusaktualisierungen in den Statusstream geschrieben werden. Ein einzelner Statusstream kann Statusaktualisierungen von mehreren Streams erhalten, die nach Amazon S3 exportieren.

Erstens:Erstellen eines -Streamsals Statusstream zu verwenden. Sie können die Größe und Aufbewahrungsrichtlinien für den Stream konfigurieren, um die Lebensdauer der Statusmeldungen zu steuern. Zum Beispiel:

- Legen Sie fest.`PersistencezuMemory`wenn Sie die Statusmeldungen nicht speichern möchten.
- Legen Sie fest.`StrategyOnFullzuOverwriteOldestData`damit neue Statusmeldungen nicht verloren gehen.

Erstellen oder aktualisieren Sie dann den Exportstream, um den Statusstream zu verwenden. Legen Sie insbesondere die Statuskonfigurationseigenschaft des Streams fest`S3ExportTaskExecutorConfig`Exportieren der Konfiguration. Dies weist Stream-Manager an, Statusmeldungen über die Exportaufgaben in den Statusstream zu schreiben. In der`StatusConfig`-Objekt, geben Sie den Namen des Status-Streams und die Ausführlichkeitsstufe an. Die folgenden unterstützten Werte reichen von am wenigsten ausführlich (`ERROR`) auf ausführlichste (`TRACE`) enthalten. Der Standardwert ist `INFO`.

- `ERROR`
- `WARN`
- `INFO`
- `DEBUG`
- `TRACE`

Der folgende Beispiel-Workflow zeigt, wie Lambda-Funktionen einen Statusstream verwenden können, um den Exportstatus zu überwachen.

1. Wie im vorherigen Workflow beschrieben, ist eine Lambda-Funktionhängt eine Exportaufgabein einen Stream, der so konfiguriert ist, dass Statusmeldungen über Exportaufgaben in einen Statusstream geschrieben werden. Der Append-Vorgang gibt eine Sequenznummer zurück, die die Aufgaben-ID darstellt.
2. Eine Lambda-Funktionliest Nachrichtensequenziell aus dem Status-Stream und filtert die Nachrichten dann basierend auf dem Stream-Namen und der Aufgaben-ID oder basierend auf einer Exportaufgabeneigenschaft aus dem Nachrichtenkontext. Beispielsweise kann die Lambda-Funktion nach der Eingabedatei-URL der Exportaufgabe filtern, die durch die`S3ExportTaskDefinition`-Objekt im Nachrichtenkontext.

   Die folgenden Statuscodes zeigen an, dass eine Exportaufgabe den Status „Abgeschlossen“ erreicht hat:

   - `Success`aus. Der Upload wurde erfolgreich abgeschlossen.
   - `Failure`aus. Der Stream-Manager ist auf einen Fehler gestoßen, z. B. existiert der angegebene Bucket nicht. Nachdem Sie das Problem behoben haben, können Sie die Exportaufgabe erneut an den Stream anhängen.
   - `Canceled`aus. Die Aufgabe wurde abgebrochen, weil die Stream- oder Exportdefinition gelöscht wurde, oder die time-to-live (TTL) Der Zeitraum der Aufgabe ist abgelaufen.

   > **Note**
   >
   > Die Aufgabe könnte auch den Status von haben`InProgress`oder`Warning`aus. Der Stream-Manager gibt Warnungen aus, wenn ein Ereignis einen Fehler zurückgibt, der sich nicht auf die Ausführung der Aufgabe auswirkt. Ein Fehler beim Bereinigen eines abgebrochenen teilweisen Uploads gibt beispielsweise eine Warnung zurück.
3. Nachdem Exportaufgaben abgeschlossen sind, 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.
```

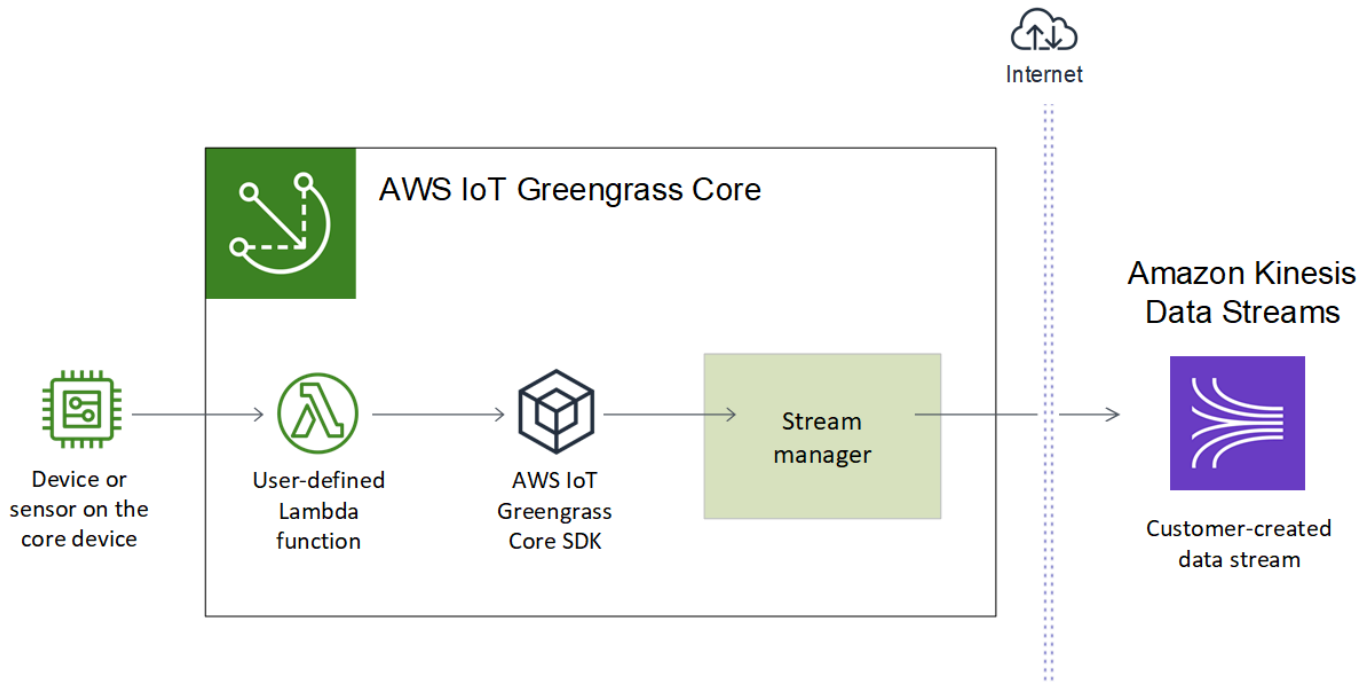
```
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Node.js SDK-Referenz:readMessages|StatusMessage

Exportieren von Daten-Streams inAWS Cloud(Konsole)

In diesem Tutorial erfahren Sie, wie Sie das verwendenAWS IoT-Konsole zum Konfigurieren und Bereitstellen einerAWS IoT GreengrassGruppe mit aktiviertem Stream-Manager. Die Gruppe enthält eine benutzerdefinierte Lambda-Funktion, die in einen Stream im Stream-Manager schreibt, der dann automatisch in denAWS Cloud.

Der Stream-Manager macht das Aufnehmen, Verarbeiten und Exportieren von Datenstreams mit hohem Volumen effizienter und zuverlässiger. In diesem Tutorial erstellen Sie einen`TransferStream`Lambda-Funktion, die IoT-Daten verbraucht. Die Lambda-Funktion verwendet denAWS IoT GreengrassCore-SDK, um einen Stream im Stream-Manager zu erstellen und ihn dann zu lesen und darin zu schreiben. Der Stream-Manager exportiert dann den Stream in Kinesis Data Streams. Im folgenden Diagramm wird dieser Workflow veranschaulicht.



Der Fokus dieses Lernprogramms ist es zu zeigen, wie benutzerdefinierte Lambda-Funktionen den`StreamManagerClient`-Objekt imAWS IoT GreengrassCore-SDK für die Interaktion mit dem Stream-Manager. Der Einfachheit halber generiert die Python-Lambda-Funktion, die Sie für dieses Lernprogramm erstellen, simulierte Gerätedaten.

Voraussetzungen

Zum Durchführen dieses Tutorials benötigen Sie Folgendes:

- Eine Greengrass-Gruppe und ein Greengrass-Core (v1.10 oder höher). Weitere Informationen zum Erstellen einer Greengrass-Gruppe und Greengrass Core finden Sie unterErste Schritte mit AWS IoT Greengrass. Das Tutorial „Erste Schritte“ enthält auch die Schritte zur Installation der AWS IoT Greengrass-Core Software.

  > **Note**
  >
  > Der Stream-Manager wird auf OpenWrt -Verteilungen.

- 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 GreengrassCore-SDK für Python v1.5.0 oder höher. Um zu verwenden`StreamManagerClient`imAWS IoT GreengrassCore SDK für Python müssen Sie:
  - Python 3.7 oder höher auf dem Core-Gerät installieren.
  - 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 unterAWS IoT GreengrassCore SDK for JavaundAWS IoT GreengrassCore SDK für Node.jsauf GitHub.

- Ein Zielstream mit dem Namen**`MyKinesisStream`**erstellt in Amazon Kinesis Data Streams im selbenAWS-Regionals Ihre Greengrass-Gruppe. Weitere Informationen finden Sie unterErstellen eines StreamsimEntwicklerhandbuch für Amazon Kinesis.

  > **Note**
  >
  > In diesem Lernprogramm exportiert der Stream-Manager Daten in Kinesis Data Streams, was zu Gebühren für Ihr führtAWS-Konto. Weitere Informationen zu Preisen finden Sie unterKinesis Data Streams Streams-Preisgestaltung.
  > Um Kosten zu vermeiden, können Sie dieses Lernprogramm ausführen, ohne einen Kinesis-Datenstream zu erstellen. In diesem Fall überprüfen Sie die Protokolle, um zu sehen, dass der Stream-Manager versucht hat, den Stream in Kinesis Data Streams zu exportieren.

- Eine IAM-Richtlinie wurde zurthe section called “Greengrass-Gruppenrolle.”das erlaubt`kinesis:PutRecords`Aktion auf dem Zieldaten-Stream, wie im folgenden Beispiel gezeigt:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecords"
            ],
            "Resource": [
                "arn:aws:kinesis:region:account-id:stream/MyKinesisStream"
            ]
        }
    ]
}
```

Das Tutorial enthält die folgenden allgemeinen Schritte:

1. Bereitstellungspaket für die Lambda-Funktion erstellen
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: Bereitstellungspaket für die Lambda-Funktion erstellen

In diesem Schritt erstellen Sie ein Bereitstellungspaket für Lambda-Funktionen, das Python —Funktionscode und Abhängigkeiten enthält. Sie laden dieses Paket später hoch, wenn Sie das erstellenLambda-Funktion inAWS Lambda. Die Lambda-Funktion verwendet denAWS IoT GreengrassCore-SDK zum Erstellen und Interagieren mit lokalen Streams.

> **Note**
>
> Ihre benutzerdefinierten Lambda-Funktionen müssen dieAWS IoT GreengrassCore-SDKum 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. Herunterladen desAWS IoT GreengrassCore SDK für Pythonv1.5.0 oder höher.
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 Bereitstellungspaket für Lambda-Funktionen aufgenommen werden

    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 unterAWS IoT GreengrassCore SDK for JavaundAWS IoT GreengrassCore SDK für Node.jsauf 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, "ABCDEFGHIJKLMNO".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 das Bereitstellungspaket Ihrer Lambda-Funktion.

   - transfer_stream.py. App-Logik.
   - greengrasssdk. Erforderliche Bibliothek für Python Greengrass-Lambda-Funktionen, die MQTT-Nachrichten veröffentlichen.

     Stream-Manager-Betriebsind in Version 1.5.0 oder höher derAWS IoT GreengrassCore SDK für Python.
   - Die Abhängigkeiten, die Sie für dasAWS IoT GreengrassCore-SDK for Python (z. B.`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 denAWS Lambda-Konsole erstellen und diese zur Verwendung des Bereitstellungspakets konfigurieren. Anschließend veröffentlichen Sie eine Funktionsversion und erstellen einen Alias.

1. Zunächst erstellen Sie die Lambda-Funktion.

   a. Wählen Sie in der AWS Management Console Services und öffnen Sie die AWS Lambda-Konsole.

   b. Klicken Sie aufFunktion erstellenund wählen Sie dannAuthor from scratch.

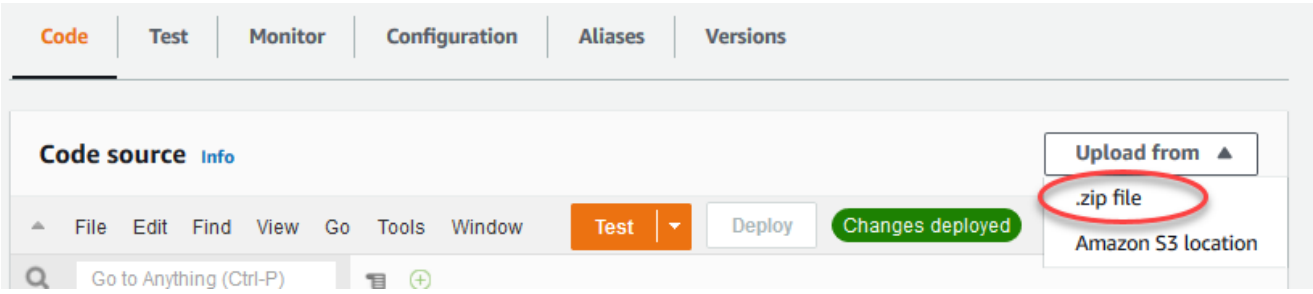
   c. Verwenden Sie im Abschnitt Basic information (Basisinformationen) folgende Werte:

      - Geben Sie für Function name (Funktionsname) **`TransferStream`** ein.
      - Wählen Sie für Runtime (Laufzeit) die Option Python 3.7 aus.
      - FürBerechtigungenverwenden, behalten Sie die Standardeinstellung bei. Dadurch wird eine Ausführungsrolle erstellt, die grundlegende Lambda-Berechtigungen gewährt. Diese Rolle wird nicht von verwendetAWS IoT Greengrass.

   d. Klicken Sie unten auf der Seite auf Create function.

2. Registrieren Sie jetzt den Handler und laden Sie Ihr Bereitstellungspaket für Lambda-Funktionen hoch

   a. Auf derCodeunterQuellcode, wählenHochladen von. Wählen Sie in der Dropdown-Liste die OptionZIP-Datei.



b. Klicken Sie aufHochladenund wählen Sie dann Ihr`transfer_stream_python.zip`Bereitstellungspaket. Wählen Sie dann Save (Speichern) aus.

c. Auf derCodefür die Funktion, unterLaufzeit-Einstellungen, wählenBearbeiten, und geben Sie 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 Save (Speichern) aus.

> **Note**
>
> DieTest-Schaltfläche auf derAWS Lambda-Konsole funktioniert nicht mit dieser -Funktion. DieAWS IoT GreengrassDas Core SDK enthält keine Module, die erforderlich sind, um Ihre Greengrass Lambda-Funktionen unabhängig imAWS Lambdaconsole. Diese Module (zum Beispiel`greengrass_common`) werden für die Funktionen bereitgestellt, nachdem sie in Ihrem Greengrass-Kern bereitgestellt wurden.

3. Veröffentlichen Sie jetzt die erste Version der Lambda-Funktion und erstellen Sie einAlias für die Version.

> **Note**
>
> Greengrass-Gruppen können eine Lambda-Funktion nach Alias (empfohlen) oder nach Version referenzieren. Mit einem Alias lassen sich Code-Updates einfacher verwalten, da die Abonnementtabelle oder Gruppendefinition nicht geändert werden muss, 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. Auf derTransferStream: 1-Konfigurationsseite, von derAktionen-Menü wählen SieErstellen eines Alias.

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 Greengrassunterstützt keine Lambda-Aliasse für$LATESTVersionen.

e. Wählen Sie Create (Erstellen) aus.

Jetzt sind Sie bereit, die Lambda-Funktion zu Ihrer Greengrass-Gruppe hinzuzufügen.

Schritt 3: Hinzufügen einer Lambda-Funktion zur Greengrass-Gruppe

In diesem Schritt fügen Sie die Lambda-Funktion der Gruppe hinzu und konfigurieren dann den Lebenszyklus und die Umgebungsvariablen. Weitere Informationen finden Sie unter the section called “Steuern der Ausführung der Greengrass-Lambda-Funktion”.

1. In derAWS IoTNavigationsbereich der -Konsole unterVerwalten, erweiternGreengrass-Geräteund wählen Sie dannGruppen (V1).
2. Wählen Sie die Zielgruppe aus.
3. Wählen Sie auf der Gruppenkonfigurationsseite die OptionLambda-FunktionenRegisterkarte.
4. UnderMeine Lambda-Funktionen, wählenAdd.
5. Auf derHinzufügen einer Lambda-Funktion-Seite wählen Sie die OptionLambda-Funktionfür Ihre Lambda-Funktion.
6. Für denLambda-Version, wählenAlias: GG_TransferStream.

   Konfigurieren Sie jetzt Eigenschaften, die das Verhalten der Lambda-Funktion in der Greengrass-Gruppe bestimmen.
7. In derKonfiguration der Lambda-Funktiondie folgenden Änderungen durch:

   - Legen Sie die Speichergrenze auf 32 MB fest.
   - FürPinned, wählenWahr.

> **Note**
>
> EINlangdauerndem(oderfestgeheftet) Lambda-Funktion startet automatisch danachAWS IoT Greengrassstartet und läuft weiter in ihrem eigenen Container. Dies steht im Gegensatz zu einemauf AnforderungLambda-Funktion, die nach einem Aufruf startet und beendet wird, sobald keine weiteren Aufgaben auszuführen sind. Weitere Informationen finden Sie unter the section called “Lebenszyklus-Konfiguration”.

8. Klicken Sie aufHinzufügen einer Lambda-Funktion.

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 OptionLambda-FunktionenRegisterkarte.
2. UnderLambda-Funktionen, wählen SieStream-Managerund 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 Lernprogramm verwenden. Weitere Informationen finden Sie unter the section called “Konfigurieren des -Stream-Managers”.

> **Note**
>
> Wenn Sie die Konsole verwenden, um den Stream-Manager zu aktivieren und die Gruppe bereitzustellen, wird die Speichergröße für den Stream-Manager standardmäßig auf 4194304 KB (4 GB) festgelegt. Wir empfehlen, dass Sie die Speichergröße auf mindestens 128000 KB einstellen.

Schritt 5: Konfigurieren der lokalen Protokollierung

In diesem Schritt konfigurieren SieAWS IoT Greengrass-Systemkomponenten, benutzerdefinierte Lambda-Funktionen und Konnektoren in der Gruppe, um Protokolle in das Dateisystem des Core-Geräts 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 keine Protokolle für Greengrass-Systemkomponenten oder benutzerdefinierte Lambda-Funktionen konfiguriert sind, wählen SieBearbeiten.
3. Klicken Sie aufBenutzer-Lambda-Funktionen auf ProtokollebeneundGreengrass-Systemprotokollebene.
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 dieAWS IoT Greengrasscore 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 in dem Pfad hängt von der AWS IoT Greengrass-Core-Softwareversion ab, die auf Ihrem Core-Gerät installiert ist.

   b. So starten Sie den Daemon:

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

2. Wählen Sie auf der Gruppenkonfigurationsseite die OptionBereitstellen.

3. a. In derLambda-Funktionenunter der RegisterkarteSystem-Lambda-FunktionenAbschnitt, wählen SieIP-Detektorund wähleBearbeiten.

   b. In derEinstellungen für IP-Detektor bearbeiten-Dialogfeld wählen SieAutomatisches Erkennen und Überschreiben von MQTT-Broker-.

   c. Wählen Sie Save (Speichern) 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.

   > **Note**
   >
   > Erteilen Sie bei Aufforderung die Berechtigung zum Erstellen desGreengrass-Servicerolleund assoziiere es mit deinemAWS-Kontoin der aktuellenAWS-Region. Diese Rolle erlaubtAWS IoT Greengrassum auf Ihre -Ressourcen zuzugreifenAWS-Services.

   Auf der Seite Deployments werden der Zeitstempel, die Versions-ID und der Status der Bereitstellung angegeben. Nach abgeschlossener Bereitstellung sollte der StatusCompleted (Abgeschlossen).

   Hilfe zur Problembehebung finden Sie unter Fehlerbehebung.

Schritt 7: Testen der Anwendung

Die`TransferStream`Die Lambda-Funktion erzeugt simulierte Gerätedaten. Sie schreibt Daten in einen Stream, den der Stream-Manager in den Kinesis-Zieldatenstream exportiert.

1. In der Amazon Kinesis Kinesis-Konsole unterKinesis Data Streams, wählenMyKinesisStream.

   > **Note**
   >
   > Wenn Sie das Lernprogramm ohne Kinesis-Zieldatenstream 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. Auf derMyKinesisStream-Seite wählen SieÜberwachung. 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. Remove dasTransferStreamLambda-Funktion aus dem Core.

   a. In derAWS IoTNavigationsbereich der -Konsole unterVerwalten, erweiternGreengrass-Geräteund wählen Sie dannGruppen (V1).

   b. UnderGreengrass-Gruppenwählen Sie Ihre Gruppe aus.

   c. Auf derLambdadie Ellipsen (...) für dasTransferStream-Funktion und wählen Sie dannFunktion entfernen.

   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 Problembehandlung benötigen, können Siesetzen der Protokollierungsstufezum Lambda-Protokollezu-Debug-Protokolleund stellen anschließend die Gruppe erneut bereit.

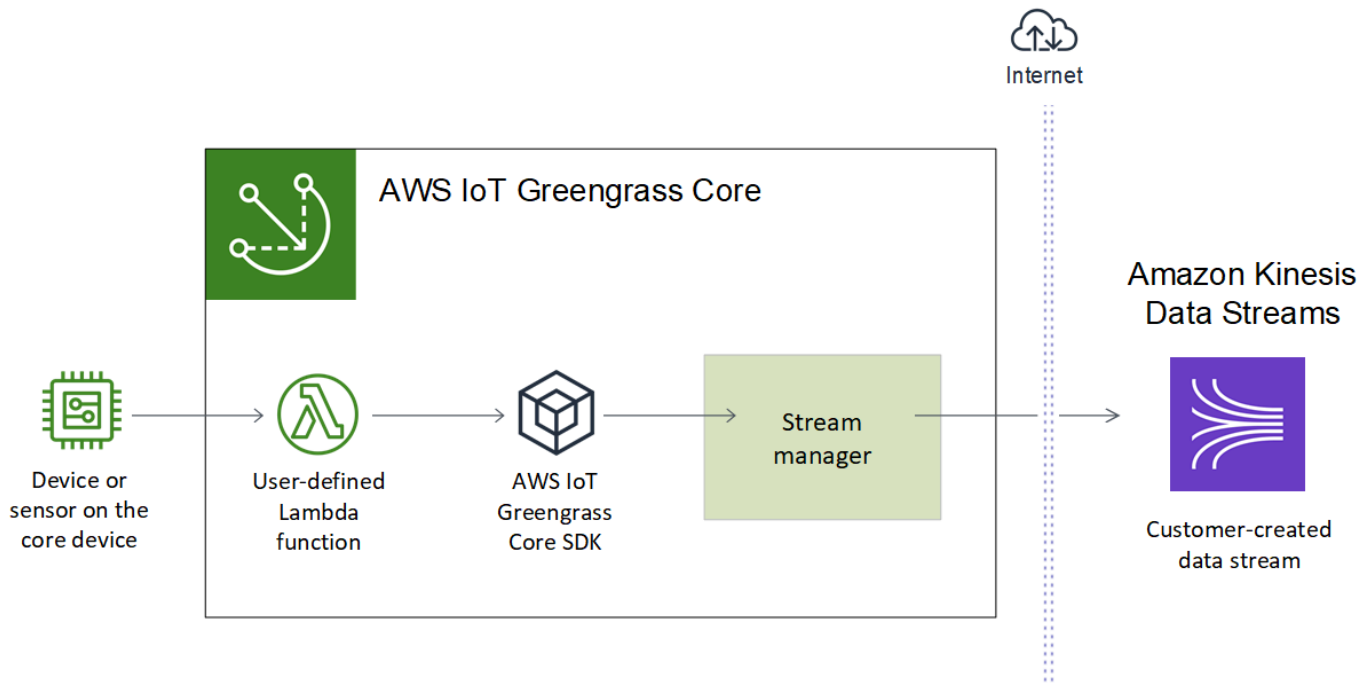
Weitere Informationen finden Sie auch unter

- Verwalten von Daten-Streams
- the section called “Konfigurieren des -Stream-Managers”
- the section called “Verwenden von StreamManagerClient um mit Streams zu arbeiten”
- the section called “Exportieren von Konfigurationen für unterstützteAWS CloudDestinationen”
- the section called “Exportieren von Daten-Streams (CLI)”

Exportieren von Daten-Streams inAWS Cloud(CLI)

Dieses Tutorial zeigt, wie Sie dasAWS CLIzur Konfiguration und Bereitstellung einerAWS IoT GreengrassGruppe mit aktiviertem Stream-Manager. Die Gruppe enthält eine benutzerdefinierte Lambda-Funktion, die in einen Stream im Stream-Managers schreibt, der dann automatisch in denAWS Cloudaus.

Der Stream-Manager macht das Aufnehmen, Verarbeiten und Exportieren von Datenstreams mit hohem Volumen effizienter und zuverlässiger. In diesem Tutorial erstellen Sie einen`TransferStream`Lambda-Funktion, die IoT-Daten verbraucht. Die Lambda-Funktion verwendet denAWS IoT GreengrassCore SDK, um einen Stream im Stream-Managers zu erstellen und ihn dann zu lesen und darin zu schreiben. Der Stream-Manager exportiert dann den Stream in Kinesis Data Streams. Im folgenden Diagramm wird dieser Workflow veranschaulicht.



Der Fokus dieses Lernprogramms ist es zu zeigen, wie benutzerdefinierte Lambda—FunktionenStreamManagerClient-Objekt imAWS IoT GreengrassCore-SDK zur Interaktion mit dem Stream-Manager. Der Einfachheit halber generiert die Python Lambda-Funktion, die Sie für dieses Lernprogramm erstellen.

Wenn Sie dasAWS IoT GreengrassAPI, die die Greengrass-Befehle in derAWS CLI, um eine Gruppe zu erstellen, ist der Stream-Managers standardmäßig deaktiviert. Um den Stream-Manager auf Ihrem Core zu aktivieren, müssen SieErstellen einer -Funktionsdefinitionsversiondas beinhaltet das SystemGGStreamManagerLambda-Funktion 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 Core (v1.10 oder höher). Weitere Informationen wie Sie eine Greengrass-Gruppe und Greengrass Core erstellen finden Sie unterErste Schritte mit AWS IoT Greengrassaus. Das Tutorial „Erste Schritte“ enthält auch die Schritte zur Installation der AWS IoT Greengrass-Core Software.

> **Note**
>
> Der Stream-Manager wird auf nicht unterstützt OpenWrt -Verteilungen.

- 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 GreengrassCore SDK for Python v1.5.0 oder höher. Um zu verwenden`StreamManagerClient`imAWS IoT GreengrassCore SDK für Python müssen Sie:
  - Python 3.7 oder höher installieren. Python 3.7 oder höher installieren.
  - 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 unterAWS IoT GreengrassCore-SDK SDK for JavaundAWS IoT GreengrassCore-SDK for Node.jsauf GitHubaus.

- Ein Zielstream mit dem Namen**`MyKinesisStream`**erstellt in Amazon Kinesis Data Streams im gleichenAWS-Regionals Ihre Greengrassgrassgrass-Gruppe. Weitere Informationen finden Sie unterErstellen eines -StreamsimEntwicklerhandbuch für Amazon Kinesisaus.

> **Note**
>
> In diesem Lernprogramm exportiert der Stream-Managers Daten in Kinesis Streams, was zu GebührenAWS-Kontoaus. Weitere Informationen zu -Preisen erhalten Sie unterKinesis Data Streams Streams-Preiseaus.
> Um Kosten zu vermeiden, können Sie dieses Lernprogramm ausführen, ohne einen Kinesis-Datenstream zu erstellen. In diesem Fall überprüfen Sie die Protokolle, um zu sehen, dass der Stream-Managers versucht hat, den Stream in Kinesis Data Streams zu exportieren.

- Eine IAM-Richtlinie wurde zurthe section called “Greengrass-Gruppenrolle.”das erlaubt`kinesis:PutRecords`Aktion auf dem Zieldaten-Stream, wie im folgenden Beispiel gezeigt:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecords"
            ],
            "Resource": [
                "arn:aws:kinesis:region:account-id:stream/MyKinesisStream"
            ]
        }
    ]
}
```

- Die AWS CLI ist auf Ihrem Computer installiert und konfiguriert. Weitere Informationen finden Sie unterInstallieren vonAWS Command Line InterfaceundKonfigurieren vonAWS CLIimAWS Command Line Interface-Benutzerhandbuchaus.

  Die Beispielbefehle in diesem Tutorial sind für Linux und andere Unix-basierte Systeme geschrieben. Wenn Sie Windows verwenden, lesen Sie Folgendes:Angeben von Parameterwerten

für dieAWS-Befehlszeilenschnittstellefinden Sie weitere Informationen über Unterschiede 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 eines Bereitstellungspakets für Lambda
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 eines Bereitstellungspakets für Lambda

In diesem Schritt erstellen Sie ein Bereitstellungspaket, das Python—Funktionscode und Abhängigkeiten enthält. Sie laden dieses Paket später hoch, sobald Sie dasLambda-Funktion inAWS Lambdaaus. Die Lambda-Funktion verwendet denAWS IoT GreengrassCore--SDK zum Erstellen und Interagieren mit lokalen Stream-Managers.

> **Note**
>
> Ihre benutzerdefinierten Lambda-Funktionen müssen dieAWS IoT GreengrassCore-SDKum 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. Herunterladen desAWS IoT GreengrassCore-SDK für Pythonv1.5.0 oder höher.
2. Entpacken Sie das heruntergeladene Paket, um das SDK zu erhalten. Das SDK ist der `greengrasssdk`-Ordner.
3. Python installieren Sie Paketabhängigkeiten, die mit dem SDK in Ihr Bereitstellungspaket der Lambda-Funktion

   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 imAWS IoT GreengrassCore-SDK SDK for JavaundAWS IoT GreengrassCore-SDK for Node.jsauf GitHubaus.

   ```
   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, "ABCDEFGHIJKLMNO".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 das Bereitstellungspaket Ihrer Lambda-Funktion.

   - transfer_stream.py. App-Logik.

- greengrasssdk. Erforderliche Bibliothek für Python Greengrass Lambda—Funktionen, die MQTT-Nachrichten veröffentlichen.

  Stream-Manager-Betriebsind in Version 1.5.0 oder höherAWS IoT GreengrassCore-SDK für Python.
- Die Abhängigkeiten, die Sie für dasAWS IoT GreengrassCore SDK for Python (z. B. dascbor2-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 beim Anlegen der -Funktion den Rollen-ARN 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 Greengrassverwendet diese -Rolle nicht, da 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 `TransferStream`-Funktion zu erstellen. Der folgende Befehl geht davon aus, dass sich die Zip-Datei im aktuellen Verzeichnis befindet.

   - Ersetzen Sie *`role-arn`* durch den kopierten `Arn`.

```
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 per Alias (empfohlen) oder nach Version referenzieren. Die Verwendung eines Alias erleichtert die Verwaltung von Code-Updates, da Sie Ihre Abonnementtabelle oder Gruppendefinition nicht ändern müssen, sobald der Funktionscode aktualisiert wird. Stattdessen verweisen Sie einfach den Alias auf die neue Funktionsversion.

```
aws lambda create-alias --function-name TransferStream --name GG_TransferStream --
function-version 1
```

> **Note**
>
> AWS IoT Greengrassunterstützt keine Lambda-Aliase für$LAEST-Versionen.

6. Kopieren Sie die `AliasArn` aus der Ausgabe. Sie verwenden diesen Wert, wenn Sie die Funktion für AWS IoT Greengrass konfigurieren.

Jetzt sind Sie bereit, die Funktion für AWS IoT Greengrass zu konfigurieren.

Schritt 3: Erstellen einer Funktionsdefinition und -version

Dieser Schritt erstellt eine Funktionsdefinitionsversion, die auf das System verweist.`GGStreamManager`Lambda-Funktion und Ihre benutzerdefinierte`TransferStream`Lambda-Funktion. Um den Stream-Manager zu aktivieren, wenn Sie denAWS IoT GreengrassAPI muss Ihre Funktionsdefinitionsversion die`GGStreamManager`Funktion.

1. Erstellen Sie eine Funktionsdefinition, die die System- und benutzerdefinierten Lambda-Funktionen enthält.

   Die folgende Definitionsversionparameter-Einstellungenaus. Um benutzerdefinierte Einstellungen zu konfigurieren, müssen Sie Umgebungsvariablen für entsprechende Stream-Managers. Ein Beispiel finden Sie unterthe section called “Aktivieren, Deaktivieren oder Konfigurieren des Stream-Managers”aus.AWS IoT Greengrassverwendet Standardeinstellungen.`MemorySize`sollte mindestens erforderlich sein`128000`aus.`Pinned`muss auf festgelegt sein`true`aus.

   > **Note**
   >
   > EINvon langlebigen(oderfestgeheftet) Lambda-Funktion startet automatisch danachAWS IoT Greengrass-Container. Dies steht im Gegensatz zu einemauf AnforderungLambda-Funktion, die nach einem Aufruf startet und beendet wird, sobald keine weiteren Aufgaben auszuführen sind. Weitere Informationen finden Sie unter the section called “Lebenszyklus-Konfiguration”.

- Ersetzen*arbitrary-function-id*mit einem Namen für die Funktion, z. B.**stream-manager**aus.
- Ersetzen*Alias-arn*mit demAliasArndie Sie beim Erstellen des AliasTransferStreamLambda-Funktion.

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"}},"MemorySize":
 128000,"Pinned": true,"Timeout": 3}},{"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 andere Einstellungen auf Gruppenebene finden Sie unterthe section called “Steuern der Ausführung der Greengrass-Lambda-Funktion”aus.

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 SieAWS IoT Greengrass-Systemkomponenten, benutzerdefinierte Lambda—Funktionen und Konnektoren, die Protokolle in das Dateisystem des Core-Geräts 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":"FileSyste
{"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 die IDs der Greengrass-Zielgruppen und die Gruppenversion ab. Dieses Verfahren setzt voraus, dass es sich um die neueste Gruppe 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 inAWS IoTconsole. Die Gruppen-ID wird auf der Seite Einstellungen der Gruppe angezeigt. Gruppenversions-IDs werden auf demBereitstellungenRegisterkarte.

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.

   - Ersetzen Sie *`group-id`* durch die kopierte `Id` für die Gruppe.
   - Ersetzen*`group-version-id`*mit dem`LatestVersion`die du für die Gruppe kopiert hast.

   ```
   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

- Ersetzen Sie *group-id* durch die kopierte `Id` für die Gruppe.
- Ersetzen*core-definition-version-arn*mit dem`CoreDefinitionVersionArn`die Sie für die Core-Definitionsversion kopiert haben.
- Ersetzen*function-definition-version-arn*mit dem`LatestVersionArn`die Sie für die neue Funktionsdefinitionsversion kopiert haben.
- Ersetzen*logger-definition-version-arn*mit dem`LatestVersionArn`die Sie für die 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: Eine Bereitstellung auswählen

Stellen Sie die Gruppe auf dem Core-Gerät bereit.

1. Stellen Sie sicher, dass dieAWS IoT Greengrasscore 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 in dem Pfad hängt von der AWS IoT Greengrass-Core-Softwareversion ab, die auf Ihrem Core-Gerät installiert ist.

   b. So starten Sie den Daemon:

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

2. Erstellen einer Bereitstellung.

   - Ersetzen Sie *group-id* durch die kopierte `Id` für die Gruppe.
   - Ersetzen*group-version-id*mit dem`Version`die 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.

   - Ersetzen Sie *group-id* durch die kopierte `Id` für die Gruppe.
   - Ersetzen Sie *deployment-id* durch die `DeploymentId`, die 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`Die Lambda-Funktion erzeugt simulierte Gerätedaten. Sie schreibt Daten in einen Stream, den der Stream-Manager in den Kinesis-Zieldatenstream exportiert.

1. In der Amazon Kinesis Kinesis-Konsole unterKinesis Data Streams, wählenMyKinesisStreamaus.

> **Note**
>
> Wenn Sie das Lernprogramm ohne Kinesis-Zieldatenstream 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. Auf derMyKinesisStream-Seite, wählen SieÜberwachungaus. 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 dasTransferStreamLambda-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. Sie können auch eine Funktionsdefinitionsversion erstellen, dieTransferStreamLambda-Funktion.

   > **Note**
   >
   > Durch das Weglassen des Systems`GGStreamManagerLambda`—Funktion aus der bereitgestellten Gruppenversion, deaktivieren Sie die Stream-Managers.

   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 “Verwenden von StreamManagerClient um mit Streams zu arbeiten”
- the section called “Exportieren von Konfigurationen für unterstützteAWS CloudDestinationen”
- the section called “Konfigurieren des -Stream-Managers”
- the section called “Exportieren von Daten-Streams (Konsole)”
- AWS Identity and Access Management(IAM) -BefehleimAWS CLIBefehlsreferenz
- AWS LambdaKommandosimAWS CLIBefehlsreferenz
- AWS IoT GreengrassKommandosimAWS CLIBefehlsreferenz

Bereitstellen von Secrets für den AWS IoT Greengrass Core

Diese Funktion ist verfügbar fürAWS IoT GreengrassCore v1.7 und höher.

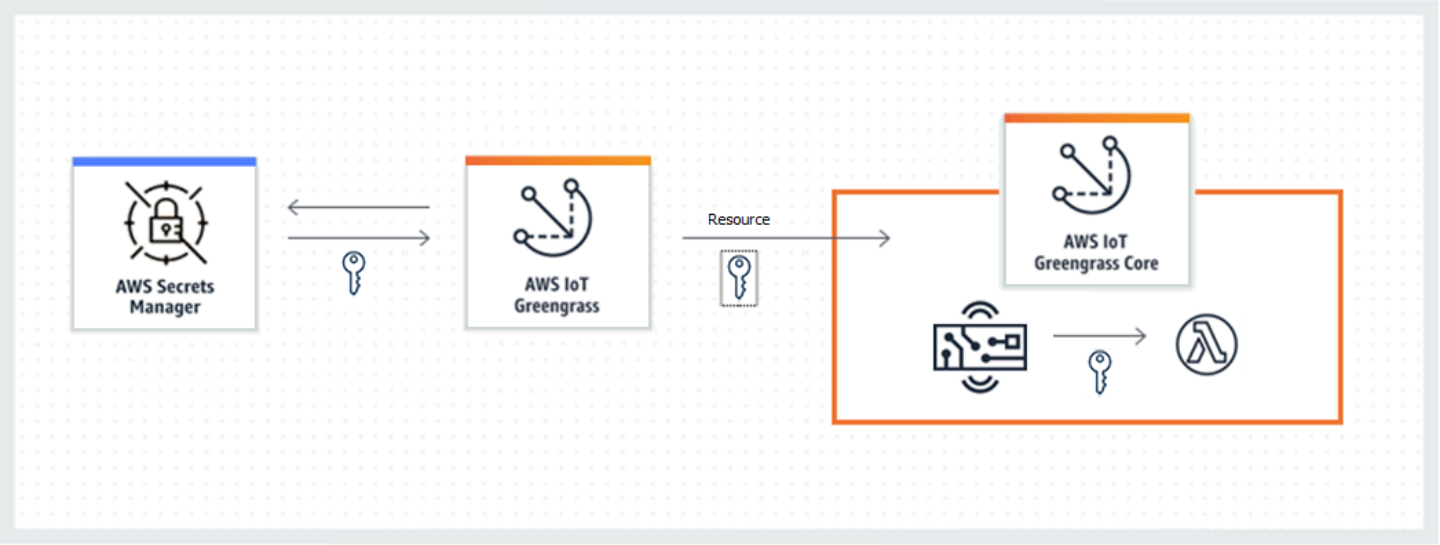
AWS IoT Greengrass ermöglicht Ihnen die Authentifizierung bei Services und Anwendungen aus Greengrass-Geräten, ohne Passwörter, Token oder andere Secrets fest kodieren zu müssen.

AWS Secrets Managerist ein Service, mit dem Sie Ihre Secrets sicher in der Cloud speichern und verwalten können.AWS IoT Greengrasserweitert Secrets Manager auf Greengrass-Kerngeräte, alsoSteckverbinderund Lambda-Funktionen können lokale Secrets für die Interaktion mit Services und Anwendungen verwenden. 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 Secrets während der Übertragung und im Ruhezustand. Während der GruppenbereitstellungAWS IoT Greengrassruft das Secret vom Secrets Manager ab und erstellt auf dem Greengrass-Core eine lokale verschlüsselte Kopie. Nachdem Sie Ihre Cloud-Secrets in Secrets Manager rotiert haben, stellen Sie die Gruppe erneut bereit, um die aktualisierten Werte an den Kern zu übertragen.

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 von AWS IoT Greengrass zur lokalen Speicherung Ihrer Secrets bietet die folgenden 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 “Secrets-Verschlüsselung”.
- Kontrollierte Rotation. Nachdem Sie Ihre Secrets in 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 Secrets nicht automatisch, nachdem Cloud-Versionen rotiert wurden. Um lokale Werte zu aktualisieren, müssen Sie die Gruppe erneut bereitstellen.

Secrets-Verschlüsselung

AWS IoT Greengrass verschlüsselt Geheimnisse während der Übertragung und im Ruhezustand.

> **Important**
>
> Stellen Sie sicher, dass Ihre benutzerdefinierten Lambda-Funktionen Geheimnisse sicher behandeln und keine sensiblen Daten protokollieren, die im Geheimnis gespeichert sind. Weitere Informationen finden Sie unterVerhindern von Risiken durch die Protokollierung oder das Debuggen Ihrer Lambda-FunktionimAWS Secrets Manager-Benutzerhandbuchaus. Obwohl sich diese Dokumentation speziell auf Rotationsfunktionen bezieht, gilt die Empfehlung auch für Greengrass Lambda-Funktionen.

Verschlüsselung während der Übertragung

AWS IoT Greengrass verwendet die Transport Layer Security (TLS) zur Verschlüsselung der gesamten Kommunikation über das Internet und das lokale Netzwerk. Dies schützt Secrets während der Übertragung, was der Fall ist, wenn Secrets vom 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 `config.json` angegebenen privaten Schlüssel zur Verschlüsselung der Secrets, die auf dem Core gespeichert sind. Aus diesem Grund ist die sichere Speicherung des privaten Schlüssels für den Schutz lokaler Secrets entscheidend. Im AWS Modell der übergreifenden Verantwortlichkeit liegt es in der Verantwortung des Kunden, die sichere Speicherung des privaten Schlüssels auf dem Core-Gerät zu gewährleisten.

AWS IoT Greengrass unterstützt zwei Arten der Speicherung privater Schlüssel:

- Die Verwendung von Hardware-Sicherheitsmodulen. Weitere Informationen finden Sie unter the section called “Integration von Hardware-Sicherheit”.

  > **Note**
  >
  > Derzeit istAWS IoT Greengrassunterstützt nur diePKCS #1 v1.5Auffüllmechanismus zur Verschlüsselung und Entschlüsselung lokaler Geheimnisse bei Verwendung hardwarebasierter privater Schlüssel. Wenn Sie die vom Hersteller bereitgestellten Anweisungen zum manuellen Generieren hardwarebasierter privater Schlüssel befolgen, wählen Sie unbedingt PKCS #1 v1.5.AWS IoT Greengrassunterstützt nicht Optimal Asymmetric Encryption Padding (OAEP).

- 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.

DieAWS IoT Greengrasscore ist die einzige Entität, die Zugriff auf den privaten Schlüssel hat. Greengrass-Konnektoren oder Lambda-Funktionen, die einer Secret-Ressource zugeordnet sind, erhalten den Wert des Secrets vom Kern.

Voraussetzungen

Dies sind die Anforderungen für den lokalen Secret-Support:

- Sie müssen die verwendenAWS IoT GreengrassCore v1.7 oder höher.
- Um die Werte lokaler Geheimnisse zu erhalten, müssen Ihre benutzerdefinierten Lambda-Funktionen verwendenAWS IoT GreengrassCore-SDK v1.3.0 oder höher.
- Der private Schlüssel, der für die Verschlüsselung lokaler Secrets verwendet wird, muss in der Greengrass-Konfigurationsdatei angegeben sein. Standardmäßig verwendet AWS IoT Greengrass 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**
  >
  > Derzeit istAWS IoT Greengrassunterstützt nur diePKCS #1 v1.5Auffüllmechanismus zur Verschlüsselung und Entschlüsselung lokaler Geheimnisse bei Verwendung hardwarebasierter privater Schlüssel. Wenn Sie die vom Hersteller bereitgestellten Anweisungen zum manuellen Generieren hardwarebasierter privater Schlüssel befolgen, wählen Sie unbedingt PKCS #1 v1.5.AWS IoT Greengrassunterstützt nicht Optimal Asymmetric Encryption Padding (OAEP).

- AWS IoT Greengrass muss die Berechtigung erteilt werden, um Ihre geheimen Werte empfangen zu können. Dies ermöglicht es AWS IoT Greengrass, die Werte während der Bereitstellung der Gruppe abzurufen. Wenn Sie die standardmäßige Greengrass-Servicerolle verwenden, hat AWS IoT Greengrass bereits Zugriff auf Secrets mit Namen, die mit greengrass- beginnen. Informationen zum Anpassen des Zugriffs finden Sie unter the section called “Gewähren des Zugriffs auf Secret-Werte für AWS IoT Greengrass”.

  > **Note**
  >
  > Wir empfehlen Ihnen, diese Namenskonvention zu verwenden, um die Secrets zu identifizieren, auf die AWS IoT Greengrass zugreifen darf, auch wenn Sie die Berechtigungen anpassen. Die Konsole verwendet verschiedene Berechtigungen, um Ihre Secrets zu lesen. Dadurch können Sie in der Konsole Secrets auswählen, für die AWS IoT Greengrass keine Berechtigung zum Abrufen besitzt. 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 Informationen zu privaten Schlüsseln, die auf derAWS IoT GreengrassCore, siehethe section called “Sicherheitsprinzipale”aus.

AWS IoT Greengrass unterstützt zwei Arten der Speicherung privater Schlüssel: hardwarebasiert oder dateisystembasiert (Standard). Weitere Informationen finden Sie unter the section called “Secrets-Verschlüsselung”.

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, unter 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. Zum 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. Zum Beispiel:

  ```
  "SecretsManager" : {
    "privateKeyPath" : "pkcs11:object=private-key-label;type=private"
  }
  ```

  Weitere Informationen finden Sie unter the section called “Hardware-Sicherheitskonfiguration”.

  > **Note**
  >
  > Derzeit istAWS IoT Greengrassunterstützt nur diePKCS #1 v1.5Auffüllmechanismus zur Verschlüsselung und Entschlüsselung lokaler Geheimnisse bei Verwendung hardwarebasierter privater Schlüssel. Wenn Sie die vom Hersteller bereitgestellten Anweisungen zum manuellen Generieren hardwarebasierter privater Schlüssel befolgen, wählen Sie unbedingt PKCS #1 v1.5.AWS IoT Greengrassunterstützt nicht Optimal Asymmetric Encryption Padding (OAEP).

Gewähren des Zugriffs auf Secret-Werte für AWS IoT Greengrass

In diesem Verfahren fügen Sie der Greengrass-Servicerolle eine Inline-Richtlinie hinzu, die es AWS IoT Greengrass ermöglicht, die Werte Ihrer Secrets zu erhalten.

Befolgen Sie dieses Verfahren nur, wenn Sie AWS IoT Greengrass benutzerdefinierte Berechtigungen für Ihre Secrets erteilen möchten oder wenn Ihre Greengrass-Servicerolle nicht die mit `AWSGreengrassResourceAccessRolePolicy` verwaltete Richtlinie enthält. `AWSGreengrassResourceAccessRolePolicy` gewährt Zugriff auf Secrets mit Namen, die 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. Detaillierte Anweisungen finden Sie unterHinzufügen und Entfernen von IAM-RichtlinienimIAM User Guideaus.

   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. Die folgende Richtlinie erlaubt es beispielsweise AWS IoT Greengrass nur die angegebenen Secrets zu lesen.

   ```
   {
       "Version": "2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": [
                   "secretsmanager:GetSecretValue"
               ],
               "Resource": [
                   "arn:aws:secretsmanager:region:account-id:secret:greengrass-
   SecretA-abc",
                   "arn:aws:secretsmanager:region:account-id:secret:greengrass-
   SecretB-xyz"
               ]
           }
       ]
   }
   ```

   > **Note**
   >
   > Wenn Sie einen vom Kunden verwalteten AWS KMS-Schlüssel zur Verschlüsselung von Secrets verwenden, muss Ihre Greengrass-Servicerolle auch die Aktion `kms:Decrypt` erlauben.

Weitere Informationen zu IAM-Richtlinien für Secrets Manager finden Sie unterAuthentifizierung und Zugriffskontrolle fürAWS Secrets ManagerundAktionen, Ressourcen und Kontextschlüssel, die Sie in einer IAM-Richtlinie oder Secret-Richtlinie für verwenden könnenAWS Secrets ManagerimAWS Secrets Manager-Benutzerhandbuchaus.

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 Secret-Ressourcen zum Integrieren von Secrets aus AWS Secrets Manager in einer Greengrass-Gruppe. 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 “Secrets-Verschlüsselung”.

Sie verwenden Secrets Manager, um die Cloud-Versionen Ihrer Secrets zu erstellen und zu verwalten. Verwenden Sie AWS IoT Greengrass zum Erstellen, Verwalten und Bereitstellen Ihrer geheimen Ressourcen.

> **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 ManagerBenutzerhandbuch 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 Secret-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 Greengrassprotokolliert 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 ManagerBenutzerhandbuch.

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 ManagerBenutzerhandbuch.

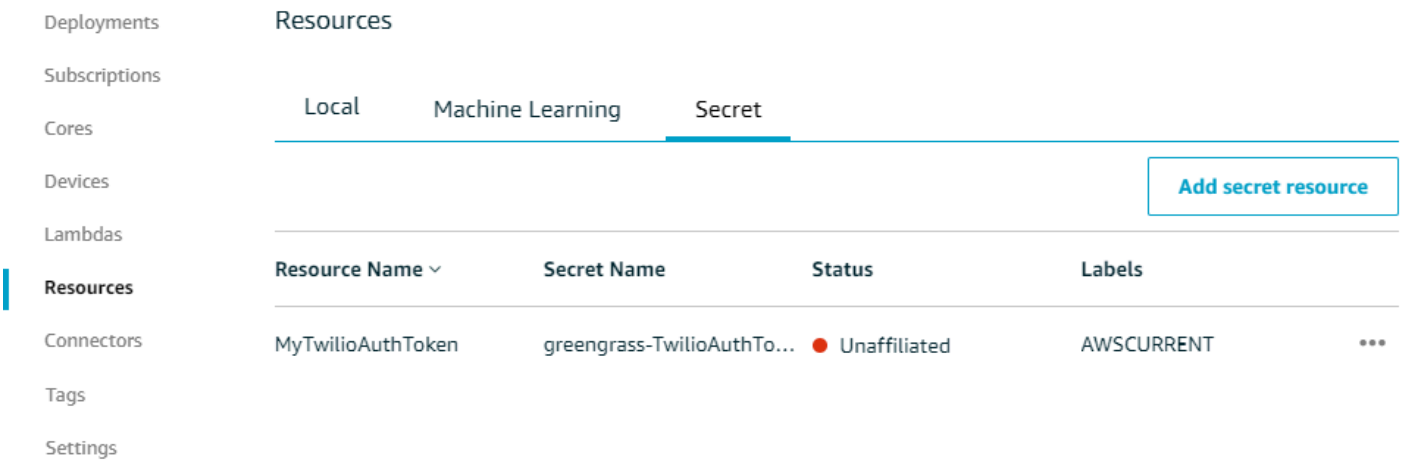
Geheime Ressourcen enthalten immer das `AWSCURRENT` Staging-Label, und sie können optional andere Staging-Labels enthalten, wenn sie von einer Lambda-Funktion oder einem Lambda-

Konnektor benötigt werden. Während der Gruppenbereitstellung ruft AWS IoT Greengrass die Werte der Staging-Beschriftungen ab, die in der Gruppe referenziert sind, und erstellt oder aktualisiert dann die entsprechenden Werte auf dem Kern.

Erstellen und Verwalten von geheimen Ressourcen (Konsole)

Erstellen von geheimen Ressourcen (Konsole)

In der AWS IoT Greengrass Konsole erstellen und verwalten Sie geheime Ressourcen über den Tab Secrets auf der Ressourcenseite der Gruppe. 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)”.



> **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 Secret ist eine Art 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 unterthe 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, wie dazu mit der AWS IoT Greengrass-CLI vorzugehen ist, finden Sie unter the section called “Erste Schritte mit Konnektoren (CLI)”.

Weitere Informationen über die Anforderungen einzelner Konnektoren finden Sie unter the section called “AWSVon bereitgestellte Greengrass-Konnektoren”.

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)”
- Dazu stellen Sie in der AWS IoT Greengrass-API die folgenden Informationen in der Eigenschaft `ResourceAccessPolicies` zur Verfügung.
  - `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 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. Beispiele:

```
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.

In diesem Tutorial wird gezeigt, wie man mit der AWS Management Console eine geheime Ressource zu einer Greengrass-Gruppe hinzufügt. Eine geheime Ressource ist ein Verweis auf ein Secret aus dem 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 Secret in der AWS Secrets Manager-Konsole. 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 die Schritte zur Installation der AWS IoT Greengrass-Core Software.
- AWS IoT Greengrass muss für die Unterstützung lokaler Secrets konfiguriert sein. 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 erstellen Sie mithilfe der AWS Secrets Manager-Konsole ein Secret.

1. Melden Sie sich an der AWS Secrets Manager-Konsole an.

   > **Note**
   >
   > Weitere Informationen zu diesem Vorgang finden Sie unter Schritt 1: Erstellen und Speichern Ihres Geheimnisses 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. 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 daher 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 GreengrassCore 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 GreengrassCore 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. Für dieses Beispiel-Secret gibt AWS IoT Greengrass das Schlüssel-Werte-Paar 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 Management Console Services und öffnen Sie die AWS Lambda-Konsole.

b. Wählen Sie Funktion erstellen und dann Author from scratch.

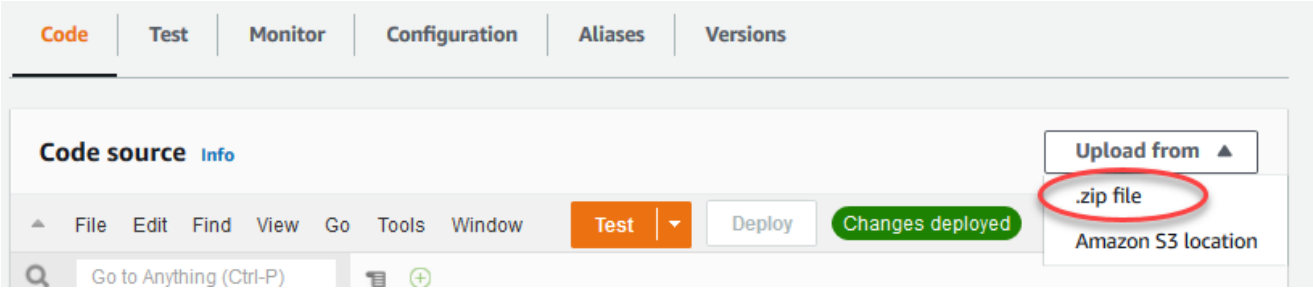
c. Verwenden Sie im Abschnitt Basic information (Basisinformationen) folgende Werte:

- 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 vonAWS 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 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.

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 Greengrassunterstützt 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 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 unterthe section called “Festlegen 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 unterthe section called “Ausführen einer Lambda-Funktion als Root”.

- 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 Auswahl 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 MyTestSecretund wählen Sie dann Associate 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 AWS IoT zu senden.

1. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Abonnements und dann Abonnement hinzufügen aus.
2. Erstellen Sie ein Abonnement, das es AWS IoT ermöglicht, Nachrichten an die Funktion zu veröffentlichen.

   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 in dem 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 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.

> **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.

Property (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.

Property (Eigenschaft)	Wert
Topic	secrets/input

Property (Eigenschaft)	Wert
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

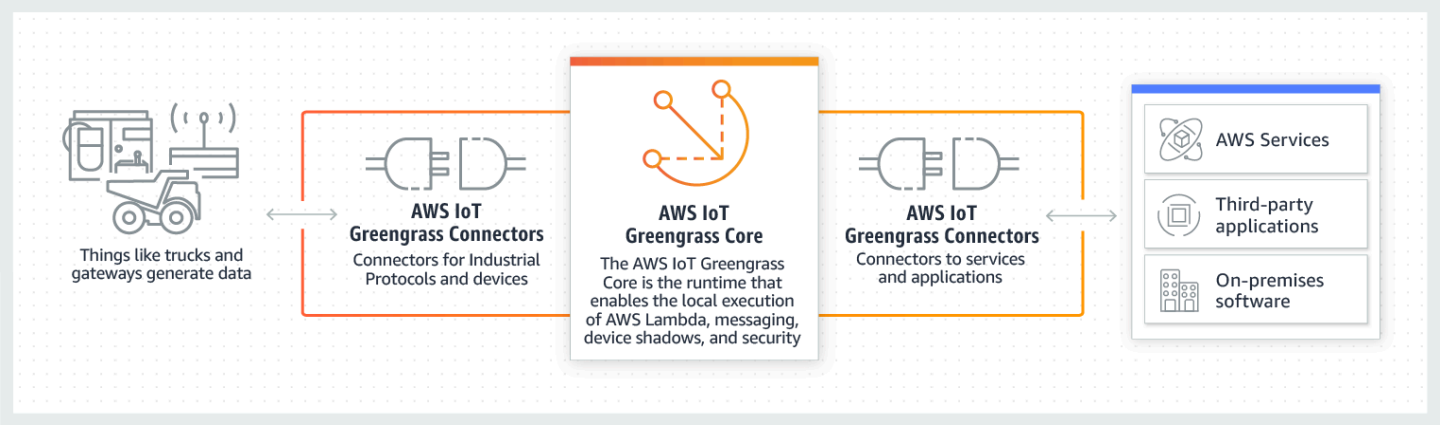
- Bereitstellen von Secrets für den Core

Integrieren von Services und Protokollen mit Greengrass-Konnektoren

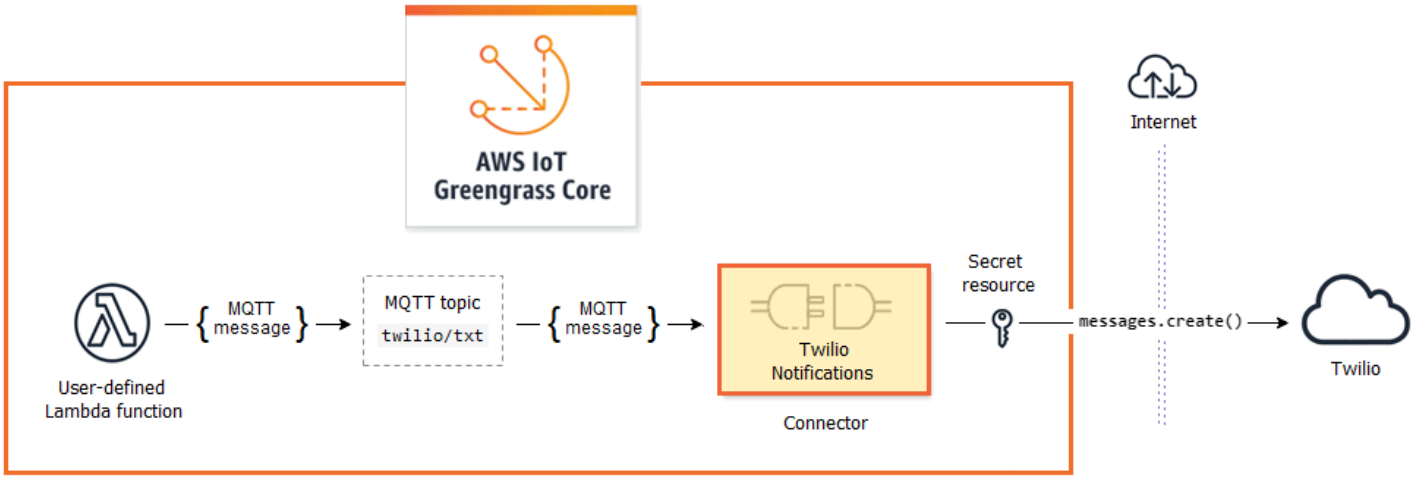
Diese Funktion ist verfügbar fürAWS IoT GreengrassCore v1.7 und höher.

Konnektoren inAWS IoT Greengrasssind vorgefertigte Module, die Interaktion mit der lokalen Infrastruktur effizienter machen, Geräteprotokolle,AWSund andere Cloud-Services. Durch die Verwendung von Konnektoren können Sie weniger Zeit mit dem Erlernen neuer Protokolle und APIs verbringen und mehr Zeit für diefolgerichtigdas ist wichtig für Ihr Unternehmen.

Das folgende Diagramm zeigt, an welcher Stelle Konnektoren in dieAWS IoT Greengrass Landschaftaus.



Viele Konnektoren verwenden MQTT-Nachrichten für die Kommunikation mit Client-Geräten und Greengrass Lambda--Funktionen in der Gruppe oder mitAWS IoTund der lokale Shadow-Service. Im folgenden Beispiel empfängt der Twilio-Benachrichtigungs-Connector MQTT-Nachrichten von einer benutzerdefinierten Lambda-Funktion und verwendet eine lokale Referenz eines Secrets vonAWS Secrets Managerund ruft die 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 Einzweck-Geräte zu erstellen. Durch die Verwendung von Connectors können Sie:

- Implementieren einer wiederverwendbaren Geschäftslogik.
- Interagieren mit der Cloud und lokalen Services, einschließlichAWSund Drittanbieterdienste.
- Erfassen und Verarbeiten von Gerätedaten.
- Aktivieren von device-to-device ruft über MQTT-Themenabonnements und benutzerdefinierte Lambda--Funktionen auf.

AWSbietet eine Reihe von Greengrass-Konnektoren, die Interaktion mit gängigen Services 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 “AWSVon bereitgestellte Greengrass-Konnektoren”.

Voraussetzungen

Beachten Sie die folgenden Punkte, um Konnektoren zu verwenden:

- Jeder Connector, den Sie verwenden, hat Anforderungen, die Sie erfüllen müssen. Diese Anforderungen können die Mindestversion der AWS IoT Greengrass Core-Software,

Gerätevoraussetzungen, erforderliche Berechtigungen und Limits umfassen. Weitere Informationen finden Sie unter the section called “AWSVon bereitgestellte Greengrass-Konnektoren”.

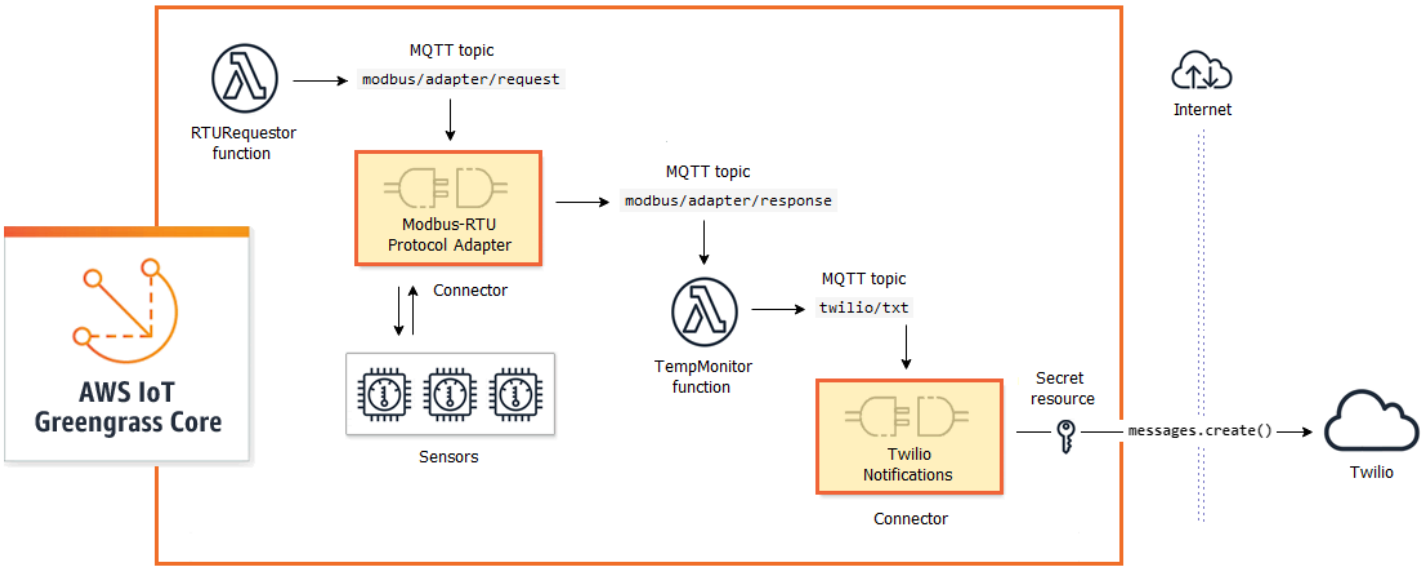
- Eine Greengrass-Gruppe kann nur eine konfigurierte Instance eines bestimmten Konnektors enthalten. Sie können die Instance jedoch in mehreren Abonnements verwenden. Weitere Informationen finden Sie unter the section called “Konfigurationsparameter”.
- Wenn die Standardcontainerisierung für die Greengrass-Gruppe 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 “AWSVon bereitgestellte Greengrass-Konnektoren”.

Verwenden von Greengrass-Konnektoren

Ein Konnektorist eine Art Gruppenkomponente. Wie bei anderen Gruppenkomponenten, z. B. Client-Geräte und benutzerdefinierte Lambda-Funktionen, fügen Sie Konnektoren zu Gruppen hinzu, konfigurieren ihre Einstellungen und stellen sie für bereitAWS IoT GreengrassKern. Konnektoren werden in der Core-Umgebung ausgeführt.

Sie können einige Konnektoren als einfache, eigenständige Anwendungen bereitstellen. Beispielsweise liest der Device Defender-Connector Systemmetriken vom Core-Gerät und sendet sie anAWS IoT Device Defenderzur Analyse.

Sie können weitere Konnektoren als Bausteine in größeren Lösungen hinzufügen. Die folgende Beispiellösung verwendet den Modbus-RTU-Protokolladapter-Konnektor zur Verarbeitung von Meldungen von Sensoren und den Twilio-Meldungs-Konnektor zur Initiierung von Twilio



Lösungen beinhalten oft benutzerdefinierte Lambda--Funktionen, die sich neben den Konnektoren befinden und die Daten verarbeiten, die der -Konnektor sendet oder empfängt. In diesem Beispiel fehlt bei der Aktion TempMonitorempfängt Daten vom Modbus-RTU-Protokolladapter, führt einige Geschäftslogik aus und sendet dann Daten an Twilio-Benachrichtigungen.

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 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 “AWSVon bereitgestellte Greengrass-Konnektoren”.
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-Anforderungen zu senden und Twilio-Benachrichtigungen einzuleiten 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 für die Authentifizierung bei Services und Anwendungen verwendet werden. Standardmäßig kann AWS IoT Greengrass auf Secrets mit Namen zugreifen, die 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 Konnektors 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. Beispiel: Sie können Abonnements erstellen, die es vielen Geräten ermöglichen, Daten an den Kinesis Firehose-Connector zu senden.

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 aus 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 Host`/proc`und Parameter für Twilio-Benachrichtigungen ermöglichen den Zugriff auf einen lokal gespeicherten 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 dieKonnektorenund wählen Sie im Konnektor-Kontextmenü aufBearbeitenaus.

  > **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 neue Konfiguration definiert.

DieAWS IoT GreengrassDie API verwendet Versionen, um Gruppen zu verwalten. Versionen sind unveränderlich. Um Gruppenkomponenten hinzuzufügen oder zu ändern — z. B. die Client-Geräte, -Funktionen und Ressourcen der Gruppe — müssen Sie Versionen neuer oder aktualisierter Komponenten erstellen. Anschließend erstellen und stellen Sie eine Gruppenversion bereit, 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 dem Konnektor den Austausch von Daten mit Lambda-Funktionen, Client-Geräten, anderen Konnektoren in der Greengrass-Gruppe, ermöglichen,AWS IoTund der lokale Shadow-Service. 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 “AWSVon bereitgestellte Greengrass-Konnektoren”.

Eingabethemen

Die meisten Konnektoren empfangen Eingabedaten zu MQTT-Themen. Einige Konnektoren abonnieren mehrere Themen für Eingabedaten. Beispielsweise unterstützt der Serial Stream-Connector zwei Themen:

- `serial/+/read/#`
- `serial/+/write/#`

Für diesen Konnektor werden Lese- und Schreibanforderungen 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 das Input- oder Output-Thema finden Sie in der Dokumentation für den Konnektor Weitere Informationen finden Sie unter the section called “AWSVon bereitgestellte Greengrass-Konnektoren”.

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 ausführenAWS IoT Greengrassin einem Docker-Container oder auf älteren Linux-Kernel ohne Cgroups. 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 “AWSVon bereitgestellte Greengrass-Konnektoren”. 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 derAWS IoT-Konsole können Sie nach neuen Versionen für die Konnektoren in Ihrer Greengrass-Gruppe suchen.

1. In derAWS IoTNavigationsbereich der -Konsole unterVerwalten, erweiternGreengrass Geräteund wählen Sie dannGruppen (V1)aus.
2. UnderGreengrass Gruppenwählen Sie Ihre Gruppe aus.

3. Wählen Sie Connectors (Konnektoren), um die Konnektoren in der Gruppe anzuzeigen.

   Wenn der Connector eine neue Version hat, wird einVerfügbarwird in derUpgradecolumn.

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 Konnektor über die AWS IoT Greengrass-API zu aktualisieren, erstellen Sie eine Gruppenversion, die den aktualisierten Konnektor und die Abonnements enthält, und stellen Sie sie bereit. Gehen Sie genauso vor wie beim Hinzufügen eines Konnektors zu einer Gruppe. Weitere Informationen zur Verwendung desAWS CLIInformationen zum Konfigurieren und Bereitstellen eines Beispielkonnektors für Twilio-Benachrichtigungen findenthe section called “Erste Schritte mit Konnektoren (CLI)”aus.

Protokollieren für Konnektoren

Greengrass-Konnektoren enthalten Lambda--Funktionen, die Ereignisse und Fehler in Greengrass-Protokolle schreiben. Abhängig von Ihren Gruppeneinstellungen werden Protokolle in CloudWatch Protokolle, das lokale Dateisystem oder beides. Protokolle von Konnektoren enthalten den ARN der entsprechenden Funktion. Nachfolgend finden Sie ein Beispiel eines ARN von dem 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 Informationen zur Greengrass-Protokollierung finden Sie unterthe section called "Überwachen mit AWS IoT Greengrass-Protokollen"aus.

AWSVon bereitgestellte Greengrass-Konnektoren

AWS bietet die folgenden Connectors, die gängige 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 in Amazon CloudWatch.	• Python 3.8* • Python 3.7 • Python 2.7	Ja
Device Defender	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
IoT-Analysen	Sendet Daten von Geräten und Sensoren an AWS IoT Analytics.	• Python 3.8*	Ja

Konnektor	Beschreibung	Unterstützte Lambda-Laufzeiten	Unterstützt No Container (Kein Container)-Modus
		• Python 3.7 • Python 2.7	
IoT Ethernet IP Protocol Adapter	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-Bereitstellungsdatenströme.	• Python 3.8* • Python 3.7 • Python 2.7	Ja
ML-Feedback	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
Bol-RTU-Protokolladapter	Sendet Anfragen an Modbus RTU-Geräte.	• Python 3.8* • Python 3.7 • Python 2.7	Nein

Konnektor	Beschreibung	Unterstützte Lambda-Laufzeiten	Unterstützt No Container (Kein Container)-Modus
Bol-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 in ServiceNow MetricBase.	• Python 3.8* • Python 3.7 • Python 2.7	Ja
SNS	Sendet Nachrichten an ein Amazon SNS-Thema.	• Python 3.8* • Python 3.7 • Python 2.7	Ja
Splunk-Integration	Veröffentlicht Daten in Splunk HEC.	• Python 3.8* • Python 3.7 • Python 2.7	Ja
Twilio-Benachrichtigungen	Initiiert einen Twilio-Text oder eine 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 Connector-spezifischen 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 Laufzeit-Supportrichtlinie im AWS Lambda -Entwicklerhandbuch.

CloudWatch Metrik-Konnektor

Der CloudWatch Metrics Connector veröffentlicht benutzerdefinierte Metriken von Greengrass-Geräten in Amazon CloudWatch. Der Konnektor 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 Amazon- CloudWatch Metriken im Amazon- CloudWatch Benutzerhandbuch.

Dieser Konnektor empfängt Metrikdaten wie MQTT-Nachrichten. Der Konnektor stapelt Metriken, die sich im selben Namespace befinden, und veröffentlicht sie CloudWatch in regelmäßigen Abständen in .

Dieser Konnektor hat die folgenden Versionen.

Version	ARN
5	`arn:aws:greengrass:  region::/connectors/CloudWatchMetrics/versions/5`
4	`arn:aws:greengrass:  region::/connectors/CloudWatchMetrics/versions/4`

Version	ARN
3	`arn:aws:greengrass:region::/connectors/CloudWatchMetrics/versions/3`
2	`arn:aws:greengrass:region::/connectors/CloudWatchMetrics/versions/2`
1	`arn:aws:greengrass:region::/connectors/CloudWatchMetrics/versions/1`

Informationen über Versionsänderungen finden Sie im Änderungsprotokoll.

Voraussetzungen

Dieser Konnektor hat die folgenden Anforderungen:

Version 3 - 5

- AWS IoT Greengrass Core-Software v1.9.3 oder höher.
- Python Version 3.7 oder 3.8 ist auf dem Core-Gerät installiert und der PATH-Umgebungsvariablen 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, die so konfiguriert ist, dass sie die `cloudwatch:PutMetricData` Aktion zulässt, wie in der folgenden Beispielrichtlinie AWS Identity and Access Management (IAM) gezeigt.

```
{
    "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. Weitere Informationen finden Sie unter the section called “Verwalten der Gruppenrolle (Konsole)” oder the section called “Verwalten der Gruppenrolle (CLI)”.

  Weitere Informationen zu CloudWatch Berechtigungen finden Sie in der Amazon- CloudWatch Berechtigungsreferenz im IAM-Benutzerhandbuch.

Versions 1 - 2

- AWS IoT Greengrass Core-Software v1.7 oder höher.
- Python Version 2.7 wurde auf dem Core-Gerät installiert und der PATH-Umgebungsvariablen hinzugefügt.
- Die Greengrass-Gruppenrolle, die so konfiguriert ist, dass sie die `cloudwatch:PutMetricData` Aktion zulässt, wie im folgenden Beispiel einer AWS Identity and Access Management (IAM)-Richtlinie gezeigt.

```
{
    "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. Weitere Informationen finden Sie unter the section called “Verwalten der Gruppenrolle (Konsole)” oder the section called “Verwalten der Gruppenrolle (CLI)”.

Weitere Informationen zu CloudWatch Berechtigungen finden Sie in der Amazon- CloudWatch Berechtigungsreferenz im IAM-Benutzerhandbuch.

Connector-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 Konnektor veröffentlicht in , CloudWatch nachdem er 20 Metriken im selben Namespace oder nach dem angegebenen Intervall erhalten hat.

> **Note**
>
> Der Konnektor garantiert nicht die Reihenfolge der veröffentlichten Ereignisse.

Anzeigename in der AWS IoTKonsole: Publish interval

Erforderlich: `true`

Typ: `string`

Zulässige Werte: `0 - 900`

Gültiges Muster: `[0-9]|[1-9]\d|[1-9]\d\d|900`

`PublishRegion`

Die AWS-Region, in der CloudWatch Metriken veröffentlicht werden sollen. Dieser Wert überschreibt die standardmäßige Greengrass-Metrik-Region. Er ist nur bei der Veröffentlichung von regionsübergreifenden Metriken erforderlich.

Anzeigename in der AWS IoTKonsole: 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 an beizubehaltenden Metriken

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `^([2-9]\d{3}|[1-9]\d{4,})$`

`IsolationMode`

Der Containerisierungsmodus für diesen Konnektor. Der Standardwert ist `GreengrassContainer`. Hierbei wird der Konnektor in einer isolierten Laufzeitumgebung innerhalb des AWS IoT Greengrass-Containers ausgeführt.

> **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 Konnektor veröffentlicht in , CloudWatch nachdem er 20 Metriken im selben Namespace oder nach dem angegebenen Intervall erhalten hat.

> **Note**
>
> Der Konnektor garantiert nicht die Reihenfolge der veröffentlichten Ereignisse.

Anzeigename in der AWS IoTKonsole: Publish interval

Erforderlich: `true`

Typ: `string`

Zulässige Werte: `0 - 900`

Gültiges Muster: `[0-9]|[1-9]\d|[1-9]\d\d|900`

`PublishRegion`

Die AWS-Region, in der CloudWatch Metriken veröffentlicht werden sollen. Dieser Wert überschreibt die standardmäßige Greengrass-Metrik-Region. Er ist nur bei der Veröffentlichung von regionsübergreifenden Metriken erforderlich.

Anzeigename in der AWS IoTKonsole: 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 an 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 Anfangsversion, die den CloudWatch Metrics-Konnektor 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 einen Connector über die Seite Connectors der Gruppe hinzufügen. Weitere Informationen finden Sie unter the section called “Beginnen Sie mit Konnektoren (Konsole)”.

Eingabedaten

Dieser Konnektor akzeptiert Metriken zu einem MQTT-Thema und veröffentlicht die Metriken in CloudWatch. Eingabenachrichten müssen im JSON-Format vorliegen.

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 `PutMetricData` API entsprechen. Es werden nur die Eigenschaften `namespace`, `metricData.metricName` und `metricData.value` benötigt.

Erforderlich: `true`

Typ: `object`, der die folgenden Eigenschaften enthält:

`namespace`

Der benutzerdefinierte Namespace für die Metrikdaten in dieser Anforderung. CloudWatch verwendet Namespaces als Container für Metrikdatenpunkte.

> **Note**
>
> Sie können keinen Namespace angeben, der mit der reservierten Zeichenfolge beginntAWS/.

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `[^:].*`

`metricData`

Die Daten für die Metrik.

Erforderlich: `true`

Typ: `object`, der die folgenden Eigenschaften enthält:

`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 Namen `coreName`, wobei der Wert der Name des Kerns ist.

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 Metrikdaten empfangen wurden, ausgedrückt als Anzahl der Sekunden seit `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 Konnektors verwenden, empfehlen wir Ihnen, 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 Werte ab, die zu klein oder zu groß sind. 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 Limits gelten für Metriken bei Verwendung dieses Konnektors. 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 Limits im Amazon- CloudWatch 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 enthält 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 Konnektor einen wiederholbaren Fehler erkennt (z. B. Verbindungsfehler), wiederholt er die Veröffentlichung im nächsten Batch.

Verwendungsbeispiel

Führen Sie die folgenden allgemeinen Schritte aus, 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. Weitere Informationen finden Sie unter 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 Konnektor sendet.

   Speichern Sie den Beispielcode als PY-Datei. Laden Sie das AWS IoT Greengrass Core SDK for 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, das Sie in hochladenAWS Lambda.

   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 nach 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 Eingabethemafilter.
   - 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 IoTKonsole anzuzeigen.

4. Stellen Sie die Gruppe bereit.

5. Abonnieren Sie in der -AWS IoTKonsole auf der Seite Test das Ausgabedatenthema, um Statusmeldungen vom Konnektor anzuzeigen. Die Lambda-Beispielfunktion ist langlebig und beginnt sofort nach der Bereitstellung der Gruppe Nachrichten zu senden.

   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-Konnektor enthält die folgende Software/Lizenzierung von Drittanbietern:

- AWS SDK for 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 Konnektor wurde gemäß der Greengrass Core Software License Agreement veröffentlicht.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in jeder Version des Konnektors beschrieben.

Version	Änderungen
5	Korrektur, um Unterstützung für doppelte Zeitstempel in Eingabedaten hinzuzufügen.
4	Der Parameter `IsolationMode` wurde hinzugefügt, um den Containerisierungsmodus für den Konnektor zu konfigurieren.
3	Die Lambda-Laufzeit wurde auf Python 3.7 aktualisiert, wodurch sich die Laufzeitanforderung 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)”
- Verwenden von Amazon- CloudWatch Metriken im Amazon CloudWatch -Benutzerhandbuch
- PutMetricData in der Amazon CloudWatch -API-Referenz zu

Device Defender-An

Der Device DefenderAnschlussbenachrichtigt Administratoren über Änderungen im Status eines Greengrass Core-Geräts. Dies kann helfen, ungewöhnliches Verhalten zu erkennen, das auf ein gefährdetes Gerät hinweisen könnte.

Dieser Konnektor liest Systemmetriken aus dem Verzeichnis `/proc` auf dem Core-Gerät und veröffentlicht die Metriken dann in AWS IoT Device Defender. Details zur Metrik-Berichterstattung finden Sie unterSpezifikationen für GerätemetrikenimAWS IoTEntwicklerhandbuchaus.

Dieser Konnektor hat die folgenden Versionen.

Version	ARN
3	`arn:aws:greengrass:  region::/connectors/DeviceDefender/versions/3`
2	`arn:aws:greengrass:  region::/connectors/DeviceDefender/versions/2`
1	`arn:aws:greengrass:  region::/connectors/DeviceDefender/versions/1`

Informationen über Versionsänderungen finden Sie im Änderungsprotokoll.

Voraussetzungen

Dieser Konnektor hat die folgenden Anforderungen:

Version 3

- AWS IoT Greengrass Core-Software v1.9.3 oder höher.
- PythonVersion 3.7 oder 3.8 auf dem Core-Gerät installiert und der Umgebungsvariablen PATH hinzugefügt.

> **Note**
>
> Um Python 3.8 zu verwenden, führen Sie den folgenden Befehl aus, um einen symbolischen Link vom Standardinstallationsordner für Python 3.8 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 ist konfiguriert, um die Detect-Funktion zu verwenden, damit Verstöße verfolgt werden. Weitere Informationen finden Sie unterErkennenimAWS IoTEntwicklerhandbuchaus.
- EINLokales Volume-RessourceIn der Greengrass-Gruppe, die auf`/proc`Verzeichnis. 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 GreengrassCore-Software v1.7 oder höher.
- PythonVersion 2.7 auf dem Core-Gerät installiert und der Umgebungsvariablen PATH hinzugefügt.
- AWS IoT Device Defender ist konfiguriert, um die Detect-Funktion zu verwenden, damit Verstöße verfolgt werden. Weitere Informationen finden Sie unterErkennenimAWS IoTEntwicklerhandbuchaus.
- EINLokales Volume-RessourceIn der Greengrass-Gruppe, die auf`/proc`Verzeichnis. 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.

Connector-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).

Anzeigename imAWS IoT-Konsole Metrik-Berichterstellungsinterv

Erforderlich`true`

Typ: `string`

Gültiges Pattern:`^[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.

Anzeigename imAWS IoT-Konsole Resource für/proc Verzeichnis

Erforderlich`true`

Typ: `string`

Gültiges Pattern:`[a-zA-Z0-9_-]+`

`ProcDestinationPath`

Der Ziel-Pfad der Volume-Ressource `/proc`.

Anzeigename imAWS IoT-Konsole Zielpfad der /proc Ressource

Erforderlich`true`

Typ: `string`

Gültiges Pattern:`\/[a-zA-Z0-9_-]+`

Beispiel für das Erstellen eines Konnektors (AWS CLI)

Der folgende CLI-Befehl erstellt eine`ConnectorDefinition`mit einer Initialversion, die den Device Defender-Konnektor 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 Anschluss hat einelangdauerndesLebenszyklus.

In derAWS IoT Greengrass-Konsole können Sie einen Konnektor aus der Gruppe hinzufügenKonnektorenangezeigten. Weitere Informationen finden Sie unter the section called “Beginnen Sie mit Konnektoren (Konsole)”.

Eingabedaten

Dieser Konnektor akzeptiert keine MQTT-Nachrichten als Eingabedaten.

Ausgabedaten

Dieser Konnektor veröffentlicht Metriken in 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. Der Konnektor ersetzt den +-Platzhalter durch den Gerätenamen (z. B. `$aws/things/thing-name/defender/metrics/json`).

Beispielausgabe

Details zur Metrik-Berichterstattung finden Sie unterSpezifikationen für GerätemetrikenimAWS IoTEntwicklerhandbuchaus.

```
{
    "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 Konnektor wird unter derLizenzvereinbarung für die Greengrass Core-Softwareaus.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in jeder Version des -Konnektors beschrieben.

Version	Änderungen
3	Die Lambda-Laufzeitumgebung wurde auf Python 3.7 aktualisiert, wodurch sich die Laufzeitanforderung ändert.
2	Beheben, um übermäßige Protokollierung zu reduzieren.
1	Erstversion.

Eine Greengrass-Gruppe kann jeweils nur eine Version des -Konnektors 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 DefenderimAWS IoTEntwicklerhandbuch

Docker-Anwendungsbereitstellungs-Konnektor

Der Greengrass Docker-Anwendungsbereitstellungs-Connector erleichtert die Ausführung Ihrer Docker-Images auf einem -AWS IoT GreengrassCore. 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 Konnektor kann auf Docker-Images zugreifen, die in Docker-Container-Registrys wie Amazon Elastic Container Registry (Amazon ECR), Docker Hub und privaten vertrauenswürdigen Docker-Registrys gespeichert sind.

Nach der Bereitstellung der Greengrass-Gruppe ruft der Konnektor die neuesten Images ab und startet die Docker-Container. Es führt den `docker-compose up` Befehl `docker-compose pull` und aus. 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 Webservers 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 -Kern ausführen, der Anfragen von Lambda-Funktionen an einen Webservice 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 Konnektor hat die folgenden Versionen.

Version	ARN
7	`arn:aws:greengrass:  region::/connectors/DockerApplicationDeployment/versions/7`
6	`arn:aws:greengrass:  region::/connectors/DockerApplicationDeployment/versions/6`

Version	ARN
5	`arn:aws:greengrass:region::/connectors/DockerApplicationDeployment/versions/5`
4	`arn:aws:greengrass:region::/connectors/DockerApplicationDeployment/versions/4`
3	`arn:aws:greengrass:region::/connectors/DockerApplicationDeployment/versions/3`
2	`arn:aws:greengrass:region::/connectors/DockerApplicationDeployment/versions/2`
1	`arn:aws:greengrass:region::/connectors/DockerApplicationDeployment/versions/1`

Informationen über Versionsänderungen finden Sie im Änderungsprotokoll.

Voraussetzungen

Dieser Konnektor hat die folgenden Anforderungen:

- AWS IoT Greengrass Core-Software v1.10 oder höher.

> **Note**
>
> Dieser Konnektor wird auf - OpenWrt Verteilungen nicht unterstützt.

- Python Version 3.7 oder 3.8 ist auf dem Core-Gerät installiert und der PATH-Umgebungsvariablen 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 zum Installieren von Docker auf Amazon Linux-Distributionen finden Sie unter Docker-Grundlagen für Amazon ECS im Amazon Elastic Container Service-Entwicklerhandbuch.
- 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`), die in Amazon Simple Storage Service (Amazon S3) gespeichert ist. 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.

```
{
    "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 versionsfähig ist, muss die Rolle so konfiguriert werden, dass auch die `s3:GetObjectVersion` Aktion zugelassen wird. Weitere Informationen finden Sie unter Verwenden der Versionsverwaltung im Benutzerhandbuch für Amazon Simple Storage Service.

Für die Gruppenrollenanforderung müssen Sie die Rolle so konfigurieren, dass die erforderlichen Berechtigungen erteilt werden, und sicherstellen, dass die Rolle der Gruppe hinzugefügt wurde. Weitere Informationen finden Sie unter 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 Folgendes zugelassen wird:
  - `ecr:GetDownloadUrlForLayer` - und -`ecr:BatchGetImage`Aktionen auf Ihren Amazon-ECR-Repositorys, die die Docker-Images enthalten.
  - Die `ecr:GetAuthorizationToken`-Aktion für die Ressourcen.

  Repositorys müssen sich im selben AWS-Konto und im selben AWS-Region wie der Konnektor 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.

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "AllowGetEcrRepositories",
            "Effect": "Allow",
            "Action": [
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": [
                "arn:aws:ecr:region:account-id: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. Weitere Informationen finden Sie unter 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 AWS Marketplace-Containerprodukte abonniert haben. Weitere Informationen finden Sie unter Suchen und Abonnieren von Containerprodukten im AWS Marketplace-Abonnentenhandbuch.
  - AWS IoT Greengrass muss so konfiguriert sein, dass lokale Secrets unterstützt werden, wie unter Secrets-Anforderungen beschrieben. Der Konnektor verwendet diese Funktion nur, um Ihre Secrets von abzurufenAWS Secrets Manager, nicht, um sie zu speichern.
  - Sie müssen ein Secret in Secrets Manager für jede AWS Marketplace Registrierung erstellen, die ein Docker-Image speichert, auf das in Ihrer Compose-Datei verwiesen wird. 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 Registrierungen als Amazon ECR verweist, z. B. Docker Hub, hat der Konnektor auch die folgenden Anforderungen:
  - AWS IoT Greengrass muss so konfiguriert sein, dass lokale Secrets unterstützt werden, wie unter Secrets-Anforderungen beschrieben. Der Konnektor verwendet diese Funktion nur, um Ihre Secrets von abzurufenAWS Secrets Manager, nicht, um sie zu speichern.
  - Sie müssen ein Secret in Secrets Manager für jedes private Repository erstellen, das ein Docker-Image speichert, auf das in Ihrer Compose-Datei verwiesen wird. 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 gespeicherte Docker-Images in Amazon ECR erteilen Sie die Berechtigung, Ihr Autorisierungstoken in der Greengrass-Gruppenrolle abzurufen. Weitere Informationen finden Sie unter the section called “Voraussetzungen”.

Für Docker-Abbilder, die in anderen privaten Repositorys oder Registrys gespeichert sind, müssen Sie ein Secret in AWS Secrets Manager erstellen, um Ihre Anmeldeinformationen zu speichern. Dies schließt Docker-Abbilder ein, die Sie in AWS Marketplace abonniert haben. Erstellen Sie ein Secret für jedes Repository. Wenn Sie Ihre Secrets in Secrets Manager aktualisieren, werden die Änderungen bei der nächsten Bereitstellung der Gruppe an den Kern weitergegeben.

> **Note**
>
> Secrets Manager ist ein Service, mit dem Sie Ihre Anmeldeinformationen, Schlüssel und andere Secrets sicher in der speichern und verwalten könnenAWS Cloud. Weitere Informationen finden Sie unter Was ist AWS Secrets Manager? im AWS Secrets Manager-Benutzerhandbuch.

Jedes Secret muss die folgenden Schlüssel enthalten:

Schlüssel	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 AWS IoT Greengrass den Zugriff auf ein Secret standardmäßig zu erlauben, muss der Name des Secrets mit greengrass- beginnen. Andernfalls muss Ihre Greengrass-Servicerolle Zugriff gewähren. Weitere Informationen finden Sie unter the section called “Gewähren des Zugriffs auf Secret-Werte für AWS IoT Greengrass”.

So rufen Sie Anmeldeinformationen für Docker-Abbilder aus AWS Marketplace ab

1. Rufen Sie Ihr Passwort für Docker-Images von AWS Marketplace mit dem `aws ecr get-login-password` Befehl ab. Weitere Informationen finden Sie unter get-login-password in der Referenz zum AWS CLI-Befehl.

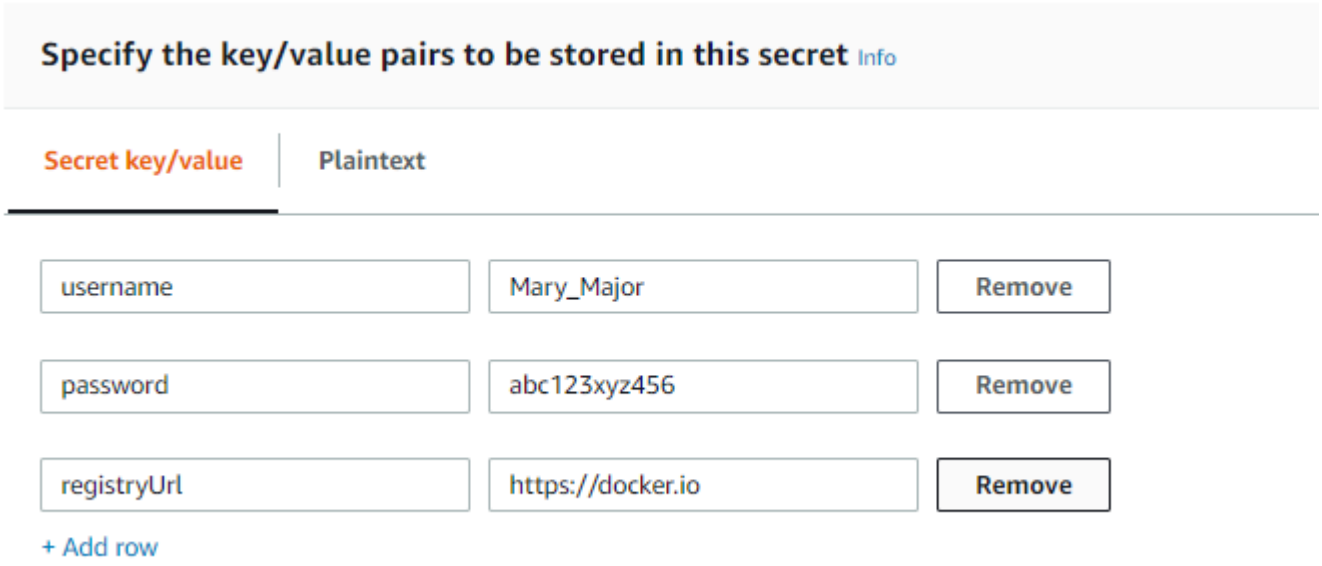
   ```
   aws ecr get-login-password
   ```

2. Rufen Sie die Registrierungs-URL für das Docker-Image ab. Öffnen Sie die -AWS MarketplaceWebsite und öffnen Sie die Startseite des Container-Produkts. 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 Secret für jede AWS Marketplace Registrierung zu erstellen, die Docker-Images speichert, auf die in Ihrer Compose-Datei verwiesen wird.

So erstellen Sie Secrets (Konsole)

Wählen Sie in der AWS Secrets Manager-Konsole Andere Art von Secrets 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 unter Erstellen eines grundlegenden Secrets im AWS Secrets Manager -Benutzerhandbuch.



So erstellen Sie Secrets (CLI)

AWS CLIVerwenden Sie in der den Secrets-Manager-`create-secret`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”.

Parameter

Dieser Konnektor stellt die folgenden Parameter bereit:

Version 7

`DockerComposeFileS3Bucket`

Der Name des S3-Buckets, der Ihre Docker Compose-Datei enthält. Wenn Sie den Bucket erstellen, müssen Sie die Regeln für Bucket-Namen befolgen, die im Benutzerhandbuch für Amazon Simple Storage Service 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 Benutzerhandbuch für Amazon Simple Storage Service.

> **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 von Versioning im Benutzerhandbuch für Amazon Simple Storage Service.

> **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 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 Lambda-Funktionen erlauben, als Root auf dem AWS IoT Greengrass Core ausgeführt zu werden. Weitere Informationen finden Sie unter the section called “Ausführen einer Lambda-Funktion als Root”.

Anzeigename in der AWS IoT Konsole: Docker-Benutzer-ID

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `^[0-9]{1,5}$`

`AWSSecretsArnList`

Die Amazon Resource Names (ARNs) der Secrets in AWS Secrets Manager, die die Anmeldeinformationen enthalten, die für den Zugriff auf Ihre Docker-Abbilder 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: Protokollierungsfrequenz

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 der 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 Bereitstellungsprogramm 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 Konnektor von Docker Deployer verwaltete Docker-Container stoppen soll, wenn CCPC gestoppt wird (CCPC stoppt, wenn eine neue Gruppe bereitgestellt wird, oder der Kernel wird heruntergefahren). Der Standardwert ist `True`.

Anzeigename in der AWS IoT Konsole: Docker bei neuer Bereitstellung stoppen

> **Note**
>
> Wir empfehlen, diesen Parameter auf seinen `True` Standardwert festzulegen. Der Parameter für `False` bewirkt, dass Ihr Docker-Container auch nach dem Beenden des AWS IoT Greengrass Kerns oder dem Starten einer neuen Bereitstellung weiter ausgeführt wird. Wenn Sie diesen Parameter auf setzen`False`, müssen Sie sicherstellen, dass Ihre Docker-Container im Falle einer Änderung oder Ergänzung des `docker-compose` Servicenamens nach Bedarf gewartet werden.
> Weitere Informationen finden Sie in der Dokumentation `docker-compose` zur Compose-Datei.

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `^(true|false)$`

`DockerOfflineMode`

Gibt an, ob die vorhandene Docker-Compose-Datei verwendet werden soll, wenn offline AWS IoT Greengrass startet. 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 beim Erstellen des Buckets darauf, die Regeln für Bucket-Namen zu befolgen, die im Benutzerhandbuch für Amazon Simple Storage Service 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 Benutzerhandbuch für Amazon Simple Storage Service.

> **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 von Versioning im Benutzerhandbuch für Amazon Simple Storage Service.

> **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 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 Lambda-Funktionen erlauben, als Root auf dem AWS IoT Greengrass Core ausgeführt zu werden. Weitere Informationen finden Sie unter the section called “Ausführen einer Lambda-Funktion als Root”.

Anzeigename in der AWS IoT Konsole: Docker-Benutzer-ID

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `^[0-9]{1,5}$`

`AWSSecretsArnList`

Die Amazon Resource Names (ARNs) der Secrets in AWS Secrets Manager, die die Anmeldeinformationen enthalten, die für den Zugriff auf Ihre Docker-Abbilder 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: Protokollierungsfrequenz

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 der 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 Bereitstellungsprogramm 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 Konnektor von Docker Deployer verwaltete Docker-Container stoppen soll, wenn CCPC gestoppt wird (wenn eine neue Gruppenbereitstellung durchgeführt wird oder der Kernel heruntergefahren wird). Der Standardwert ist `True`.

Anzeigename in der AWS IoT Konsole: Docker bei neuer Bereitstellung stoppen

> **Note**
>
> Wir empfehlen, diesen Parameter auf seinen `True` Standardwert festzulegen. Der Parameter für `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 festlegen`False`, müssen Sie sicherstellen, dass Ihre Docker-Container im Falle einer Änderung oder Ergänzung des `docker-compose` Servicenamens nach Bedarf verwaltet werden.
> Weitere Informationen finden Sie in der Dokumentation `docker-compose` zur Compose-Datei.

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `^(true|false)$`

Version 5

`DockerComposeFileS3Bucket`

Der Name des S3-Buckets, der Ihre Docker Compose-Datei enthält. Wenn Sie den Bucket erstellen, müssen Sie die Regeln für Bucket-Namen befolgen, die im Benutzerhandbuch für Amazon Simple Storage Service 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 Benutzerhandbuch für Amazon Simple Storage Service.

> **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 von Versioning im Benutzerhandbuch für Amazon Simple Storage Service.

> **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 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 Lambda-Funktionen erlauben, als Root auf dem AWS IoT Greengrass Core ausgeführt zu werden. Weitere Informationen finden Sie unter the section called “Ausführen einer Lambda-Funktion als Root”.

Anzeigename in der AWS IoT Konsole: Docker-Benutzer-ID

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `^[0-9]{1,5}$`

`AWSSecretsArnList`

Die Amazon Resource Names (ARNs) der Secrets in AWS Secrets Manager, die die Anmeldeinformationen enthalten, die für den Zugriff auf Ihre Docker-Abbilder 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: Protokollierungsfrequenz

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 der 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 Bereitstellungsprogramm 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 beim Erstellen des Buckets darauf, die Regeln für Bucket-Namen zu befolgen, die im Benutzerhandbuch für Amazon Simple Storage Service 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 Benutzerhandbuch für Amazon Simple Storage Service.

> **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 von Versioning im Benutzerhandbuch für Amazon Simple Storage Service.

> **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 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 Lambda-Funktionen erlauben, als Root auf dem AWS IoT Greengrass Core ausgeführt zu werden. Weitere Informationen finden Sie unter the section called “Ausführen einer Lambda-Funktion als Root”.

Anzeigename in der AWS IoT Konsole: Docker-Benutzer-ID

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `^[0-9]{1,5}$`

`AWSSecretsArnList`

Die Amazon Resource Names (ARNs) der Secrets in AWS Secrets Manager, die die Anmeldeinformationen enthalten, die für den Zugriff auf Ihre Docker-Abbilder 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: Protokollierungsfrequenz

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 der 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 beim Erstellen des Buckets darauf, die Regeln für Bucket-Namen zu befolgen, die im Benutzerhandbuch für Amazon Simple Storage Service 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 Benutzerhandbuch für Amazon Simple Storage Service.

> **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 von Versioning im Benutzerhandbuch für Amazon Simple Storage Service.

> **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 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 Lambda-Funktionen erlauben, als Root auf dem AWS IoT Greengrass Core ausgeführt zu werden. Weitere Informationen finden Sie unter the section called “Ausführen einer Lambda-Funktion als Root”.

Anzeigename in der AWS IoT Konsole: Docker-Benutzer-ID

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `^[0-9]{1,5}$`

`AWSSecretsArnList`

Die Amazon Resource Names (ARNs) der Secrets in AWS Secrets Manager, die die Anmeldeinformationen enthalten, die für den Zugriff auf Ihre Docker-Abbilder 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: Protokollierungsfrequenz

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 eine `ConnectorDefinition` mit einer Anfangsversion, die den Greengrass Docker-Anwendungsbereitstellungs-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": "myS3Bucket",
                "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":"myS3Bucket",
  "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.

Einrichten des Docker-Benutzers auf dem AWS IoT Greengrass Core

Der Bereitstellungs-Connector der Greengrass-Docker-Anwendung 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 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 Lambda-Funktionen erlauben, als Root auf dem AWS IoT Greengrass Core ausgeführt zu werden. Weitere Informationen finden Sie unter the section called “Ausführen einer Lambda-Funktion als Root”.

1. Erstellen Sie den Benutzer. Sie können den `useradd`-Befehl ausführen und die optionale `-u`-Option zum Zuweisen einer UID hinzufügen. Beispielsweise:

   ```
   sudo useradd -u 1234 user-name
   ```

2. Fügen Sie den Benutzer der `docker`-Gruppe auf dem Core hinzu. Beispielsweise:

   ```
   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. Beispielsweise:

   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.

Nutzungsinformationen

Wenn Sie den Greengrass-Docker-Anwendungsbereitstellungs-Connector verwenden, sollten Sie die folgenden implementierungsspezifischen Nutzungsinformationen beachten.

- Präfix für Projektnamen behoben. 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.
- Protokollierungsverhalten. Der Konnektor schreibt Statusinformationen und Fehlerbehebungsinformationen in eine Protokolldatei. Sie können so konfigurierenAWS IoT Greengrass, dass Protokolle an CloudWatch -Protokolle gesendet werden und Protokolle 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.

- Aktualisieren von Docker-Images. 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 Bereinigungsvorgänge. Wenn der Greengrass-Daemon während eines Neustarts beendet wird, wird der `docker-compose down` Befehl initiiert. Alle Docker-Container haben maximal 10 Minuten, nachdem initiiert `docker-compose down` wurde, um 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.
- Ausführen von Docker-Befehlen. 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. Ressourcen-IDs müssen in der Gruppe eindeutig sein. Weisen Sie daher keine Ressourcen-ID zu, die möglicherweise mit dieser reservierten Ressourcen-ID in Konflikt steht.
- Offline-Modus. Wenn Sie den `DockerOfflineMode` Konfigurationsparameter auf setzen`True`, kann der Docker-Konnektor im Offline-Modus arbeiten. Dies kann passieren, wenn eine Greengrass-Gruppenbereitstellung neu gestartet wird, während das Core-Gerät offline ist, und der Konnektor keine Verbindung zu Amazon S3 oder Amazon ECR herstellen kann, um die Docker-Compose-Datei abzurufen.

  Wenn der Offline-Modus aktiviert ist, versucht der Konnektor, Ihre Compose-Datei herunterzuladen und `docker login` Befehle wie bei einem normalen Neustart auszuführen. Wenn diese Versuche fehlschlagen, sucht der Konnektor 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 ruft aus lokalen Bildern ab. Wenn die Compose-Datei oder die lokalen Bilder nicht vorhanden sind, schlägt der Konnektor 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-APIs verwenden, 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 auf 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 Connectors in der Gruppe oder mit AWS IoT und dem lokalen Schattenservice kommunizieren kann. Weitere Informationen finden Sie unter the section called “Konfigurieren der MQTT-Kommunikation mit Docker-Containern”.
- Greengrass-Lambda-Funktionen können eine freigegebene Datei aktualisieren, um Informationen an Docker-Container zu übergeben. 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 Kerns finden Sie unter Erste Schritte mit AWS IoT Greengrass.

So konfigurieren Sie einen Docker-Container als Client-Gerät und fügen ihn einer Greengrass-Gruppe hinzu

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. In diesem Beispiel *path-to-device-certs* steht für den Ordner, 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 IoTKonsole 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 Client-Geräte und dann Zuordnen aus.
5. Wählen Sie im Modal Client-Gerät mit dieser Gruppe verknüpfen die Option Neues AWS IoT Objekt erstellen aus.

   Die Seite Objekte erstellen wird in einer neuen Registerkarte geöffnet.
6. Wählen Sie auf der Seite Objekte erstellen die Option Einzelobjekt erstellen und dann Weiter aus.
7. Geben Sie auf der Seite Objekteigenschaften 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 vorhandene Richtlinie aus, die Berechtigungen gewährt, die Client-Geräte benötigen, und wählen Sie dann Objekt erstellen aus.

     Ein Modal wird geöffnet, in dem Sie die Zertifikate und Schlüssel herunterladen können, die das Gerät zum Herstellen einer Verbindung mit dem AWS Cloud und dem -Kern verwendet.
   - Erstellen und fügen Sie eine neue Richtlinie an, die Client-Geräteberechtigungen gewährt. 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 unter Richtlinienname einen Namen ein, der die Richtlinie beschreibt, z. B. **`GreengrassV1ClientDevicePolicy`**.

        ii. Wählen Sie auf der Registerkarte Richtlinienanweisungen 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 zum Einschränken des Zugriffs dieser Richtlinie finden Sie unter Geräteauthentifizierung und -autorisierung für AWS IoT Greengrass.

```
{
  "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. B. GreengrassV1ClientDevicePolicy.

Wenn die Richtlinie nicht angezeigt wird, wählen Sie die Schaltfläche Aktualisieren.

ii. Wählen Sie Objekt erstellen aus.

Ein Modal wird geöffnet, in dem Sie die Zertifikate und Schlüssel herunterladen können, die das Gerät zum Herstellen einer Verbindung mit dem AWS Cloud und dem -Kern verwendet.

10. Laden Sie im Modal Zertifikate und Schlüssel herunterladen die Gerätezertifikate herunter.

> **Important**
>
> Bevor Sie Fertig wählen, laden Sie die Sicherheitsressourcen herunter.

Gehen Sie wie folgt vor:

a. Wählen Sie für Gerätezertifikat die Option Herunterladen aus, um das Gerätezertifikat herunterzuladen.

b. Wählen Sie für Öffentliche Schlüsseldatei die Option Herunterladen aus, um den öffentlichen Schlüssel für das Zertifikat herunterzuladen.

c. Wählen Sie für Private Schlüsseldatei die Option Herunterladen aus, um die private Schlüsseldatei für das Zertifikat herunterzuladen.

d. Überprüfen Sie die Serverauthentifizierung im -AWS IoTEntwicklerhandbuch und wählen Sie das entsprechende Stammzertifizierungsstellenzertifikat aus. Wir empfehlen Ihnen, Amazon Trust Services (ATS)-Endpunkte und ATS-Stammzertifizierungsstellenzertifikate zu verwenden. Wählen Sie unter Stammzertifizierungsstellenzertifikate die Option Herunterladen für ein Stammzertifizierungsstellenzertifikat aus.

e. Wählen Sie Erledigt aus.

Notieren Sie sich die Zertifikat-ID, die in den Dateinamen für das Gerätezertifikat und die Schlüssel üblich ist. 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 Daten nach einem Thema filtern für Themenfilter die Option **$aws/things/*MyDockerDevice*/shadow/update/accepted**und dann Weiter aus. Ersetzen Sie *MyDockerDevice* 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

Beachten Sie bei der Verwendung des Greengrass Docker-Anwendungsbereitstellungs-Connectors die folgenden Sicherheitsüberlegungen.

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 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. Sie erfordert auch Zugriff auf Ihr Amazon-ECR-Autorisierungstoken, wenn Ihre Docker-Images in einem privaten Repository in Amazon ECR gespeichert sind.

Lizenzen

Der Greengrass Docker-Anwendungsbereitstellungs-Connector enthält die folgende Software/Lizenzierung von Drittanbietern:

- AWS SDK for 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 Konnektor wurde gemäß der Greengrass Core Software License Agreement veröffentlicht.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in jeder Version des Konnektors beschrieben.

Version	Änderungen
7	Es wurde hinzugefügt`DockerOfflineMode` , um eine vorhandene Docker-Compose-Datei zu verwenden, wenn offline AWS IoT Greengrass startet. Implementierte Wiederholungsversuche für den `docker login` Befehl. Unterstützung für 32-Bit-UIDs .
6	Es wurde hinzugefügt`StopContainersOnNewDeployment` , um die Containerbereinigung zu überschreiben, wenn eine neue Bereitstellung durchgeführt wird oder CCPC beendet wird. Sicherere Mechanismen zum Herunterfahren und Starten. YAML-Validierungsfehlerbehebung.
5	Images werden abgerufen, bevor ausgeführt wird`docker-compose down` .
4	pull-before-up Verhalten zum Aktualisieren von Docker-Images hinzugefügt.
3	Es wurde ein Problem beim Suchen von Umgebungsvariablen behoben.
2	Der Parameter `ForceDeploy` 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 AWS

> **Warning**
>
> Dieser Connector ist in die erweiterte Lebensphase übergegangen undAWS IoT Greengrass wird keine Updates veröffentlichen, die Funktionen, Verbesserungen vorhandener Funktionen, Sicherheitspatches oder Bugfixes bereitstellen. Weitere Informationen finden Sie unter AWS IoT Greengrass Version 1Wartungspolitik.

Der IoT Analytics Analytics-Connector sendet lokale Gerätedaten anAWS IoT Analytics. Sie können diesen Connector 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 anAWS IoT Analytics Kanäle in der aktuellenAWS-Konto und 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 erfassen, speichern, verarbeiten und abfragen können. In 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 um unstrukturierte oder stapelabhängige Daten von Geräten mit beschränkten Ressourcen zu verarbeiten.

Dieser Connector verwendet die `BatchPutMessage`API, 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	`arn:aws:greengrass:region::/connectors/IoTAnalytics/versions/4`
3	`arn:aws:greengrass:region::/connectors/IoTAnalytics/versions/3`
2	`arn:aws:greengrass:region::/connectors/IoTAnalytics/versions/2`
1	`arn:aws:greengrass:region::/connectors/IoTAnalytics/versions/1`

Informationen über Versionsänderungen finden Sie im Änderungsprotokoll.

Voraussetzungen

Dieser Konnektor hat die folgenden Anforderungen:

Version 3 - 4

- AWS IoT Greengrass Core-Software v1.9.3 oder höher.
- Python-Version 3.7 oder 3.8 wurde auf dem Kerngerät installiert und zur PATH-Umgebungsvariablen hinzugefügt.

> **Note**
>
> Um Python 3.8 zu verwenden, führen Sie den folgenden Befehl aus, um einen symbolischen Link vom Standard-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 beide AWS IoT Greengrassund unterstützt AWS IoT Analyticswerden.
- 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 AnalyticsZielkanäle müssen dasselbe Konto verwenden und sich im selben BereichAWS-Region wie dieser Connector befinden.

- Die Greengrass-Gruppenrolle wurde so konfiguriert, dass sie die`iotanalytics:BatchPutMessage` Aktion auf Zielkanälen zulässt, wie in der folgenden Beispiel-IAM-Richtlinie gezeigt. Die Kanäle müssen im aktuellen ModusAWS-Konto und in der Region sein.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1528133056761",
            "Action": [
                "iotanalytics:BatchPutMessage"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iotanalytics:region:account-id:channel/channel_1_name",
```

```
                "arn:aws:iotanalytics:region:account-id: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. Weitere Informationen finden Sie unter the section called “Verwalten der Gruppenrolle (Konsole)” oder the section called “Verwalten der Gruppenrolle (CLI)”.

Versions 1 - 2

- AWS IoT GreengrassCore-Software v1.7 oder höher.
- Python-Version 2.7 wurde auf dem Kerngerät installiert und zur Umgebungsvariablen PATH hinzugefügt.
- Dieser Connector kann nur in Amazon Web Services Services-Regionen verwendet werden, in denen beide AWS IoT Greengrassund unterstützt AWS IoT Analyticswerden.
- 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 AnalyticsZielkanäle müssen dasselbe Konto verwenden und sich im selben BereichAWS-Region wie dieser Connector befinden.

- Die Greengrass-Gruppenrolle wurde so konfiguriert, dass sie die`iotanalytics:BatchPutMessage` Aktion auf Zielkanälen zulässt, wie in der folgenden Beispiel-IAM-Richtlinie gezeigt. Die Kanäle müssen im aktuellen ModusAWS-Konto und in der Region sein.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1528133056761",
            "Action": [
```

```
                "iotanalytics:BatchPutMessage"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iotanalytics:region:account-id:channel/channel_1_name",
                "arn:aws:iotanalytics:region:account-id: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. Weitere Informationen finden Sie unter the section called “Verwalten der Gruppenrolle (Konsole)” oder the section called “Verwalten der Gruppenrolle (CLI)”.

Parameter

`MemorySize`

Die Speichergröße (in KB), die diesem Konnektor zugewiesen werden soll.

Anzeigename in derAWS IoT Konsole: Speichergröße

Erforderlich:`true`

Typ: `string`

Gültiges Muster:`^[0-9]+$`

`PublishRegion`

DasAWS-Region, in dem deineAWS 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 derAWS IoT Konsole: Region veröffentlichen

Erforderlich:`false`

Typ: `string`

Gültiges Muster:`^$|([a-z]{2}-[a-z]+-\\d{1})`

`PublishInterval`

Das Intervall (in Sekunden) für die Veröffentlichung eines empfangener Datenstapels an AWS IoT Analytics.

Anzeigename in derAWS IoT Konsole: Veröffentlichungsintervall

Erforderlich:`false`

Typ: `string`

Standardwert: `1`

Gültiges Muster:`$|^[0-9]+$`

`IotAnalyticsMaxActiveChannels`

Die maximale Anzahl der AWS IoT Analytics-Kanäle, die von dem Konnektor aktiv überwacht werden. 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 derAWS IoT Konsole: Maximale Anzahl aktiver Kanäle

Erforderlich:`false`

Typ: `string`

Standardwert: `50`

Gültiges Muster:`^$|^[1-9][0-9]*$`

`IotAnalyticsQueueDropBehavior`

Das Verhalten für das Löschen von Nachrichten aus einer Kanalwarteschlange, wenn die Warteschlange voll ist.

Anzeigename in derAWS IoT Konsole: Verhalten beim Löschen von Warteschlangen

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 derAWS IoT Konsole: Maximale Warteschlangengröße pro Kanal

Erforderlich:`false`

Typ: `string`

Standardwert: `2048`

Gültiges Muster:`^$|^[1-9][0-9]*$`

`IotAnalyticsBatchSizePerChannel`

Die maximale Anzahl der in einer Stapelanfrage an einen AWS IoT Analytics-Kanal gesendeten Nachrichten. Dieser Wert muss größer als 0 sein.

Anzeigename in derAWS IoT Konsole: Maximale Anzahl von Nachrichten, die pro Kanal gebündelt werden sollen

Erforderlich:`false`

Typ: `string`

Standardwert: `5`

Gültiges Muster:`^$|^[1-9][0-9]*$`

`IotAnalyticsDefaultChannelName`

Der Name des AWS IoT Analytics-Kanals, der von diesem Konnektor für Nachrichten verwendet wird, die an ein vom Kunden definiertes Eingabethema gesendet werden.

Anzeigename in derAWS IoT Konsole: Standard-Kanalname

Erforderlich:`false`

Typ: `string`

Gültiges Muster:`^[a-zA-Z0-9_]$`

`IsolationMode`

Der Containerisierungsmodus für diesen Konnektor. Der Standardwert ist `GreengrassContainer`. Hierbei wird der Konnektor in einer isolierten Laufzeitumgebung innerhalb des AWS IoT Greengrass-Containers ausgeführt.

> **Note**
>
> Die Standardeinstellung für Containerisierung für die Gruppe gilt nicht für Konnektoren.

Anzeigename in derAWS 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 Version`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 derAWS IoT Greengrass Konsole können Sie auf der Seite Connectors 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 Konnektor bestätigt, dass IDs für jede an AWS IoT Analytics gesendete Nachricht in der ausgehenden `BatchPutMessage`-Anfrage eindeutig sind. 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` das die folgenden Eigenschaften:

`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. Bevor die Nachrichten an AWS IoT Analytics gesendet werden, verschlüsselt und formatiert der Konnektor diese Nachrichten 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. Bevor die Nachrichten an AWS IoT Analytics gesendet werden, verschlüsselt und formatiert der Konnektor diese Nachrichten 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.

Ausgabedaten

Dieser Connector veröffentlicht Statusinformationen als Ausgabedaten im MQTT-Thema. Diese Information enthält die Antwort, die vonAWS IoT Analytics für jede Eingangsnachricht zurückgegeben wird, die sie empfängt und an die sie sendetAWS 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 wiederholbaren Fehler feststellt (z. B. Verbindungsfehler), versucht er die Veröffentlichung im nächsten Batch erneut. Der exponentielle Backoff wird vomAWS SDK behandelt. Anfragen mit wiederholbaren Fehlern werden zur weiteren Veröffentlichung entsprechend dem `IotAnalyticsQueueDropBehavior`-Parameter wieder der Kanalwarteschlange hinzugefügt.

Verwendungsbeispiel

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. Weitere Informationen finden Sie unter 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 GreengrassCore 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 hochladenAWS Lambda.

   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.

- Stellen Sie die Lambda-Funktion als Quelle und den Konnektor als Ziel ein 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 derAWS IoT Konsole anzuzeigen.

4. Stellen Sie die Gruppe bereit.
5. Abonnieren Sie in derAWS IoT Konsole auf der Testseite das Thema Ausgabedaten, um Statusmeldungen vom Connector einzusehen. Die Lambda-Beispielfunktion ist langlebig und beginnt sofort 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 Connector.

```
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 von derAWS SDK for Python (Boto3) für dieAWS IoT Analytics `batch_put_message`Aktion festgelegten Grenzen.
- Alle von derAWS IoT Analytics BatchPutMessageAPI festgelegten Kontingente. Weitere Informationen finden Sie unter Service Quotas fürAWS 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 Kanal-ARNs), daher wird das Senden von Daten an regionsübergreifende oder kontoübergreifende Kanäle nicht unterstützt.
- Alle vom AWS IoT Greengrass-Core vorgegebenen Kontingente. Weitere Informationen finden Sie unter Service Quotas für denAWS 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 for 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 -Entwicklers.

Version	Änderungen
4	Fügt den`IsolationMode` Parameter hinzu, um den Containerisierungsmodus für den Connector zu konfigurieren.
3	Die Lambda-Laufzeit wurde auf Python 3.7 aktualisiert, wodurch sich die Laufzeitanforderungen ändern.
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)”
- Was ist AWS IoT Analytics? im AWS IoT Analytics-Benutzerhandbuch

Anschluss für IoT-Ethernet-IP-Protokoll-Adapter

Der IoT-Ethernet-IP-ProtokolladapterAnschlusssammelt Daten von lokalen Geräten unter Verwendung des EtherNet/IP-Protokolls. Sie können diesen Konnektor verwenden, um Daten von mehreren Geräten zu sammeln und in einer`StreamManager`Nachrichten-Stream.

Sie können diesen Konnektor auch mit dem IoT verwenden SiteWise Connector und Ihr IoT SiteWise SiteWise-Gateway. Ihr Gateway muss die Konfiguration für den Connector angeben. Weitere Informationen finden Sie unterKonfigurieren einer EtherNet/IP (EIP) -Quelleim IoT SiteWise - Benutzerhandbuch.

> **Note**
>
> Dieser Konnektor läuft inKein ContainerIsolationsmodus, damit Sie ihn in einemAWS IoT GreengrassGruppe, die in einem Docker-Container ausgeführt wird.

Dieser Konnektor hat die folgenden Versionen.

Version	ARN
2 (empfohlen)	`arn:aws:greengrass:  region::/connectors/IoTEIPProtocolAdaptor/versions/2`
1	`arn:aws:greengrass:  region::/connectors/IoTEIPProtocolAdaptor/versions/1`

Informationen über Versionsänderungen finden Sie im Änderungsprotokoll.

Voraussetzungen

Dieser Konnektor hat die folgenden Anforderungen:

Version 1 and 2

- AWS IoT GreengrassCore-Software v1.10.2 oder höher.
- Stream-Manager ist auf derAWS IoT GreengrassGruppe.

- Java 8 auf dem Core-Gerät installiert und demPATHUmgebungsvariable.
- Mindestens 256 MB zusätzlicher RAM. Diese Anforderung gilt zusätzlich zuAWS IoT GreengrassAnforderungen an den Kernspeicher.

> **Note**
>
> Dieser Konnektor 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

Connector-Parameter

Dieser Konnektor unterstützt die folgenden Parameter:

`LocalStoragePath`

Das Verzeichnis auf derAWS IoT Greengrasshosten, dass das IoT SiteWise -Konnektor kann persistente Daten in schreiben. Das Standardverzeichnis ist `/var/sitewise`.

Anzeigename imAWS IoT-Konsole Lokaler Speicherpfad

Erforderlich`false`

Typ: `string`

Gültiges Pattern:`^\s*$|\/.`

`ProtocolAdapterConfiguration`

Der Satz von EtherNet/IP-Collector-Konfigurationen, von denen der Konnektor Daten sammelt oder mit dem der Konnektor eine Verbindung herstellt. Dies kann sich um eine leere Liste handeln.

Anzeigename imAWS IoT-Konsole Konfiguration des Protokolladapters

Erforderlich`true`

Type: Eine wohlgeformte JSON-Zeichenfolge, die die Gruppe der unterstützten Feedback-Konfigurationen definiert.

Es folgt 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 erstelltConnectorDefinitionEr enthält eine Erstversion, die den IoT-Ethernet-IP-Protokoll-Adapter-Konnektor 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 Anschluss hat einelangdauerndLebenszyklus.

Eingabedaten

Dieser Konnektor akzeptiert keine MQTT-Nachrichten als Eingabedaten.

Ausgabedaten

Dieser Konnektor veröffentlicht Daten in`StreamManager`aus. Sie müssen den Ziel-Message-Stream konfigurieren. Die Ausgabemeldungen haben die folgende Struktur:

```
{
    "alias": "string",
    "messages": [
        {
            "name": "string",
            "value": boolean|double|integer|string,
            "timestamp": number,
            "quality": "string"
        }
    ]
}
```

Lizenzen

Der Connector des IoT-Ethernet-IP-Protokolladapters enthält die folgende Drittanbieter-Software/Lizenz:

- EtherNet/IP-Client
- MapDB
- Elsa

Dieser Konnektor wird unter derLizenzvereinbarung für die Greengrass Core-Softwareaus.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in jeder Version des -Konnektors beschrieben.

Version	Änderungen	Datum
2	Diese Version enthält Fehlerbehebungen.	23. Dezember 2021
1	Erstversion.	15. Dezember 2020

Eine Greengrass-Gruppe kann jeweils nur eine Version des -Konnektors 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 SiteWise -Konnektor

Der IoT SiteWise -Konnektor sendet lokale Geräte- und Gerätedaten an Komponenteneigenschaften in AWS IoT SiteWise. Sie können diesen Konnektor verwenden, um Daten von mehreren OPC-UA-Servern zu sammeln und sie in IoT SiteWise zu veröffentlichen. Der Konnektor sendet die Daten an Komponenteneigenschaften im aktuellen AWS-Konto und in der Region.

> **Note**
>
> IoT SiteWise ist ein vollständig verwalteter Service, der Daten von Industriegeräten und Geräten 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 Konnektor sendet Daten an IoT SiteWise mit den OPC-UA-Datenstrompfaden, die von den OPC-UA-Servern gesendet werden. 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 Kern die Verbindung zum Internet verliert, speichert der Konnektor Daten zwischen, bis er erfolgreich eine Verbindung zum herstellen kannAWS 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 IoT SiteWise -Konnektor konfiguriert und bereitgestellt haben, können Sie ein Gateway und OPC-UA-Quellen in der IoT SiteWise -Konsole hinzufügen. Wenn Sie eine Quelle in der Konsole konfigurieren, können Sie die vom IoT SiteWise -Konnektor gesendeten OPC-UA-

Datenstrompfade filtern oder ihr ein Präfix voranstellen. 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 Messungseigenschaften von IoT SiteWise -Komponenten 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 aus anderen Quellen als OPC-UA-Servern in IoT SiteWise hochzuladen. Stream Manager bietet auch anpassbare Unterstützung für Persistenz und Bandbreitenmanagement. Weitere Informationen finden Sie unter Verwalten von Daten-Streams.
> Dieser Konnektor wird im Modus Keine Containerisolierung ausgeführt, sodass Sie ihn in einer Greengrass-Gruppe bereitstellen können, die in einem Docker-Container ausgeführt wird.

Dieser Konnektor hat die folgenden Versionen.

Version	ARN
12 (empfohlen)	`arn:aws:greengrass:region::/connectors/IoTSiteWise/versions/12`
11	`arn:aws:greengrass:region::/connectors/IoTSiteWise/versions/11`
10	`arn:aws:greengrass:region::/connectors/IoTSiteWise/versions/10`

Version	ARN
9	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/9
8	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/8
7	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/7
6	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/6
5	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/5
4	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/4
3	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/3
2	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/2
1	arn:aws:greengrass:*region*::/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 Core-Software v1.10.2 und Stream Manager .

- AWS IoT Greengrass Core-Software 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 Konnektor kann nur in Amazon Web Services-Regionen verwendet werden, in denen AWS IoT Greengrass sowohl als auch IoT SiteWise unterstützt werden.
- Eine IAM-Richtlinie, die der Greengrass-Gruppenrolle hinzugefügt wurde.
  Diese Rolle ermöglicht der AWS IoT Greengrass-Gruppe den Zugriff auf die `iotsitewise:BatchPutAssetPropertyValue`-Aktion für die Zielstammkomponente und die untergeordneten Elemente, wie im folgenden Beispiel gezeigt. Sie können die `Condition` aus der Richtlinie entfernen, um dem Konnektor den Zugriff auf alle Ihre IoT SiteWise -Komponenten zu ermöglichen.

```
{
    "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**
>
> Mit dieser Version werden neue Anforderungen eingeführt: AWS IoT Greengrass Core-Software v1.10.0 und Stream-Manager.

- AWS IoT Greengrass Core-Software 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 Konnektor kann nur in Amazon Web Services-Regionen verwendet werden, in denen AWS IoT Greengrass sowohl als auch IoT SiteWise unterstützt werden.
- Eine IAM-Richtlinie, die der Greengrass-Gruppenrolle hinzugefügt wurde.
  Diese Rolle ermöglicht der AWS IoT Greengrass-Gruppe den Zugriff auf die `iotsitewise:BatchPutAssetPropertyValue`-Aktion für die Zielstammkomponente und die untergeordneten Elemente, wie im folgenden Beispiel gezeigt. Sie können die `Condition` aus der Richtlinie entfernen, um dem Konnektor den Zugriff auf alle Ihre IoT SiteWise -Komponenten zu ermöglichen.

```
{
    "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 Core-Software v1.9.4.
- Java 8 auf dem Core-Gerät installiert und der Umgebungsvariablen PATH hinzugefügt.
- Dieser Konnektor kann nur in Amazon Web Services-Regionen verwendet werden, in denen AWS IoT Greengrass sowohl als auch IoT SiteWise unterstützt werden.
- Eine IAM-Richtlinie, die der Greengrass-Gruppenrolle hinzugefügt wurde.
  Diese Rolle ermöglicht der AWS IoT Greengrass-Gruppe den Zugriff auf die `iotsitewise:BatchPutAssetPropertyValue`-Aktion für die Zielstammkomponente und die untergeordneten Elemente, wie im folgenden Beispiel gezeigt. Sie können die `Condition` aus der Richtlinie entfernen, um dem Konnektor den Zugriff auf alle Ihre IoT SiteWise -Komponenten zu ermöglichen.

```
{
    "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 Core-Software v1.10.0.
- Java 8 auf dem Core-Gerät installiert und der Umgebungsvariablen PATH hinzugefügt.
- Dieser Konnektor kann nur in Amazon Web Services-Regionen verwendet werden, in denen AWS IoT Greengrass sowohl als auch IoT SiteWise unterstützt werden.
- Eine IAM-Richtlinie, die der Greengrass-Gruppenrolle hinzugefügt wurde.
  Diese Rolle ermöglicht der AWS IoT Greengrass-Gruppe den Zugriff auf die `iotsitewise:BatchPutAssetPropertyValue`-Aktion für die Zielstammkomponente und die untergeordneten Elemente, wie im folgenden Beispiel gezeigt. Sie können die `Condition` aus der Richtlinie entfernen, um dem Konnektor den Zugriff auf alle Ihre IoT SiteWise -Komponenten zu ermöglichen.

```
{
    "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 Core-Software v1.9.4.
- Java 8 auf dem Core-Gerät installiert und der Umgebungsvariablen PATH hinzugefügt.
- Dieser Konnektor kann nur in Amazon Web Services-Regionen verwendet werden, in denen AWS IoT Greengrass sowohl als auch IoT SiteWise unterstützt werden.
- Eine IAM-Richtlinie, die der Greengrass-Gruppenrolle hinzugefügt wurde.
  Diese Rolle ermöglicht der AWS IoT Greengrass-Gruppe den Zugriff auf die `iotsitewise:BatchPutAssetPropertyValue`-Aktion für die Zielstammkomponente und die untergeordneten Elemente, wie im folgenden Beispiel gezeigt. Sie können die `Condition` aus der Richtlinie entfernen, um dem Konnektor den Zugriff auf alle Ihre IoT SiteWise -Komponenten zu ermöglichen.

  ```
  {
      "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 Core-Software v1.9.4.
- Java 8 auf dem Core-Gerät installiert und der Umgebungsvariablen PATH hinzugefügt.
- Dieser Konnektor kann nur in Amazon Web Services-Regionen verwendet werden, in denen AWS IoT Greengrass sowohl als auch IoT SiteWise unterstützt werden.
- Eine IAM-Richtlinie, die der Greengrass-Gruppenrolle hinzugefügt wurde und den Zugriff auf AWS IoT Core und die `-iotsitewise:BatchPutAssetPropertyValue`Aktion für die Ziel-Root-Komponente und ihre untergeordneten Elemente ermöglicht, wie im folgenden Beispiel gezeigt. Sie können die `Condition` aus der Richtlinie entfernen, um dem Konnektor den Zugriff auf alle Ihre IoT SiteWise -Komponenten zu ermöglichen.

```
{
    "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.

Parameter

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 IoT SiteWise -Konnektor persistente Daten schreiben kann. Standardeinstellung: `/var/sitewise`.

Anzeigename in der AWS IoT Konsole: Lokaler Speicherpfad

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `^\s*$|\/.`

`AWSSecretsArnList`

Eine Liste von Secrets in AWS Secrets Manager, die jeweils ein OPC-UA-Schlüssel-Wert-Paar aus Benutzername und Passwort enthalten. Alle Secrets müssen vom Typ Schlüssel-Wert-Paar sein.

Anzeigename in der AWS IoT Konsole: Liste der ARNs für Benutzernamen-/Passwortgeheimnisse von OPC-UA

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 IoT SiteWise -Festplattennutzung. Standardmäßig 10 GB.

Anzeigename in der AWS IoT Konsole: Maximale Größe des Festplattenpuffers

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `^\s*$|[0-9]+`

Version 1

`SiteWiseLocalStoragePath`

Das Verzeichnis auf dem AWS IoT Greengrass Host, in das der IoT SiteWise -Konnektor persistente Daten schreiben kann. Standardeinstellung: `/var/sitewise`.

Anzeigename in der AWS IoT Konsole: Lokaler Speicherpfad

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `^\s*$|\/.`

`SiteWiseOpcuaUserIdentityTokenSecretArn`

Das Secret in AWS Secrets Manager, das das OPC-UA-Benutzernamen- und Passwort-Schlüssel-Wert-Paar enthält. Dieses Secret muss ein Schlüssel-Wert-Paar-Typ-Secret sein.

Anzeigename in der AWS IoT Konsole: ARN des geheimen OPC-UA-Benutzernamens/Passworts

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 ein OPC-UA-Benutzernamen- und Passwort-Secret verweist.

Anzeigename in der AWS IoT Konsole: geheime Ressource für OPC-UA-Benutzername/Passwort

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `^$|.+`

`MaximumBufferSize`

Die maximale Größe in GB für die IoT SiteWise -Festplattennutzung. Standardmäßig 10 GB.

Anzeigename in der AWS IoT Konsole: Maximale Größe des Festplattenpuffers

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 Anfangsversion, die den IoT SiteWise -Konnektor 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 einen Connector über die Seite Connectors der Gruppe hinzufügen. Weitere Informationen finden Sie unter the section called “Beginnen Sie mit Konnektoren (Konsole)”.

Eingabedaten

Dieser Konnektor akzeptiert keine MQTT-Nachrichten als Eingabedaten.

Ausgabedaten

Dieser Konnektor veröffentlicht keine MQTT-Nachrichten als Ausgabedaten.

Einschränkungen

Dieser Konnektor unterliegt den folgenden Einschränkungen, die von IoT SiteWise auferlegt werden, einschließlich der folgenden. Weitere Formate finden Sie unter -AWS IoT SiteWiseEndpunkte und -Kontingente im Allgemeine AWS-Referenz.

- Maximale Anzahl von Gateways pro AWS-Konto.
- Maximale Anzahl von OPC-UA-Quellen pro Gateway.
- Maximale Rate von timestamp-quality-value (TQV)-Datenpunkten, die pro gespeichert werdenAWS-Konto.
- Maximale Rate der pro Objekteigenschaft gespeicherten TQV-Datenpunkte.

Lizenzen

Version 9, 10, 11, and 12

Der IoT SiteWise -Konnektor enthält die folgende Software/Lizenzierung von Drittanbietern:

- MapDB
- Trichter
- Eclipse-Milo

Dieser Konnektor wurde gemäß der Greengrass Core Software License Agreement veröffentlicht.

Versions 6, 7, and 8

Der IoT SiteWise -Konnektor enthält die folgende Software/Lizenzierung von Drittanbietern:

- Milo / EDL 1.0

Dieser Konnektor wurde gemäß der Greengrass Core Software License Agreement veröffentlicht.

Versions 1, 2, 3, 4, and 5

Der IoT SiteWise -Konnektor enthält die folgende Software/Lizenzierung von Drittanbietern:

- Milo / EDL 1.0
- Chronicle-Queue/Apache-Lizenz 2.0

Dieser Konnektor wurde gemäß der Greengrass Core Software License Agreement veröffentlicht.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in jeder Version des Konnektors beschrieben.

Version	Änderungen	Datum
12	• Diese Version enthält Fehlerbehebungen.	22. Dezember 2021
11	• Unterstützung für Zeichenfolgen, die versteckte oder nicht darstellbare Zeichen enthalten. Ausgeblendete und nicht darstellbare Zeichen werden automatisch entfernt, bevor die Zeichenfolgen an den gesendet werdenAWS Cloud. • Es wurde ein Problem behoben, das dazu führte, dass das IoT SiteWise -Gateway ungültige Anforderungen unendlich erneut versuchte. • Es wurde ein Problem behoben, das zu einem beschädigten Checkpoint führte, wenn das IoT SiteWise -Gateway mit einer	24. März 2021

Version	Änderungen	Datum
	Hochfrequenz-Datenquelle verbunden war. • Verbesserte Fehlermeldungen zur Fehlerbehebung bei der Gateway-Konfiguration.	
10	Konfiguriert `StreamManager` für die Verbesserung der Handhabung, wenn die Quellverbindung unterbrochen und wieder hergestellt wird. Diese Version akzeptiert auch OPC-UA-Werte mit einem , `ServerTimestamp` wenn kein verfügbar `SourceTimestamp` ist.	22. Januar 2021
9	Unterstützung für benutzerdefinierte Greengrass `s-StreamManager` Stream-Ziele, OPC-UA-Deadbanding, benutzerdefinierten Scanmodus und benutzerdefinierte Scanrate gestartet. Bietet auch eine verbesserte Leistung bei Konfigurationsaktualisierungen, die vom IoT SiteWise -Gateway durchgeführt wurden.	15. Dezember 2020
8	Verbesserte Stabilität, wenn der Konnektor zeitweilige Netzwerkkonnektivität aufweist.	19. November 2020

Version	Änderungen	Datum
7	Es wurde ein Problem mit Gateway-Metriken behoben.	14. August 2020
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 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 `iot:*`-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
  - Gateway- CloudWatch Metriken
  - Fehlerbehebung bei einem IoT SiteWise -Gateway

Kinesis Firehose

Der Kinesis-Firehose-Konnektor veröffentlicht Daten über einen Amazon-Data-Firehose-Bereitstellungsdatenstrom an Ziele wie Amazon S3, Amazon Redshift oder Amazon OpenSearch Service.

Dieser Konnektor ist ein Datenproduzent für einen Kinesis-Bereitstellungsdatenstrom. 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:  region::/connectors/KinesisFirehose/versions/5`
4	`arn:aws:greengrass:  region::/connectors/KinesisFirehose/versions/4`
3	`arn:aws:greengrass:  region::/connectors/KinesisFirehose/versions/3`

Version	ARN
2	`arn:aws:greengrass:region::/connectors/KinesisFirehose/versions/2`
1	`arn:aws:greengrass:region::/connectors/KinesisFirehose/versions/1`

Informationen über Versionsänderungen finden Sie im Änderungsprotokoll.

Voraussetzungen

Dieser Konnektor hat die folgenden Anforderungen:

Version 4 - 5

- AWS IoT Greengrass Core-Software v1.9.3 oder höher.
- Python Version 3.7 oder 3.8 ist auf dem Core-Gerät installiert und der PATH-Umgebungsvariablen 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-Bereitstellungs-Stream. Weitere Informationen finden Sie unter Erstellen eines Amazon-Data-Firehose-Bereitstellungsdatenstroms im Amazon Kinesis Firehose-Entwicklerhandbuch.

- Die Greengrass-Gruppenrolle, die so konfiguriert ist, dass sie die `firehose:PutRecordBatch` Aktionen `firehose:PutRecord` und für den Zielbereitstellungsdatenstrom zulässt, wie in der folgenden Beispiel-IAM-Richtlinie gezeigt.

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"Stmt1528133056761",
            "Action":[
                "firehose:PutRecord",
                "firehose:PutRecordBatch"
            ],
            "Effect":"Allow",
            "Resource":[
                "arn:aws:firehose:region:account-id: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. Weitere Informationen finden Sie unter the section called “Verwalten der Gruppenrolle (Konsole)” oder the section called “Verwalten der Gruppenrolle (CLI)”.

Versions 2 - 3

- AWS IoT Greengrass Core-Software v1.7 oder höher.
- Python Version 2.7 ist auf dem Core-Gerät installiert und der PATH-Umgebungsvariablen hinzugefügt.

- Ein konfigurierter Kinesis-Bereitstellungs-Stream. Weitere Informationen finden Sie unter Erstellen eines Amazon-Data-Firehose-Bereitstellungsdatenstroms im Amazon Kinesis Firehose-Entwicklerhandbuch.
- Die Greengrass-Gruppenrolle, die so konfiguriert ist, dass die `firehose:PutRecordBatch` Aktionen `firehose:PutRecord` und im Zielbereitstellungsdatenstrom zugelassen werden, wie in der folgenden Beispiel-IAM-Richtlinie gezeigt.

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"Stmt1528133056761",
            "Action":[
                "firehose:PutRecord",
                "firehose:PutRecordBatch"
            ],
            "Effect":"Allow",
            "Resource":[
                "arn:aws:firehose:region:account-id: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. Weitere Informationen finden Sie unter the section called “Verwalten der Gruppenrolle (Konsole)” oder the section called “Verwalten der Gruppenrolle (CLI)”.

Version 1

- AWS IoT Greengrass Core-Software v1.7 oder höher.

- Python Version 2.7 ist auf dem Core-Gerät installiert und der PATH-Umgebungsvariablen hinzugefügt.
- Ein konfigurierter Kinesis-Bereitstellungs-Stream. Weitere Informationen finden Sie unter Erstellen eines Amazon-Data-Firehose-Bereitstellungsdatenstroms im Amazon Kinesis Firehose-Entwicklerhandbuch.
- Die Greengrass-Gruppenrolle, die so konfiguriert ist, dass sie die `firehose:PutRecord` Aktion im Zielbereitstellungsdatenstrom zulässt, wie in der folgenden Beispiel-IAM-Richtlinie gezeigt.

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"Stmt1528133056761",
            "Action":[
                "firehose:PutRecord"
            ],
            "Effect":"Allow",
            "Resource":[
                "arn:aws:firehose:region:account-id: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. Weitere Informationen finden Sie unter the section called “Verwalten der Gruppenrolle (Konsole)” oder the section called “Verwalten der Gruppenrolle (CLI)”.

Connector-Parameter

Dieser Konnektor stellt die folgenden Parameter bereit:

Versions 5

`DefaultDeliveryStreamArn`

Der ARN des standardmäßigen Firehose-Bereitstellungsdatenstroms, 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-ARN des Bereitstellungsdatenstroms

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 von Datensätzen, die gepuffert werden sollen (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: Publish interval

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. Der Standardwert 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 standardmäßigen Firehose-Bereitstellungsdatenstroms, 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-ARN des Bereitstellungsdatenstroms

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 von Datensätzen, die gepuffert werden sollen (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: Publish interval

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 standardmäßigen Firehose-Bereitstellungsdatenstroms, 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-ARN des Bereitstellungsdatenstroms

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 das Erstellen eines Konnektors (AWS CLI)

Der folgende CLI-Befehl erstellt eine `ConnectorDefinition` mit einer Anfangsversion, die den Konnektor 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 einen Connector über die Seite Connectors der Gruppe 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 enthält:

`data`

Die Daten, die an den Lieferstrom gesendet werden sollen.

Erforderlich: `true`

Typ: `string`

`delivery_stream_arn`

Der ARN des Ziel-Kinesis-Bereitstellungs-Streams. 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 enthält:

`data`

Die Daten, die an den Lieferstrom gesendet werden sollen.

Erforderlich: `true`

Typ: `string`

`delivery_stream_arn`

Der ARN des Ziel-Kinesis-Bereitstellungs-Streams. 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 Konnektor einen wiederholbaren Fehler erkennt (z. B. Verbindungsfehler), wiederholt er die Veröffentlichung im nächsten Batch. Exponentielles Backoff wird vom AWS SDK abgewickelt. 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"
}
```

Verwendungsbeispiel

Führen Sie die folgenden allgemeinen Schritte aus, 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. Weitere Informationen finden Sie unter 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 Konnektor sendet.

   Speichern Sie den Beispielcode als PY-Datei. Laden Sie das AWS IoT Greengrass Core SDK for 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, das Sie in hochladen AWS Lambda.

   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 nach 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 Eingabethemafilter.
      - 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 Seite Test das Ausgabedatenthema, um Statusmeldungen vom Connector anzuzeigen. Die Lambda-Beispielfunktion ist langlebig und beginnt sofort nach der Bereitstellung der Gruppe Nachrichten zu senden.

   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-Konnektor enthält die folgende Software/Lizenzierung von Drittanbietern:

- AWS SDK for 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 Konnektor wurde gemäß der Greengrass Core Software License Agreement veröffentlicht.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in jeder Version des Konnektors beschrieben.

Version	Änderungen
5	Der Parameter `IsolationMode` wurde hinzugefügt, um den Containerisierungsmodus für den Konnektor zu konfigurieren.

Version	Änderungen
4	Die Lambda-Laufzeit wurde auf Python 3.7 aktualisiert, was die Laufzeitanforderung ändert.
3	Korrektur, um übermäßige Protokollierung und andere kleinere Fehlerbehebungen zu reduzieren.
2	Unterstützung für das Senden von Batch-Dat ensätzen an Firehose in einem bestimmten Intervall hinzugefügt. • Außerdem ist die `firehose:PutRecord Batch` -Aktion in der Gruppenrolle erforderl ich. • Neue `MemorySize` -, `DeliveryS treamQueueSize` - und `PublishIn terval` -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-Entwicklerhandbuch

ML-Feedback-

> **Warning**
>
> Dieser Konnektor ist in diePhase „verlängerte Lebensdauer“, undAWS IoT Greengrassveröffentlicht keine Updates, die Funktionen, Verbesserungen vorhandener Funktionen, Sicherheitspatches oder Bugfixes bereitstellen. Weitere Informationen finden Sie unter AWS IoT Greengrass Version 1Wartungspolitik.

Der ML-Feedback-Konnektor erleichtert den Zugriff auf Ihre Machine Learning (ML) -Modelldaten für die Modellneuschulung und -analyse. Der Konnektor:

- Lädt Eingabedaten (Beispiele), die von Ihrem ML-Modell verwendet werden, in 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 die Datei verwendenSageMaker Ground Truthum Ihre Proben zu kennzeichnen undSageMakerum 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 in Amazon CloudWatch.

Um diesen Konnektor zu konfigurieren, beschreiben Sie Ihre unterstützten Feedback-Konfigurationen im JSON-Format. Eine Feedback-Konfiguration definiert Eigenschaften wie z. B. den Amazon S3 S3-Ziel-Bucket, den Inhaltstyp undSamplingstrategieaus. (Mit einer Samplingstrategie wird ermittelt, welche Beispiele hochgeladen werden sollen.)

Sie können den ML-Feedback-Konnektorin den folgenden Szenarien verwenden:

- Mit benutzerdefinierten Lambda-Funktionen. Ihre lokalen Inferenz Lambda-Funktionen verwenden dieAWS IoT GreengrassMachine Learning SDK, um diesen Konnektor aufzurufen und die Ziel-Feedback-Konfiguration, die Modelleingabe und die Modellausgabe (Prognoseergebnisse) zu übergeben. Ein Beispiel finden Sie unter the section called “Beispiel für die Verwendung”.

- Mit derKonnektor für ML-Bildklassi(V2). Um diesen Konnektor mit dem -Konnektor ML Image Classification zu verwenden, konfigurieren Sie`MLFeedbackConnectorConfigId`Parameter für den ML-Bildklassifikations-Konnektor.
- Mit derML-Objekterkennungsanschlussaus. Um diesen Konnektor mit dem ML Object Detection - Konnektor zu verwenden, konfigurieren Sie`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 GreengrassCore-Software v1.9.3 oder höher.
- PythonVersion 3.7 oder 3.8 auf dem Core-Gerät installiert und der Umgebungsvariablen PATH hinzugefügt.

> **Note**
>
> Um Python 3.8 zu verwenden, führen Sie den folgenden Befehl aus, um einen symbolischen Link vom Standardinstallationsordner für Python 3.8 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.
- DieGreengrass-Gruppenrolle.konfiguriert, um dies`3:PutObject`-Aktion für Objekte im Amazon S3 S3-Ziel-Bucket, wie im folgenden Beispiel-IAM-Richtlinie gezeigt.

```
{
    "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. Weitere Informationen finden Sie unter the section called “Verwalten der Gruppenrolle (Konsole)” oder the section called “Verwalten der Gruppenrolle (CLI)”.

- DieKonnektor CloudWatch-MetrikenSie wurden der Greengrass-Gruppe hinzugefügt und konfiguriert. Nur erforderlich, wenn Sie die Berichterstellungsfunktion für Metriken verwenden möchten.
- AWS IoT GreengrassMachine Learning Learning-SDKVersion 1.1.0 ist für die Interaktion mit diesem Konnektor erforderlich.

Parameter

`FeedbackConfigurationMap`

Eine Gruppe von einer oder mehreren Feedback-Konfigurationen, die der -Konnektor verwenden kann, um Beispiele in Amazon S3 hochzuladen. Eine Feedback-Konfiguration definiert Parameter wie z. B. den Ziel-Bucket, den Inhaltstyp und die Samplingstrategie. Wenn dieser Konnektor aufgerufen wird, gibt die aufrufende Lambda-Funktion oder der Konnektor eine Ziel-Feedback-Konfiguration an.

Anzeigename imAWS IoT-Konsole Kartonkonfigurationskarte

Erforderlich:`true`

Type: Eine wohlgeformte JSON-Zeichenfolge, die den Satz der unterstützten Feedback-Konfigurationen definiert. Ein Beispiel finden Sie unter the section called “Beispiel für FeedbackConfigurationMap”.

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 Pattern:`^[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 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 Pattern:`^[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 unterObjektschlüssel und MetadatenimBenutzerhandbuch für Amazon Simple Storage Serviceaus.

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`

Type: Eine wohlgeformte JSON-Zeichenfolge mit den folgenden Eigenschaften.

`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`ist`RANDOM_SAMPLING`aus.

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_CONFIDENCE`,`MARGIN`, oder`ENTROPY`aus.

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 imAWS IoT-Konsole Anforderungslimit

Erforderlich:`false`

Typ: `string`

Zulässige Werte: `0 - 999`

Gültiges Pattern:`^$|^[0-9]{1,3}$`

Beispiel für das Erstellen eines Konnektors (AWS CLI)

Der folgende -CLI-Befehl erstellt einen`ConnectorDefinition`mit einer Erstversion, die den ML-Feedback-Konnektorenthält.

```
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"
            }
        }
    ]
}'
```

Beispiel für FeedbackConfigurationMap

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`) <= 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`) <= 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`Kunde imAWS IoT GreengrassMachine Learning SDK zum Aufrufen des Konnektors. 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. Diese muss mit der ID einer Feedback-Konfiguration übereinstimmen, die imFeedbackConfigurationMap-Parameter für den ML-Feedback-Anschluss.

Erforderlich: true

Typ: Zeichenfolge

ModelInput

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: true

Typ: Byte

ModelPrediction

Die Prognoseergebnisse aus dem Modell. Ergebnistyp kann ein Wörterbuch oder eine Liste sein. Die Prognoseergebnisse des -Konnektors ML Image Classification beispielsweise sind eine Liste von Wahrscheinlichkeiten (z. B.`[0.25, 0.60, 0.15]`) enthalten. Diese Daten werden im Thema `/feedback/message/prediction` veröffentlicht.

Erforderlich: true

Typ: Wörterbuch oder Liste von`float`Werte

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: false

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 für CloudWatch auf der`cloudwatch/metric/put`-Thema.

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":
 "lbbqaDVFOhMlyU3gRvAX1ZIdg8P0WkGkCSSFsYFvSwLZk3j7QZhG5EXAMPLEdd4/pEXAMPLEUqU=",
          "server": "AmazonS3",
          "x-amz-expiration": "expiry-date=\"Wed, 17 Jul 2019 00:00:00 GMT\",
 rule-id=\"OGZjYWY3OTgtYWI2Zi00ZDllLWE4YmQtNzMyYzEXAMPLEoUw\"",
          "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=
\"OGZjYWY3OTgtYWI2Zi00ZDllLWE4YmQtNzMyYzEXAMPLEoUw\"",
      "key": "s3-key-prefix/UUID.file_ext",
      "ServerSideEncryption": "AES256"
    }
```

```
  },
  "id": "5aaa913f-97a3-48ac-5907-18cd96b89eeb"
}
```

Der Konnektor fügt`bucket`und`key`-Felder auf die Antwort von Amazon S3. Weitere Informationen über die Amazon S3 S3-Antwort finden Sie unterPUT ObjectimAmazon Simple Storage Service API-Referenzaus.

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. Modellprognosen werden nur dann in diesem Thema veröffentlicht, wenn Daten erfolgreich in 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 dieIoT Analytics Konnektorum dieses Thema zu abonnieren und die Informationen an zu sendenAWS IoT Analyticsfür weitere oder historische Analysen.

Themenfilter: `cloudwatch/metric/put`

Dies ist das Ausgabethema, das zum Veröffentlichen von Metriken in CloudWatch verwendet wird. Für diese Funktion müssen Sie dasKonnektor CloudWatch-Metrikenaus.

Zu den Metriken gehören:

- Die Anzahl der hochgeladenen Beispiele.
- Die Größe der hochgeladenen Beispiele.
- Die Anzahl der Fehler aus Uploads in Amazon S3.
- Die Anzahl der verworfenen Beispiele basierend auf der Samplingstrategie.
- Die Anzahl der gedrosselten Anforderungen.

Beispielausgabe: Größe der Datenprobe (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

Im folgenden Beispiel sehen Sie eine benutzerdefinierte Lambda-Funktion, die dasAWS IoT GreengrassMachine Learning Learning-SDKum Daten an den ML Feedback-Konnektor zu senden.

> **Note**
>
> Sie können dieAWS IoT GreengrassSDK für Machine Learning von derAWS IoT Greengrass Seite Downloadsaus.

```
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-Konnektor enthält die folgende Drittanbieter-Software/Lizenz:

- AWS SDK for 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 Konnektor wird unter derLizenzvereinbarung für die Greengrass Core-Softwareaus.

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 1Wartungspolitik.

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 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 Konnektors bereit, der für mehrere Plattformen verfügbar ist.

Version 2

Konnektor	Beschreibung und ARN
ML-Bildklassifizierung: Aarch64, JTX2	Bildklassifikation Inferenzdienst für NVIDIA Jetson TX2. Unterstützt die GPU-Besch leunigung. ARN: `arn:aws:greengrass:` *`region`*`::/connectors/Imag`

Konnektor	Beschreibung und ARN
	`eClassificationAarch64JTX2/versions/2`
ML-Bildklassifizierung x86_64	Bildklassifikation Inferenzdienst für x86_64 Plattformen. ARN: `arn:aws:greengrass:` *`region`*`::/connectors/ImageClassificationx86-64/versions/2`
ML-Bildklassifizierung ARMv7	Bildklassifikation Inferenzdienst für ARMv7-Plattformen. ARN: `arn:aws:greengrass:` *`region`*`::/connectors/ImageClassificationARMv7/versions/2`

Version 1

Konnektor	Beschreibung und ARN
ML-Bildklassifizierung Aarch64 JTX2	Bildklassifikation Inferenzdienst für NVIDIA Jetson TX2. Unterstützt die GPU-Beschleunigung. ARN: `arn:aws:greengrass:` *`region`*`::/connectors/ImageClassificationAarch64JTX2/versions/1`
ML-Bildklassifizierung x86_64	Bildklassifikation Inferenzdienst für x86_64 Plattformen.

Konnektor	Beschreibung und ARN
	ARN: `arn:aws:greengrass:` *region*`::/connectors/ImageClassificationx86-64/versions/1`
ML-Bildklassifizierung Armv7	Inferenzdienst zur Bildklassifizierung für ARMv7-Plattformen. ARN: `arn:aws:greengrass:` *region*`::/connectors/ImageClassificationARMv7/versions/1`

Informationen über Versionsänderungen finden Sie im Änderungsprotokoll.

Voraussetzungen

Für diese Konnektoren gelten die folgenden Anforderungen:

Version 2

- AWS IoT GreengrassKernsoftware 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 Core-Gerät installierte Apache MXNet-Framework. Weitere Informationen finden Sie unter the section called “Installieren von MXNet-Abhängigkeiten”.
- Eine ML-Ressource in der Greengrass-Gruppe, die auf eine SageMaker Modellquelle verweist. Dieses Modell muss mit dem SageMaker Bildklassifizierungsalgorithmus trainiert werden. Weitere Informationen finden Sie unter Algorithmus zur Bildklassifizierung im Amazon SageMaker 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.

  ```
  {
      "Version": "2012-10-17",
      "Statement": [
          {
              "Effect": "Allow",
              "Action": [
                  "sagemaker:DescribeTrainingJob"
              ],
              "Resource": "arn:aws:sagemaker:region:account-id: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. Weitere Informationen finden Sie unter 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 GreengrassFür die Interaktion mit diesem Connector ist das Machine Learning SDK v1.1.0 erforderlich.

Version 1

- AWS IoT GreengrassCore 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 Core-Gerät installierte Apache MXNet-Framework. Weitere Informationen finden Sie unter the section called “Installieren von MXNet-Abhängigkeiten”.
- Eine ML-Ressource in der Greengrass-Gruppe, die auf eine SageMaker Modellquelle verweist. Dieses Modell muss mit dem SageMaker Bildklassifizierungsalgorithmus trainiert werden. Weitere Informationen finden Sie unter Algorithmus zur Bildklassifizierung im Amazon SageMaker 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.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:DescribeTrainingJob"
            ],
            "Resource": "arn:aws:sagemaker:region:account-id: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. Weitere Informationen finden Sie unter 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 GreengrassFür 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.

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 Trainingsjobs, der die SageMaker Modellquelle darstellt. Das Modell muss mit dem SageMaker 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 ML Image Classification Aarch64 JTX2-Konnektor.

Anzeigename in der Konsole: AWS IoT 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-Konnektordefiniert 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 Trainingsjobs, der die SageMaker Modellquelle darstellt. Das Modell muss mit dem SageMaker 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 ML Image Classification Aarch64 JTX2-Konnektor.

Anzeigename in der Konsole: AWS IoT 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 Aarch64 JTX2-Connectors erstellt, der die GPU-Beschleunigung auf einer NVIDIA Jetson TX2-Karte unterstützt.

```
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 GreengrassMachine Learning SDK, um mit einem ML Image Classification Connector zu interagieren.

> **Note**
>
> Sie können das SDK von der Downloadseite des AWS IoT GreengrassMachine 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	Beschreibung
`AlgoType`	Der Name des Algorithmentyps, der für die Inferenz verwendet werden soll. Derzeit wird nur `image-classification` unterstützt. Erforderlich: `true` Typ: `string` Zulässige Werte: `image-classification`
`ServiceName`	Der Name des lokalen Inferenzdienstes. Verwenden Sie den Namen, den Sie für den Parameter `LocalInferenceServiceName` bei der Konfiguration des Konnektors angegeben haben. Erforderlich: `true` Typ: `string`
`ContentType`	Der Mime-Typ des Eingangsbildes. Erforderlich: `true` Typ: `string` Zulässige Werte: `image/jpeg, image/png`
`Body`	Der Inhalt der Eingabebilddatei. Erforderlich: `true`

Argument	Beschreibung
	Typ: `binary`

Installieren von MXNet-Abhängigkeiten auf dem AWS IoT Greengrass Core

Um einen ML Image Classification Connector zu verwenden, müssen Sie die Abhängigkeiten für das Apache MXNet-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 Core-Gerät installieren müssen.

AWS IoT Greengrass bietet Skripts zum Installieren der Abhängigkeiten für die folgenden gängigen Plattformen und Geräte (oder zur Verwendung als Referenz für die Installation). Wenn Sie eine andere Plattform oder ein anderes Gerät verwenden, lesen Sie die MXNet-Dokumentation für Ihre Konfiguration.

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 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 unterthe 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
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	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 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: `Failed to initialize libdc1394`	Führen Sie den folgenden Befehl aus, um den Treiber zu deaktivieren: `sudo ln /dev/null /dev/raw1394` Diese Operation ist flüchtig, und der symbolisc he Link wird nach dem Neustart verschwinden. Schlagen Sie im Handbuch Ihrer Betriebss 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 for 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 `MLFeedbackConnectorConfigId` 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 Developer Guide

Objekterkennungsstecker

> **Warning**
>
> Dieser Konnektor ist in diePhase der verlängerten Lebensdauer, undAWS IoT Greengrassveröffentlicht keine Updates, die Funktionen, Verbesserungen vorhandener Funktionen, Sicherheitspatches oder Bugfixes bereitstellen. Weitere Informationen finden Sie unter AWS IoT Greengrass Version 1Wartungspolitik.

Die ML-ObjekterkennungSteckverbinderstellen Sie einen Machine Learning (ML) -Inferenzservice zur Verfügung, der auf demAWS IoT Greengrass-Kern. Dieser lokale Inferenzservice führt die Objekterkennung mithilfe eines vom SageMaker Neo Deep Learning Compiler. Zwei Arten von Objekterkennungsmodellen werden unterstützt: Single Shot Multibox Detector (SSD) und Sie schauen nur einmal (YOLO) v3. Weitere Informationen finden Sie unter Objekterkennungsmodell-Anforderungen.

Benutzerdefinierte Lambda-Funktionen verwenden dieAWS IoT GreengrassMachine Learning SDK zur Übermittlung von Inferenzanforderungen an den lokalen Inferenzservice. 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 Greengrassstellt ML-Objekterkennungs-Konnektoren für mehrere Plattformen bereit:

Konnektor	Beschreibung und ARN
ML-Objekterkennung Aarch64 JTX2	Inferenzservice für die Objekterkennung für NVIDIA Jetson TX2. Unterstützt die GPU-Beschleunigung. ARN: `arn:aws:greengrass:` *`region`*`::/connectors/ObjectDetectionAarch64JTX2/versions/1`
ML-Objekterkennung x86_64	Inferenzservice für die Objekterkennung für x86_64-Plattformen.

Konnektor	Beschreibung und ARN
	ARN: `arn:aws:greengrass:` *`region`*`::/connectors/ObjectDetectionx86-64/versions/1`
ML-Objekterkennung armV7	Inferenzservice für die Objekterkennung für ARMv7-Plattformen. ARN: `arn:aws:greengrass:` *`region`*`::/connectors/ObjectDetectionARMv7/versions/1`

Voraussetzungen

Für diese Konnektoren gelten die folgenden Anforderungen:

- AWS IoT GreengrassCore-Software v1.9.3 oder höher.
- PythonVersion 3.7 oder 3.8 auf dem Core-Gerät installiert und der Umgebungsvariablen PATH hinzugefügt.

  > **Note**
  >
  > Um Python 3.8 zu verwenden, führen Sie den folgenden Befehl aus, um einen symbolischen Link vom Standardinstallationsordner für Python 3.8 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 SageMaker Die Neo Deep Learning-Laufzeitumgebung muss auf dem Core-Gerät installiert sein 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 unterAmazon S3 S3-Modellquellenaus.

> **Note**
>
> Das Modell muss ein Objekterkennungsmodell des Typs Single Shot Multibox Detector oder You Only Look Once v3 sein. Es muss mit dem SageMaker Neo Deep Learning Compiler. Weitere Informationen finden Sie unter Objekterkennungsmodell-Anforderungen.

- DieML Feedback-AnschlussDer Greengrass-Gruppe hinzugefügt und konfiguriert wurde. Nur erforderlich, wenn Sie den Konnektor verwenden möchten, um Modelleingabedaten hochzuladen und Prognosen in einem MQTT-Thema zu veröffentlichen.
- AWS IoT GreengrassMachine Learning Learning-SDKVersion 1.1.0 ist für die Interaktion mit diesem Konnektor erforderlich.

Objekterkennungsmodell-Anforderungen

Die ML-Objekterkennungs-Konnektoren unterstützen Objekterkennungsmodelle des Typs 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 worden sein SageMaker Neo Deep Learning Compiler. Stellen Sie beim Kompilieren sicher, dass die Zielhardware mit der Hardware Ihres Greengrass Core-Geräts übereinstimmt. Weitere Informationen finden Sie unter SageMaker NeoimAmazon SageMaker Entwicklerhandbuchaus.

Das kompilierte Modell muss als ML-Ressource hinzugefügt werden (Amazon S3 S3-Modellquelle) zur selben Greengrass-Gruppe wie der Konnektor.

Connector-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.

Anzeigename imAWS IoT-Konsole Modellzielpfad

Erforderlich`true`

Typ: `string`

Gültiges Pattern:`.+`

`MLModelResourceId`

Die ID der ML-Ressource, die auf das Quellmodell verweist.

Anzeigename imAWS IoT-Konsole Greengrass group ML-Ressource

Erforderlich`true`

Typ: `S3MachineLearningModelResource`

Gültiges Pattern:`^[a-zA-Z0-9:_-]+$`

`LocalInferenceServiceName`

Der Name für den lokalen Inferenzdienst. Benutzerdefinierte Lambda-Funktionen rufen den Service auf, indem sie den Namen an den`invoke_inference_service`Funktion desAWS IoT GreengrassMachine Learning Learning-SDK. Ein Beispiel finden Sie unter the section called “Nutzungsbeispiel”.

Anzeigename imAWS IoT-Konsole Lokaler Inferenz-Service-Name

Erforderlich`true`

Typ: `string`

Gültiges Pattern:`^[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 imAWS IoT-Konsole Timeout (Sekunden)

Erforderlich`true`

Typ: `string`

Gültiges Pattern:`^[1-9][0-9]*$`

`LocalInferenceServiceMemoryLimitKB`

Die Speichergröße (in KB), auf die der Service Zugriff hat. Der minimale Wert beträgt 1.

Anzeigename imAWS IoT-Konsole Speicherlimit

Erforderlich`true`

Typ: `string`

Gültiges Pattern:`^[1-9][0-9]*$`

`GPUAcceleration`

Der CPU- oder GPU-(beschleunigte) Berechnungskontext. Diese Eigenschaft gilt nur für den -Konnektor ML Image Classification Aarch64 JTX2.

Anzeigename imAWS 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-Konnektordefiniert ist.

DieserNur erforderlich, wenn Sie den ML-Feedback-Konnektor verwenden möchten, um Modelleingabedaten hochzuladen und Prognosen in einem MQTT-Thema zu veröffentlichen.

Anzeigename imAWS IoT-Konsole ML-Feedback-Konnektor-

Erforderlich`false`

Typ: `string`

Gültiges Pattern:`^$|^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$`

Beispiel für das Erstellen eines Konnektors (AWS CLI)

Der folgende -CLI-Befehl erstellt einen`ConnectorDefinition`mit einer Initialversion, die einen ML-Objekterkennungskonnektor enthält. In diesem Beispiel wird eine Instance des -ARMv7l-Konnektors für die ML-Objekterkennung 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 Anschlüssen hat einelangdauerndesLebenszyklus.

In derAWS IoT Greengrass-Konsole können Sie einen Konnektor aus der Gruppe hinzufügenKonnektorenangezeigten. 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 “Nutzungsbeispiel”.

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. Die Objektkategorien und ihre jeweiligen Werte werden beim Trainieren Ihres Machine Learning-Modells für die Objekterkennung im Neo Deep Learning-Compiler festgelegt.
- 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.

Nutzungsbeispiel

Die folgende Lambda-Beispielfunktion verwendet dieAWS IoT GreengrassMachine Learning Learning-SDKum mit einem ML Object Detection Connector zu interagieren.

> **Note**
>
> Sie können das SDK überAWS IoT GreengrassMachine Learning Learning-SDKDownload-Seite.

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*Das Machine Learning SDK akzeptiert die folgenden Argumente.

Argument	Beschreibung
`AlgoType`	Der Name des Algorithmentyps, der für die Inferenz verwendet werden soll. Derzeit wird nur `object-detection` unterstützt. Erforderlich`true` Typ: `string` Zulässige Werte: `object-detection`

Argument	Beschreibung
`ServiceName`	Der Name des lokalen Inferenzdienstes. Verwenden Sie den Namen, den Sie für den Parameter `LocalInferenceServiceName` bei der Konfiguration des Konnektors angegeben haben. Erforderlich`true` Typ: `string`
`ContentType`	Der Mime-Typ des Eingangsbildes. Erforderlich`true` Typ: `string` Zulässige Werte: `image/jpeg, image/png`
`Body`	Der Inhalt der Eingabebilddatei. Erforderlich`true` Typ: `binary`

Installieren von Neo Deep Learning Runtime-Abhängigkeiten im AWS IoT Greengrass Core

Die ML-Objekterkennungs-Konnektoren sind mit dem SageMaker Neo Deep Learning Runtime (DLR) 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
   ```

3. 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.

4. Führen Sie im 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 im 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 im 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 Problemlösungen

Abhängig von Ihren Gruppeneinstellungen werden Ereignis- und Fehlerprotokolle in geschrieben CloudWatch Protokolle, das lokale Dateisystem oder beides. 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 das SymbolAWS IoT Greengrassgroup ist so konfiguriert, dass lokale Protokolle geschrieben werden, der Konnektor schreibt Protokolldateien in*greengrass-root*`/ggc/var/log/user/`*region*`/aws/`aus. Weitere Informationen zur Greengrass-Protokollierung finden Sie unterthe section called “Überwachen mit AWS IoT Greengrass-Protokollen”aus.

Verwenden Sie die folgenden Informationen für die Problembehandlung bei den -Konnektoren für die ML-Objekterkennung.

Erforderliche Systembibliotheken

Die folgenden Registerkarten listen die für jeden ML-Objekterkennungskonnektor erforderlichen Systembibliotheken auf.

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	n.v.
libcudart.so.9.0	n.v.
libcudnn.so.7	n.v.
libcufft.so.9.0	n.v.
libcurand.so.9.0	n.v.
libcusolver.so.9.0	n.v.
libgcc_s.so.1	GCC_4.2.0
libgomp.so.1	GOMP_4.0, OMP_1.0

Bibliothek	Mindestversion
libm.so.6	GLIBC_2.23
libnvinfer.so.4	n.v.
libnvrm_gpu.so	n.v.
libnvrm.so	n.v.
libnvidia-fatbinaryloader.so.28.2.1	n.v.
libnvos.so	n.v.
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
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: `Failed to initialize libdc1394`	Führen Sie den folgenden Befehl aus, um den Treiber zu deaktivieren: `sudo ln /dev/null /dev/raw1394` Diese Operation ist flüchtig. Der symbolisc he 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 enthalten die folgende Drittanbieter-Software/Lizenz:

- AWS SDK for 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 Konnektor wird unter derLizenzvereinbarung für die Greengrass Core-Softwareaus.

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
- ObjekterkennungsalgorithmusimAmazon SageMaker Entwicklerhandbuch

Modbus-RTU-Protokolladapter -Konnektor

Der Modbus-RTU-ProtokolladapterAnschlussfragt Informationen von Modbus RTU-Geräten ab, die sich im befindenAWS IoT GreengrassGruppe.

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:region::/connectors/ModbusRTUProtocolAdapter/versions/3`
2	`arn:aws:greengrass:region::/connectors/ModbusRTUProtocolAdapter/versions/2`
1	`arn:aws:greengrass:region::/connectors/ModbusRTUProtocolAdapter/versions/1`

Informationen über Versionsänderungen finden Sie im Änderungsprotokoll.

Voraussetzungen

Dieser Konnektor hat die folgenden Anforderungen:

Version 3

- AWS IoT Greengrass Core-Software v1.9.3 oder höher.
- PythonVersion 3.7 oder 3.8 auf dem Core-Gerät installiert und der 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 physikalische Verbindung zwischen demAWS IoT Greengrass-Core und die Modbus-Geräte. Der Core muss physisch über eine serielle Schnittstelle (z. B. eine USB-Schnittstelle) mit dem Modbus RTU-Netzwerk verbunden sein.
- EINlokale Geräteressourcein der Greengrass-Gruppe, der auf die physische serielle Modbus-Schnittstelle zeigt.
- Eine benutzerdefinierte Lambda-Funktion, die Modbus RTU-Anforderungsparameter an diesen Konnektor sendet. Die Anforderungsparameter müssen den erwarteten Mustern entsprechen und die IDs und Adressen der Zielgeräte im Modbus RTU-Netzwerk beinhalten. Weitere Informationen finden Sie unter the section called “Eingabedaten”.

Versions 1 - 2

- AWS IoT GreengrassCore-Software v1.7 oder höher.
- PythonVersion 2.7 auf dem Core-Gerät installiert und der Umgebungsvariablen PATH hinzugefügt.
- Eine physikalische Verbindung zwischen demAWS IoT Greengrass-Core und die Modbus-Geräte. Der Core muss physisch über eine serielle Schnittstelle (z. B. eine USB-Schnittstelle) mit dem Modbus RTU-Netzwerk verbunden sein.
- EINlokale Geräteressourcein der Greengrass-Gruppe, der auf die physische serielle Modbus-Schnittstelle zeigt.
- Eine benutzerdefinierte Lambda-Funktion, die Modbus RTU-Anforderungsparameter an diesen Konnektor sendet. Die Anforderungsparameter müssen den erwarteten Mustern entsprechen und die IDs und Adressen der Zielgeräte im Modbus RTU-Netzwerk beinhalten. Weitere Informationen finden Sie unter the section called “Eingabedaten”.

Connector-Parameter

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.

Anzeigename im AWS IoT-Konsole Modbus-Schnittstelle -Ressource

Erforderlich: `true`

Typ: `string`

Gültiges Pattern: `.+`

`ModbusSerialPort`

Der absolute Pfad zur physikalischen seriellen Modbus-Schnittstelle auf dem Gerät. Dies ist der Quellpfad, der für die lokale Modbus-Geräteressource angegeben ist.

Anzeigename im AWS IoT-Konsole Quellpfad der Ressource der seriellen Modbus-Schnittstelle

Erforderlich: `true`

Typ: `string`

Gültiges Pattern: `.+`

Beispiel für das Erstellen eines Konnektors (AWS CLI)

Der folgende -CLI-Befehl erstellt eine `ConnectorDefinition` mit einer Erstversion, die den Modbus-RTU-Protokolladapter 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 einelanglebiglebenszyklus.

In derAWS IoT Greengrass-Konsole können Sie einen Connector aus demKonnektorenangezeigten. Weitere Informationen finden Sie unter the section called “Beginnen Sie mit Konnektoren (Konsole)”.

> **Note**
>
> Nach der Bereitstellung des Modbus-RTU-Protokolladapters -Konnektors können Sie verwendenAWS IoT Things Graphum 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. Eingabemeldungen 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 des auszuführenden Vorgangs. 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 Datendeduplizierung 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. Zum 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: Spulen lesen

```
{
    "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 eine Ausnahmeantwort wurde zurückgegeben. Weitere Informationen finden Sie unter the section called “Antwortstatus: Exception”.
    - `No Response`. Die Anforderung war ungültig, und der Konnektor hat den Fehler abgefangen, bevor die Anforderung über das Modbus RTU-Netzwerk gesendet wurde. Weitere Informationen finden Sie unter the section called “Antwortstatus: Keine Antwort”.
  - `device`. Das Gerät, an das die Anforderung gesendet wurde.
  - `operation`. Der Anfragetyp, der gesendet wurde.
  - `payload`. Der Antwortinhalt, 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 Datendeduplizierung verwendet wird.

Beispielausgabe: Herzlichen Glückwunsch

```
{
    "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"
}
```

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: Exception

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.
- Die`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"
}
```

Verwendungsbeispiel

Führen Sie die folgenden allgemeinen Schritte aus, um eine Python 3.7—Beispielfunktion Lambda einzurichten, mit der Sie den Konnektor ausprobieren können.

> **Note**
>
> - Wenn Sie andere Python-Laufzeitumgebung verwenden, können Sie einen Symlink von Python 3.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 -Konnektor sendet.

Speichern Sie den Beispielcode als PY-Datei. Downloaden und entpacken Sie esAWS IoT GreengrassCore-SDK für Python. Erstellen Sie dann ein ZIP-Paket, das die PY-Datei und den Ordner `greengrasssdk` auf Stammebene enthält. Dieses ZIP-Paket ist das Bereitstellungspaket, das Sie auf hochladenAWS Lambda.

Nachdem Sie die Python 3.7-Lambda-Funktion erstellt haben, veröffentlichen Sie eine Funktionsversion und erstellen Sie einen Alias.

3. Konfigurieren Sie Ihre Greengrass-Gruppe.

   a. Fügen Sie die Lambda-Funktion über ihren Alias hinzu (empfohlen). Konfiguration des Lambda-Lebenszyklus als langlebig (oder`"Pinned": true`in der CLI).

   b. Fügen Sie die erforderliche lokale Geräteressource 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 zum Anzeigen von Statusmeldungen imAWS IoTconsole.

4. Stellen Sie die Gruppe bereit.

5. In derAWS IoT-Konsole auf derTestdas Thema Ausgabedaten abonnieren, um Statusmeldungen vom Connector anzuzeigen. Die -Beispielfunktion Lambda ist langlebig und beginnt sofort 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) und stellen Sie die Gruppe bereit. Dadurch wird verhindert, dass die Funktion Nachrichten sendet.

Beispiel

Die folgende Lambda-Beispielfunktion sendet eine Eingabemeldung 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 enthält die folgende Drittanbieter-Software/Lizenz:

- pymodbus/BSD
- pyserial/BSD

Dieser Connector wird unter demLizenzvereinbarung für die Greengrass Core-Software.

Änderungsprotokoll

In der folgenden Tabelle werden die wichtigen Änderungen in jeder Version des -Konnektors für beschrieben.

Version	Änderungen
3	Die Lambda-Laufzeitumgebung wurde auf Python 3.7 aktualisiert, was die Laufzeita nforderung ändert.
2	Konnektor-ARN aktualisiert fürAWS-Region-Support. Verbesserte Fehlerprotokollierung.
1	Erstversion.

Eine Greengrass-Gruppe kann nur jeweils eine Version des -Konnektors 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-Protokoll-Adapter-Anschluss

Der Modbus-TCP-ProtokolladapterAnschlusssammelt Daten von lokalen Geräten über das ModBustCP-Protokoll und veröffentlicht sie im ausgewähltenStreamManagerStreams.

Sie können diesen Konnektor auch mit dem IoT verwenden SiteWise Connector und Ihr IoT SiteWise -Gateway. Ihr Gateway muss die Konfiguration für den Connector angeben. Weitere Informationen finden Sie unterKonfigurieren einer Modbus-TCP-Quelleim IoT SiteWise -Benutzerhandbuch.

> **Note**
>
> Dieser Konnektor läuft inKein ContainerIsolationsmodus, damit Sie ihn in einemAWS IoT GreengrassGruppe, die in einem Docker-Container ausgeführt wird.

Dieser Konnektor hat die folgenden Versionen.

Version	ARN
3	`arn:aws:greengrass:region::/connectors/ModbusTCPConnector/versions/3`
2	`arn:aws:greengrass:region::/connectors/ModbusTCPConnector/versions/2`
1	`arn:aws:greengrass:region::/connectors/ModbusTCPConnector/versions/1`

Informationen über Versionsänderungen finden Sie im Änderungsprotokoll.

Voraussetzungen

Dieser Konnektor hat die folgenden Anforderungen:

Version 1 - 3

- AWS IoT GreengrassCore-Software v1.10.2 oder höher.
- Stream-Manager ist aufAWS IoT GreengrassGruppe.
- Java 8 auf dem Core-Gerät installiert und zurPATHUmgebungsvariable.

> **Note**
>
> Dieser Konnektor 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

Connector-Parameter

Dieser Konnektor unterstützt die folgenden Parameter:

`LocalStoragePath`

Das Verzeichnis auf derAWS IoT Greengrasshosten, dass das IoT SiteWise Connector kann persistente Daten in schreiben. Das Standardverzeichnis ist `/var/sitewise`.

Anzeigename imAWS IoT-Konsole Lokaler Speicherpfad

: Erforderlich`false`

Typ: `string`

Gültiges Pattern:`^\s*$|\/.`

`MaximumBufferSize`

Die maximale Größe in GB für IoT SiteWise Festplattennutzung. Die Standardgröße ist 10 GB.

Anzeigename imAWS IoT-Konsole Maximale Datenträgerpuffergröße

: Erforderlich`false`

Typ: `string`

Gültiges Pattern:`^\s*$|[0-9]+`

`CapabilityConfiguration`

Der Satz von Modbus TCP-Collector-Konfigurationen, von denen der Konnektor Daten sammelt und mit denen er eine Verbindung herstellt.

Anzeigename imAWS IoT-Konsole CapabilityConfiguration

: Erforderlich`false`

Type: Eine wohlgeformte JSON-Zeichenfolge, die die Gruppe der unterstützten Feedback-Konfigurationen definiert.

Es folgt ein Beispiel für eine`CapabilityConfiguration`:

```
{
    "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 eine`ConnectorDefinition`mit einer Initialversion, die den Modbus-TCP-Protokoll-Adapter-Konnektor 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 Anschluss hat einelangdauerndesLebenszyklus.

Eingabedaten

Dieser Konnektor akzeptiert keine MQTT-Nachrichten als Eingabedaten.

Ausgabedaten

Dieser Konnektor veröffentlicht Daten in`StreamManager`aus. Sie müssen den Ziel-Message-Stream konfigurieren. Die Ausgabemeldungen haben die folgende Struktur:

```
{
    "alias": "string",
    "messages": [
        {
            "name": "string",
            "value": boolean|double|integer|string,
            "timestamp": number,
            "quality": "string"
        }
    ]
}
```

Lizenzen

Der Modbus-TCP-Protokolladapter-Konnektor enthält die folgende Drittanbieter-Software/Lizenz:

- Digitale PetriModbus

Dieser Connector wird unter derLizenzvereinbarung für die Greengrass Core-Softwareaus.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in jeder Version des -Konnektors beschrieben.

Version	Änderungen	Datum
3 (empfohlen)	Diese Version enthält Fehlerbehebungen.	22. Dezember 2021
2	Unterstützung für ASCII-, UTF8- und ISO8859-codierte Quellzeichenfolgen hinzugefügt.	24. Mai 2021
1	Erstversion.	15. Dezember 2020

Eine Greengrass-Gruppe kann jeweils nur eine Version des -Konnektors 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

> **Warning**
>
> Dieser Konnektor wurde in dieverlängerte Lebensphase, undAWS IoT Greengrassveröffentlicht keine Updates, die Funktionen, Verbesserungen vorhandener Funktionen, Sicherheitspatches oder Fehlerbehebungen bieten. Weitere Informationen finden Sie unter AWS IoT Greengrass Version 1Wartungspolitik.

Der Raspberry Pi-GPIOAnschlusssteuert universell einsetzbare Input/Output (GPIO) -Pins auf einem Raspberry Pi-Core Gerät

Dieser Konnektor fragt Eingangspins in einem bestimmten Intervall ab und veröffentlicht Zustandsänderungen an MQTT-Themen. Es akzeptiert auch Lese- und Schreibanforderungen als MQTT-Nachrichten von benutzerdefinierten Lambda-Funktionen. Schreibanforderungen 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:  region::/connectors/RaspberryPiGPIO/versions/3`
2	`arn:aws:greengrass:  region::/connectors/RaspberryPiGPIO/versions/2`
1	`arn:aws:greengrass:  region::/connectors/RaspberryPiGPIO/versions/1`

Informationen über Versionsänderungen finden Sie im Änderungsprotokoll.

Voraussetzungen

Dieser Konnektor hat die folgenden Anforderungen:

Version 3

- AWS IoT Greengrass Core-Software v1.9.3 oder höher.

- PythonVersion 3.7 auf dem Core-Gerät installiert und der 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”.
- EINlokale Geräteressourcein der Greengrass-Gruppe, die auf zeigt`/dev/gpiomem`auf dem Raspberry Pi. Wenn Sie die Ressource in der -Konsole erstellen, müssen Sie dasAutomatisch Betriebssystemgruppenberechtigungen der Linux-Gruppe hinzufügen, zu der die Ressource gehörtOption. Legen Sie in der API das fest`GroupOwnerSetting.AutoAddGroupOwner`Eigentum zu`true`.
- Das Modul RPi.GPIO, das auf dem Raspberry Pi installiert ist. 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 GreengrassCore-Software v1.7 oder höher.
- PythonVersion 2.7 auf dem Core-Gerät installiert und der 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”.
- EINlokale Geräteressourcein der Greengrass-Gruppe, die auf zeigt`/dev/gpiomem`auf dem Raspberry Pi. Wenn Sie die Ressource in der -Konsole erstellen, müssen Sie dasAutomatisch Betriebssystemgruppenberechtigungen der Linux-Gruppe hinzufügen, zu der die Ressource gehörtOption. Legen Sie in der API das fest`GroupOwnerSetting.AutoAddGroupOwner`Eigentum zu`true`.
- Das Modul RPi.GPIO, das auf dem Raspberry Pi installiert ist. 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 zugrundeliegenden System on Chip (SoC), nicht anhand des physischen Layouts der GPIO-Pins. Die physische Reihenfolge der Pins kann in Raspberry Pi-Versionen variieren. Weitere Informationen finden Sie unterGPIOin 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.

Connector-Parameter

Dieser Konnektor stellt die folgenden Parameter bereit:

`InputGpios`

Eine kommagetrennte 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 derAWS IoT-Konsole Input GPIO Pins

Erforderlich?: `false`. Sie müssen Eingangspins, Ausgangspins oder beides angeben.

Typ: `string`

Gültiges Pattern:`^$|^[0-9]+[UD]?(,[0-9]+[UD]?)*$`

`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 derAWS IoT-Konsole Input GPIO Abrufzeitraum?

Erforderlich?:`false`

Typ: `string`

Gültiges Pattern:`^$|^[1-9][0-9]*$`

`OutputGpios`

Eine kommagetrennte 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 derAWS IoT-Konsole Output GPIO Pins

Erforderlich?: `false`. Sie müssen Eingangspins, Ausgangspins oder beides angeben.

Typ: `string`

Gültiges Pattern:`^$|^[0-9]+[HL]?(,[0-9]+[HL]?)*$`

`GpioMem-ResourceId`

Die ID der lokalen Geräteressource, die `/dev/gpiomem` repräsentiert.

> **Note**
>
> Dieser Konnektor hat Zugriff zum Lesen und Schreiben auf die Ressource.

Anzeigename in derAWS IoT-Konsole Ressource für /dev/gpiomem Gerät

Erforderlich?:`true`

Typ: `string`

Gültiges Pattern:`.+`

Beispiel für das Erstellen eines Konnektors (AWS CLI)

Der folgende -CLI-Befehl erstellt eine`ConnectorDefinition`mit einer Erstversion, die den Raspberry Pi-GPIO -Konnektor 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 einelanglebig-Lebenszyklus.

In derAWS IoT Greengrass-Konsole können Sie einen Connector aus demKonnektorenangezeigten. 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-Konnektor verwendet, einen Wert für mindestens einen der+-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-Konnektor verwendet, einen Wert für mindestens einen der+-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."
}
```

Verwendungsbeispiel

Führen Sie die folgenden allgemeinen Schritte aus, um eine Python 3.7—Lambda—Beispielfunktion einzurichten, mit der Sie den Konnektor ausprobieren können.

> **Note**
>
> - Wenn Sie andere Python-Laufzeitumgebung 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 Konnektor sendet.

   Speichern Sie den Beispielcode als PY-Datei. Downloaden und entpacken Sie dasAWS IoT GreengrassCore-SDK für Python. Erstellen Sie dann ein ZIP-Paket, das die PY-Datei und den Ordner `greengrasssdk` auf Stammebene enthält. Dieses ZIP-Paket ist das Bereitstellungspaket, das Sie auf hochladen.AWS Lambda.

   Nachdem Sie die Python 3.7—Lambda-Funktion erstellt haben, veröffentlichen Sie eine Funktionsversion und erstellen Sie einen Alias.
3. Konfigurieren Sie Ihre Greengrass-Gruppe.

   a. Fügen Sie die Lambda-Funktion über ihren Alias hinzu (empfohlen). Konfiguration des Lambda-Lebenszyklus als langlebig (oder`"Pinned": true`in der CLI).

   b. Fügen Sie die erforderliche lokale Geräteressource 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 zum Anzeigen von Statusmeldungen in derAWS IoTconsole.

4. Stellen Sie die Gruppe bereit.
5. In derAWS IoT-Konsole auf derTestdas Thema Ausgabedaten abonnieren, um Statusmeldungen vom Connector anzuzeigen. Die -Beispielfunktion Lambda ist langlebig und beginnt sofort 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) und stellen Sie die Gruppe bereit. Dadurch wird verhindert, dass die Funktion Nachrichten sendet.

Beispiel

Die folgende Lambda-Beispielfunktion sendet eine Input-Nachricht an den Konnektor. Dieses Beispiel sendet Leseanforderungen für einen Satz von Input-GPIO-Pins. 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_GPIOS = [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_GPIOS:
     read_gpio_state(i)

publish_basic_message()

def lambda_handler(event, context):
    return
```

Lizenzen

Der Raspberry PiO; -Konnektor enthält die folgende Drittanbieter-Software/Lizenz:

- RPi.GPIO/MIT

Dieser Connector wird unter demLizenzvereinbarung für die Greengrass Core-Software.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in jeder Version des Konnektors für beschrieben.

Version	Änderungen
3	Die Lambda-Laufzeitumgebung auf Python 3.7 aktualisiert, wodurch sich die Laufzeita nforderung ändert.
2	Konnektor-ARN für aktualisiertAWS-Region-Unterstützung.
1	Erstversion.

Eine Greengrass-Gruppe kann nur jeweils eine Version des Konnektors 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

Serial stream connector (

> **Warning**
>
> Dieser Konnektor wurde in dieverlängere, undAWS IoT Greengrassveröffentlicht keine Updates, die Funktionen, Verbesserungen vorhandener Funktionen, Sicherheitspatches oder Fehlerbehebungen bieten. Weitere Informationen finden Sie unter AWS IoT Greengrass Version 1Wartungspolitik.

Der Serielle StreamAnschlussliest und schreibt über eine serielle Schnittstelle einesAWS IoT Greengrass-Core-Gerät

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:region::/connectors/SerialStream/versions/3`
2	`arn:aws:greengrass:region::/connectors/SerialStream/versions/2`
1	`arn:aws:greengrass:region::/connectors/SerialStream/versions/1`

Informationen über Versionsänderungen finden Sie im Änderungsprotokoll.

Voraussetzungen

Dieser Konnektor hat die folgenden Anforderungen:

Version 3

- AWS IoT Greengrass Core-Software v1.9.3 oder höher.
- PythonVersion 3.7 oder 3.8 auf dem Core-Gerät installiert und der 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.

- EINlokale Geräteressourcein der Greengrass-Gruppe, der auf die serielle Zielschnittstelle zeigt.

> **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 GreengrassCore-Software v1.7 oder höher.
- PythonVersion 2.7 auf dem Core-Gerät installiert und der Umgebungsvariablen PATH hinzugefügt.
- EINlokale Geräteressourcein der Greengrass-Gruppe, der auf die serielle Zielschnittstelle zeigt.

> **Note**
>
> Bevor Sie diesen Konnektor bereitstellen, empfehlen wir, die serielle Schnittstelle einzurichten und zu überprüfen, ob sie lesbar und beschreibbar ist.

Connector-muster

Dieser Konnektor stellt die folgenden Parameter bereit:

`BaudRate`

Die Baudrate der seriellen Verbindung.

ame inAWS IoT-Konsole Baumuster

Erforderlich`true`

Typ: `string`

Zulässige Werte: `110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 56000, 57600, 115200, 230400`

muster`^110$|^300$|^600$|^1200$|^2400$|^4800$|^9600$|^14400$|^19200$|^28800$|^38400$|^56000$|^57600$|^115200$|^230400$`

`Timeout`

Der Timeout (in Sekunden) für einen Lesevorgang.

ame inAWS IoT-Konsole Timeout (Zeitüberschreitung)

Erforderlich`true`

Typ: `string`

Zulässige Werte: `1 - 59`

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 Geräteressource angegeben ist.

ame inAWS IoT-Konsole Serielle Schnittstelle

Erforderlich`true`

Typ: `string`

muster`[/a-zA-Z0-9_-]+`

`SerialPort-ResourceId`

Die ID der lokalen Geräteressource, die die physische serielle Schnittstelle darstellt.

> **Note**
>
> Dieser Konnektor hat Zugriff zum Lesen und Schreiben auf die Ressource.

ame inAWS IoT-Konsole Serial port resource

Erforderlich`true`

Typ: `string`

muster`[a-zA-Z0-9_-]+`

`PollingRead`

Setzt den Lesemodus: 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.
- Für den Read-On-Demand-Modus geben Sie `false` an. In diesem Modus werden die Werte für Typ und Länge in der Leseanforderung angegeben.

ame inAWS IoT-Konsole Read mode (

Erforderlich`true`

Typ: `string`

Zulässige Werte: `true, false`

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.

ame inAWS IoT-Konsole Polling read length

Erforderlich `false`. Diese Eigenschaft ist erforderlich, wenn`PollingReadist`true.

Typ: `string`

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.

ame inAWS IoT-Konsole Polling read interval

Erforderlich `false`. Diese Eigenschaft ist erforderlich, wenn`PollingReadist`true.

Typ: `string`

Zulässige Werte: 1 - 999

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.

ame inAWS IoT-Konsole muster

Erforderlich `false`. Diese Eigenschaft ist erforderlich, wenn`PollingReadist``true`.

Typ: `string`

Zulässige Werte: `ascii, hex`

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.

ame inAWS IoT-Konsole RTS/CTS-Flusssteuerung

Erforderlich`false`

Typ: `string`

Zulässige Werte: `true, false`

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.

ame inAWS IoT-Konsole Software flow control

Erforderlich`false`

Typ: `string`

Zulässige Werte: `true, false`

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.

ame inAWS IoT-Konsole Serielle Schnittstelle

Erforderlich`false`

Typ: `string`

Zulässige Werte: `N, E, O, S, M`

muster^(|[NEOSMneosm])$

Beispiel für das Erstellen eines Konnektors (AWS CLI)

Der folgende -CLI-Befehl erstellt eine`ConnectorDefinition`mit einer Erstversion, die den Serial Stream-Konnektor 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 derAWS IoT Greengrass-Konsole können Sie einen Connector aus demKonnektorenangezeigten. Weitere Informationen finden Sie unter the section called “Beginnen Sie mit Konnektoren (Konsole)”.

Eingabedaten

Dieser Konnektor akzeptiert Lese- oder Schreibanforderungen für serielle Schnittstellen zu zwei MQTT-Themen. Eingabemeldungen müssen im JSON-Format vorliegen.

- 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`

muster`^[1-9][0-9]*$`

`type`

Der Typ der zu lesenden Daten.

Erforderlich`true`

Typ: `string`

Zulässige Werte: `ascii, hex`

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`

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`

muster`^[1-9][0-9]*$`

`type`

Der Typ der zu lesenden Daten.

Erforderlich`true`

Typ: `string`

Zulässige Werte: `ascii, hex`

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`

muster.+

Beispieleingabe: ASCII-Anfrage

```
{
    "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: Herzlichen Glückwunsch

```
{
    "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"
}
```

Verwendungsmuster

Führen Sie die folgenden allgemeinen Schritte aus, um eine Python 3.7—Lambda-Beispielfunktion einzurichten, mit der Sie den Konnektor ausprobieren können.

> **Note**
>
> - Wenn Sie andere Python -Laufzeitumgebung verwenden, können Sie einen Symlink von Python 3.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 Konnektor sendet.

   Speichern Sie den Beispielcode als PY-Datei. Downloaden und entpacken Sie esAWS IoT GreengrassCore SDK für Python. Erstellen Sie dann ein ZIP-Paket, das die PY-Datei und den Ordner `greengrasssdk` auf Stammebene enthält. Dieses ZIP-Paket ist das Bereitstellungspaket, das Sie auf hochladenAWS Lambda.

   Nachdem Sie die Python 3.7-Lambda-Funktion erstellt haben, veröffentlichen Sie eine Funktionsversion und erstellen Sie einen Alias.
3. Konfigurieren Sie Ihre Greengrass-Gruppe.

   a. Fügen Sie die Lambda-Funktion über ihren Alias hinzu (empfohlen). Konfigurieren Sie den Lambda-Lebenszyklus als langlebig (oder`"Pinned": true`in der CLI).

   b. Fügen Sie die erforderliche lokale Geräteressource 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 zum Anzeigen von Statusmeldungen in derAWS IoTconsole.

4. Stellen Sie die Gruppe bereit.
5. In derAWS IoT-KonsoleTestdas Thema Ausgabedaten abonnieren, um Statusmeldungen vom Connector anzuzeigen. Die Lambda-Beispielfunktion ist langlebig und beginnt sofort 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) und stellen Sie die Gruppe bereit. Dadurch wird verhindert, dass die Funktion Nachrichten sendet.

Beispiel

Die folgende Lambda-Beispielfunktion sendet eine Eingabfrage 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-Konnektor enthält die folgende Drittanbieter-Software/Lizenz:

- pyserial/BSD

Dieser Connector wird unter demLizenzvereinbarung für die Greengrass Core-Software.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in jeder Version des Konnektors beschrieben.

Version	Änderungen
3	Die Lambda-Version wurde auf Python 3.7 aktualisiert, wodurch sich die Laufzeita nforderung ändert.
2	Konnektor-ARN aktualisiert fürAWS-Region-Unterstützung.
1	Erstversion.

Eine Greengrass-Gruppe kann nur jeweils eine Version des Konnektors 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 Connector

> **Warning**
>
> Dieser Konnektor wurde in dieverlängerte Phase, undAWS IoT Greengrassveröffentlicht keine Updates, die Funktionen, Verbesserungen vorhandener Funktionen, Sicherheitspatches oder Fehlerbehebungen bieten. Weitere Informationen finden Sie unter AWS IoT Greengrass Version 1Wartungspolitik.

Die ServiceNow MetricBase IntegrationAnschlussveröffentlicht Zeitreihenmetriken von Greengrass-Geräten für 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 im ServiceNow API in regelmäßigen Abständen.

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 von Zeitservicedaten von Greengrass-Geräten mit benutzerdefinierten Anwendungen, die auf dem ServiceNow platform.

Dieser Konnektor hat die folgenden Versionen.

Version	ARN
4	`arn:aws:greengrass:  region::/connectors/ServiceNowMetricBaseIntegration/versions/4`
3	`arn:aws:greengrass:  region::/connectors/ServiceNowMetricBaseIntegration/versions/3`

Version	ARN
2	`arn:aws:greengrass:region::/connectors/ServiceNowMetricBaseIntegration/versions/2`
1	`arn:aws:greengrass:region::/connectors/ServiceNowMetricBaseIntegration/versions/1`

Informationen über Versionsänderungen finden Sie im Änderungsprotokoll.

Voraussetzungen

Dieser Konnektor hat die folgenden Anforderungen:

Version 3 - 4

- AWS IoT GreengrassCore-Software v1.9.3 oder höher.AWS IoT Greengrassmuss für die Unterstützung lokaler Secrets konfiguriert sein, wie inSecrets Anforderungen.

  > **Note**
  >
  > Diese Anforderung umfasst den Zugriff auf Ihre Secrets Manager Manager-Schlüssel. Wenn Sie die standardmäßige Greengrass-Servicerolle verwenden, hat Greengrass die Berechtigung, die Werte von Secrets mit Namen zu erhalten, die mitgreengrass-.

- PythonVersion 3.7 oder 3.8 auf dem Core-Gerät installiert und der 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 -Konsole mit aktiviertem Abonnement für MetricBase. In :Darüber hinaus muss im Konto eine Metrik und eine metrische Tabelle angelegt werden. Weitere Informationen finden Sie unterMetricBaseim ServiceNow -Dokumentation.
- Ein Texttyp-Secret inAWS Secrets Manager, das den Benutzernamen und das Passwort für Ihre speichert ServiceNow -Instanz mit Standardauthentifizierung. Das Secret muss "Benutzer-" und "Passwort-"Schlüssel mit entsprechenden Werten enthalten. Weitere Informationen finden Sie unterErstellen eines Basis-SecretsimAWS Secrets ManagerBenutzerhandbuch.
- Eine geheime Ressource in der Greengrass-Gruppe, die auf das Secrets Manager verweist. Weitere Informationen finden Sie unter Bereitstellen von Secrets für den Core.

Versions 1 - 2

- AWS IoT GreengrassCore-Software v1.7 oder höher.AWS IoT Greengrassmuss für die Unterstützung lokaler Secrets konfiguriert sein, wie inSecrets Anforderungen.

  Note

  Diese Anforderung umfasst den Zugriff auf Ihre Secrets Manager Manager-Schlüssel. Wenn Sie die standardmäßige Greengrass-Servicerolle verwenden, hat Greengrass die Berechtigung, die Werte von Secrets mit Namen zu erhalten, die mitgreengrass-.

- PythonVersion 2.7 auf dem Core-Gerät installiert und der Umgebungsvariablen PATH hinzugefügt.
- EIN ServiceNow -Konsole mit aktiviertem Abonnement für MetricBase. In :Darüber hinaus muss im Konto eine Metrik und eine metrische Tabelle angelegt werden. Weitere Informationen finden Sie unterMetricBaseim ServiceNow -Dokumentation.
- Ein Texttyp-Secret inAWS Secrets Manager, das den Benutzernamen und das Passwort für Ihre speichert ServiceNow -Instanz mit Standardauthentifizierung. Das Secret muss "Benutzer-" und "Passwort-"Schlüssel mit entsprechenden Werten enthalten. Weitere Informationen finden Sie unterErstellen eines Basis-SecretsimAWS Secrets ManagerBenutzerhandbuch.
- Eine geheime Ressource in der Greengrass-Gruppe, die auf das Secrets Manager verweist. Weitere Informationen finden Sie unter Bereitstellen von Secrets für den Core.

Connector-Parameter

Dieser Konnektor stellt die folgenden Parameter bereit:

Version 4

`PublishInterval`

Die maximale Wartezeit in Sekunden zwischen den Veröffentlichungsereignissen für ServiceNow. Der maximale Wert beträgt 900.

Der -Konnektor veröffentlicht in ServiceNow wenn`PublishBatchSize`erreicht`PublishInterval`Ablauf.

Anzeigename in derAWS IoT-Konsole Veröffentlichungsintervall in Sekunden

Erforderlich:`true`

Typ: `string`

Zulässige Werte: `1 - 900`

Gültig:`[1-9]|[1-9]\d|[1-9]\d\d|900`

`PublishBatchSize`

Die maximale Anzahl von Metrikenwerten, die vor der Veröffentlichung gesammelt werden können ServiceNow.

Der -Konnektor veröffentlicht in ServiceNow wenn`PublishBatchSize`erreicht`PublishInterval`Ablauf.

Anzeigename in derAWS IoT-Konsole Publish Größe

Erforderlich:`true`

Typ: `string`

Gültig:`^[0-9]+$`

`InstanceName`

Der Name der Instance, mit der die Verbindung hergestellt wird ServiceNow.

Anzeigename in derAWS IoT-Konsole Name von ServiceNow Instanz

Erforderlich:`true`

Typ: `string`

Gültig:`.+`

`DefaultTableName`

Der Name der Tabelle, die den`GlideRecord`verbunden mit der Zeitreihe MetricBase Datenbank enthalten. Die Eigenschaft `table` in der Nutzlast der Input-Message kann verwendet werden, um diesen Wert zu überschreiben.

Anzeigename in derAWS IoT-Konsole Der Name der Tabelle, die die Metrik enthält

Erforderlich:`true`

Typ: `string`

Gültig:`.+`

`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 derAWS IoT-Konsole Maximum metrics to retain in memory (Maximale Anzahl

Erforderlich:`true`

Typ: `string`

Gültig:`^[0-9]+$`

`AuthSecretArn`

Das Secret inAWS Secrets Managerdas die ServiceNow -Benutzername und das Passwort. Dies muss ein Texttyp Secret sein. Das Secret muss "Benutzer-" und "Passwort-"Schlüssel mit entsprechenden Werten enthalten.

Anzeigename in derAWS IoT-Konsole ARN of auth secret (ARN of

Erforderlich:`true`

Typ: `string`

Gültig:`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--Secret verweist ServiceNow -Anmeldeinformationen.

Anzeigename in derAWS IoT-Konsole Auth token-Ressource

Erforderlich:`true`

Typ: `string`

Gültig:`.+`

`IsolationMode`

Der Containerisierungsmodus für diesen Konnektor. Der Standardwert ist `GreengrassContainer`. Hierbei wird der Konnektor in einer isolierten Laufzeitumgebung innerhalb des AWS IoT Greengrass-Containers ausgeführt.

> **Note**
>
> Die Standardeinstellung für Containerisierung für die Gruppe gilt nicht für Konnektoren.

Anzeigename in derAWS IoT-Konsole Containerisolierungsmodus

Erforderlich:`false`

Typ: `string`

Gültige Werte: `GreengrassContainer` oder `NoContainer`.

Gültig:`^NoContainer$|^GreengrassContainer$`

Version 1 - 3

`PublishInterval`

Die maximale Wartezeit in Sekunden zwischen den Veröffentlichungsereignissen für ServiceNow. Der maximale Wert beträgt 900.

Der -Konnektor veröffentlicht in ServiceNow wenn`PublishBatchSize`erreicht`PublishInterval`Ablauf.

Anzeigename in derAWS IoT-Konsole Veröffentlichungsintervall in Sekunden

Erforderlich:`true`

Typ: `string`

Zulässige Werte: `1 - 900`

Gültig:`[1-9]|[1-9]\d|[1-9]\d\d|900`

`PublishBatchSize`

Die maximale Anzahl von Metrikenwerten, die vor der Veröffentlichung gesammelt werden können ServiceNow.

Der -Konnektor veröffentlicht in ServiceNow wenn`PublishBatchSize`erreicht`PublishInterval`Ablauf.

Anzeigename in derAWS IoT-Konsole Publish Größe

Erforderlich:`true`

Typ: `string`

Gültig:`^[0-9]+$`

`InstanceName`

Der Name der Instance, mit der die Verbindung hergestellt wird ServiceNow.

Anzeigename in derAWS IoT-Konsole Name von ServiceNow Instanz

Erforderlich:`true`

Typ: `string`

Gültig:`.+`

`DefaultTableName`

Der Name der Tabelle, die den`GlideRecord`verbunden mit der Zeitreihe MetricBase Datenbank enthalten. Die Eigenschaft `table` in der Nutzlast der Input-Message kann verwendet werden, um diesen Wert zu überschreiben.

Anzeigename in derAWS IoT-Konsole Der Name der Tabelle, die die Metrik enthält

Erforderlich:`true`

Typ: `string`

Gültig:`.+`

`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 derAWS IoT-Konsole Maximum metrics to retain in memory (Maximale Anzahl

Erforderlich:`true`

Typ: `string`

Gültig:`^[0-9]+$`

`AuthSecretArn`

Das Secret in AWS Secrets Managerdas die ServiceNow -Benutzername und das Passwort. Dies muss ein Texttyp Secret sein. Das Secret muss "Benutzer-" und "Passwort-"Schlüssel mit entsprechenden Werten enthalten.

Anzeigename in derAWS IoT-Konsole ARN of auth secret (ARN of

Erforderlich:`true`

Typ: `string`

Gültig:`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--Secret verweist ServiceNow -Anmeldeinformationen.

Anzeigename in derAWS IoT-Konsole Auth token-Ressource

Erforderlich:`true`

Typ: `string`

Gültig:`.+`

Beispiel für das Erstellen eines Konnektors (AWS CLI)

Der folgende -CLI-Befehl erstellt eine`ConnectorDefinition`mit einer Erstversion, die den ServiceNow MetricBase Connector für die Integration

```
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 einelangdauerndemLebenszyklus.

In derAWS IoT Greengrass-Konsole können Sie einen Connector aus demKonnektorenangezeigten. Weitere Informationen finden Sie unter the section called “Beginnen Sie mit Konnektoren (Konsole)”.

Eingabedaten

Dieser Konnektor akzeptiert Zeitreihenmetriken zu einem MQTT-Thema und veröffentlicht die Metriken ServiceNow. Eingabebotschaften müssen im JSON-Format vorliegen.

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`, das 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: Herzlichen Glückwunsch

```
{
    "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 Konnektor einen wiederholbaren Fehler erkennt (z. B. Verbindungsfehler), versucht er die Veröffentlichung im nächsten Batch erneut.

Verwendungsbeispiele

Führen Sie die folgenden allgemeinen Schritte aus, um eine Python 3.7-Lambda-Beispielfunktion einzurichten, mit der Sie den Konnektor ausprobieren können.

> **Note**
>
> - Wenn Sie andere Python-Laufzeitumgebung verwenden, können Sie einen Symlink von Python 3.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 Konnektor sendet.

   Speichern Sie den Beispielcode als PY-Datei. Downloaden und entpacken Sie dasAWS IoT GreengrassKern-SDK für Python. Erstellen Sie dann ein ZIP-Paket, das die PY-Datei und den Ordner `greengrasssdk` auf Stammebene enthält. Dieses ZIP-Paket ist das Bereitstellungspaket, das Sie auf hochladen.AWS Lambda.

   Nachdem Sie die Python 3.7-Lambda-Funktion erstellt haben, veröffentlichen Sie eine Funktionsversion und erstellen Sie einen Alias.

3. Konfigurieren Sie Ihre Greengrass-Gruppe.

   a. Fügen Sie die Lambda-Funktion über ihren Alias hinzu (empfohlen). Konfigurieren Sie den Lambda-Lebenszyklus als langlebig (oder`"Pinned": true`in der CLI).

   b. Fügen Sie die erforderliche Secret-Ressource hinzu und gewähren Sie Lesezugriff 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 zum Anzeigen von Statusmeldungen in derAWS IoTconsole.

4. Stellen Sie die Gruppe bereit.

5. In derAWS IoT-Konsole aufTestdas Thema Ausgabedaten abonnieren, um Statusmeldungen vom Connector anzuzeigen. Die Lambda-Beispielfunktion ist langlebig und beginnt sofort 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) und stellen Sie die Gruppe bereit. 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

Die ServiceNow MetricBase Der -Konnektor enthält die folgende Drittanbieter-Software/Lizenz:

- pysnow/MIT

Dieser Connector wird unter demLizenzvereinbarung für die Greengrass Core-Software.

Änderungsprotokoll

In der folgenden Tabelle werden die wichtigen Änderungen in jeder Version für das Konnektor für beschrieben.

Version	Änderungen
4	Der Parameter `IsolationMode` wurde hinzugefügt, um den Containerisierungsmodus für den Konnektor zu konfigurieren.
3	Die Lambda-Laufzeit wurde auf Python 3.7 aktualisiert, wodurch sich die Laufzeita nforderung ändert.

Version	Änderungen
2	Beheben, um übermäßige Protokollierung zu reduzieren.
1	Erstversion.

Eine Greengrass-Gruppe kann nur jeweils eine Version des Konnektors 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-Konnektor

Der SNS-Konnektor veröffentlicht Nachrichten zu einem Amazon 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 in diesem Konnektor veröffentlicht werden.

Dieser Konnektor hat die folgenden Versionen.

Version	ARN
4	`arn:aws:greengrass:region::/connectors/SNS/versions/4`
3	`arn:aws:greengrass:region::/connectors/SNS/versions/3`

Version	ARN
2	`arn:aws:greengrass:  region::/connectors/SNS/versions/2`
1	`arn:aws:greengrass:  region::/connectors/SNS/versions/1`

Informationen über Versionsänderungen finden Sie im Änderungsprotokoll.

Voraussetzungen

Dieser Konnektor hat die folgenden Anforderungen:

Version 3 - 4

- AWS IoT Greengrass Core-Software v1.9.3 oder höher.
- Python Version 3.7 oder 3.8 ist auf dem Core-Gerät installiert und der PATH-Umgebungsvariablen 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, die so konfiguriert ist, dass sie die `-sns:Publish`Aktion für den Amazon SNStopic-Ziel- zulässt, wie in der folgenden Beispiel-IAM-Richtlinie gezeigt.

  ```
  {
      "Version":"2012-10-17",
  ```

```
    "Statement":[
        {
            "Sid":"Stmt1528133056761",
            "Action":[
                "sns:Publish"
            ],
            "Effect":"Allow",
            "Resource":[
                "arn:aws:sns:region:account-id: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 die `sns:Publish` Berechtigung 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. Weitere Informationen finden Sie unter the section called “Verwalten der Gruppenrolle (Konsole)” oder the section called “Verwalten der Gruppenrolle (CLI)”.

Versions 1 - 2

- AWS IoT Greengrass Core-Software v1.7 oder höher.
- Python Version 2.7 wurde auf dem Core-Gerät installiert und der PATH-Umgebungsvariablen hinzugefügt.
- Ein konfiguriertes SNS-Thema. Weitere Informationen finden Sie unter Amazon SNS-Thema anlegen im Amazon Simple Notification Service-Entwicklerhandbuch.
- Die Greengrass-Gruppenrolle, die so konfiguriert ist, dass sie die -`sns:Publish`Aktion für den Amazon SNStopic-Ziel- zulässt, wie in der folgenden Beispiel-IAM-Richtlinie gezeigt.

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
```

```
            "Sid":"Stmt1528133056761",
            "Action":[
                "sns:Publish"
            ],
            "Effect":"Allow",
            "Resource":[
                "arn:aws:sns:region:account-id: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 die `sns:Publish` Berechtigung 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. Weitere Informationen finden Sie unter the section called “Verwalten der Gruppenrolle (Konsole)” oder the section called “Verwalten der Gruppenrolle (CLI)”.

Connector-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-ARN des SNS-Themas

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. Der Standardwert ist `GreengrassContainer`. Hierbei wird der Konnektor in einer isolierten Laufzeitumgebung innerhalb des AWS IoT Greengrass-Containers ausgeführt.

> **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”.

Anzeigename in der AWS IoT Konsole: Standard-ARN des SNS-Themas

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `arn:aws:sns:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):([a-zA-Z0-9-_]+)$`

Beispiel für das Erstellen eines Konnektors (AWS CLI)

Der folgende CLI-Befehl erstellt eine `ConnectorDefinition` mit einer Anfangsversion, die den SNS-Konnektor 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 einen Connector über die Seite Connectors der Gruppe hinzufügen. Weitere Informationen finden Sie unter the section called “Beginnen Sie mit Konnektoren (Konsole)”.

Eingabedaten

Dieser Konnektor akzeptiert SNS-Nachrichteninformationen zu einem MQTT-Thema und veröffentlicht dann die Nachricht unverändert im SNS-Zielthema. 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 enthält:

`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 zeichenfolgenkodiertes 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 zeichenfolgenkodiertes 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"
}
```

Verwendungsbeispiel

Führen Sie die folgenden allgemeinen Schritte aus, 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. Weitere Informationen finden Sie unter 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 Konnektor sendet.

Speichern Sie den Beispielcode als PY-Datei. Laden Sie das AWS IoT Greengrass Core SDK for 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, das Sie in hochladenAWS Lambda.

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 nach 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 Eingabethemafilter.
      - 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 IoTKonsole anzuzeigen.

4. Stellen Sie die Gruppe bereit.
5. Abonnieren Sie in der -AWS IoTKonsole auf der Seite Test das Ausgabedatenthema, um Statusmeldungen vom Konnektor anzuzeigen. Die Lambda-Beispielfunktion ist langlebig und beginnt sofort nach der Bereitstellung der Gruppe Nachrichten zu senden.

   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-Konnektor enthält die folgende Software/Lizenzierung von Drittanbietern:

- AWS SDK for 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 Konnektor wurde gemäß der Greengrass Core Software License Agreement veröffentlicht.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in jeder Version des Konnektors beschrieben.

Version	Änderungen
4	Der Parameter `IsolationMode` wurde hinzugefügt, um den Containerisierungsmodus für den Konnektor zu konfigurieren.
3	Die Lambda-Laufzeit wurde auf Python 3.7 aktualisiert, was die Laufzeitanforderung 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

Splunk Integration Connector

> **Warning**
>
> Dieser Konnektor wurde in dieverlängerte, undAWS IoT Greengrassveröffentlicht keine Updates, die Funktionen, Verbesserungen vorhandener Funktionen, Sicherheitspatches

oder Fehlerbehebungen bieten. Weitere Informationen finden Sie unter AWS IoT Greengrass Version 1Wartungspolitik.

Die Splunk-IntegrationAnschlussveröffentlicht Daten von Greengrass-Geräten für 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.:

- Die Betreiber können periodische Daten von Stellgliedern und Sensoren (z. B. Temperatur-, Druck- und Wasserwerte) verwenden, um Alarme zu initiieren, wenn Werte bestimmte Schwellenwerte bestimmte Schwellenwerte bestimmte Schwellenwerte
- Developer können Daten verwenden, die von Industriemaschinen gesammelt wurden, um ML-Modelle zu entwerfen, die die Anlagen auf mögliche Probleme überwachen.

Dieser Konnektor hat die folgenden Versionen.

Version	ARN
4	`arn:aws:greengrass:region::/connectors/SplunkIntegration/versions/4`
3	`arn:aws:greengrass:region::/connectors/SplunkIntegration/versions/3`
2	`arn:aws:greengrass:region::/connectors/SplunkIntegration/versions/2`

Version	ARN
1	`arn:aws:greengrass:  region::/connectors/SplunkIntegration/versions/1`

Informationen über Versionsänderungen finden Sie im Änderungsprotokoll.

Voraussetzungen

Dieser Konnektor hat die folgenden Anforderungen:

Version 3 - 4

- AWS IoT GreengrassCore-Software v1.9.3 oder höher.AWS IoT Greengrassmuss für die Unterstützung lokaler Secrets konfiguriert sein, wie inSecrets Anforderungen.

  > **Note**
  >
  > Diese Anforderung umfasst den Zugriff auf Ihre Secrets Manager Manager-Schlüssel. Wenn Sie die standardmäßige Greengrass-Servicerolle verwenden, hat Greengrass die Berechtigung, die Werte von Secrets mit Namen zu erhalten, die mitgreengrass.

- PythonVersion 3.7 oder 3.8 auf dem Core-Gerät installiert und der 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 Texttyp Secret in AWS Secrets Manager, der Ihr Splunk HTTP Event Collector Token speichert. Weitere Informationen finden Sie unterInformationen zu Event-Collectorin der Splunk-Dokumentation undErstellen eines Basis-SecretsimAWS Secrets ManagerBenutzerhandbuch.

  > **Note**
  >
  > Um das Secret in der Secrets Manager Manager-Konsole zu erstellen, geben Sie Ihr Token auf derNur-TextTabulator. Verwenden Sie keine Anführungszeichen oder andere Formatierungen. Geben Sie in der API das Token als Wert für das`SecretString`Eigentum.

- Eine geheime Ressource in der Greengrass-Gruppe, die auf das Secrets Manager Manager-Secret verweist. Weitere Informationen finden Sie unter Bereitstellen von Secrets für den Core.

Versions 1 - 2

- AWS IoT GreengrassCore-Software v1.7 oder höher.AWS IoT Greengrassmuss für die Unterstützung lokaler Secrets konfiguriert sein, wie inSecrets Anforderungen.

  > **Note**
  >
  > Diese Anforderung umfasst den Zugriff auf Ihre Secrets Manager Manager-Schlüssel. Wenn Sie die standardmäßige Greengrass-Servicerolle verwenden, hat Greengrass die Berechtigung, die Werte von Secrets mit Namen zu erhalten, die mitgreengrass.

- PythonVersion 2.7 auf dem Core-Gerät installiert und der 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 Texttyp Secret in AWS Secrets Manager, der Ihr Splunk HTTP Event Collector Token speichert. Weitere Informationen finden Sie unterInformationen zu Event-Collectorin der Splunk-Dokumentation undErstellen eines Basis-SecretsimAWS Secrets ManagerBenutzerhandbuch.

> **Note**
>
> Um das Secret in der Secrets Manager Manager-Konsole zu erstellen, geben Sie Ihr Token auf derNur-TextTabulator. Verwenden Sie keine Anführungszeichen oder andere Formatierungen. Geben Sie in der API das Token als Wert für das`SecretString`Eigentum.

- Eine geheime Ressource in der Greengrass-Gruppe, die auf das Secrets Manager Manager-Secret verweist. Weitere Informationen finden Sie unter Bereitstellen von Secrets für den Core.

Connector-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 imAWS IoT-Konsole Splunk-Endpunkt)

Erforderlich`true`

Typ: `string`

Gültiges Pattern:`^(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 imAWS IoT-Konsole Arbeitsspeichergröße

Erforderlich`true`

Typ: `string`

Gültiges Pattern:`^[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 imAWS IoT-Konsole Maximale Anzahl zu erhaltender Elemente

Erforderlich`true`

Typ: `string`

Gültiges Pattern:`^[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 imAWS IoT-Konsole Splunk Veröffentlichungsintervall

Erforderlich`true`

Typ: `string`

Gültiges Pattern:`[0-9]|[1-9]\d|[1-9]\d\d|900`

`SplunkTokenSecretArn`

Das Secret in[AWS Secrets Manager](secretsmanager/latest/userguide/)das den Splunk-Token speichert. Dies muss ein Texttyp Secret sein.

Anzeigename imAWS IoT-Konsole ARN of Splunk auth token secret (ARN des Splunk

Erforderlich`true`

Typ: `string`

Gültiges Pattern:`arn:aws:secretsmanager:[a-z]{2}-[a-z]+-\d{1}:\d{12}?:secret:[a-zA-Z0-9-_]+-[a-zA-Z0-9-_]+`

`SplunkTokenSecretArn-ResourceId`

Die geheime Ressource in der Greengrass-Gruppe, die auf das Splunk-Secret verweist.

Anzeigename imAWS IoT-Konsole Splunk auth token Ressource

Erforderlich`true`

Typ: `string`

Gültiges Pattern:`.+`

`SplunkCustomCALocation`

Der Dateipfad der benutzerdefinierten Zertifizierungsstelle (CA) für Splunk (z. B. `/etc/ssl/certs/splunk.crt`).

Anzeigename imAWS IoT-Konsole Ort der Splunk Custom Certificate Authority

Erforderlich`false`

Typ: `string`

Gültiges Pattern:`^$|/.*`

`IsolationMode`

Der Containerisierungsmodus für diesen Konnektor. Der Standardwert ist `GreengrassContainer`. Hierbei wird der Konnektor in einer isolierten Laufzeitumgebung innerhalb des AWS IoT Greengrass-Containers ausgeführt.

> **Note**
>
> Die Standardeinstellung für Containerisierung für die Gruppe gilt nicht für Konnektoren.

Anzeigename imAWS IoT-Konsole Containerisolierungsmodus

Erforderlich`false`

Typ: `string`

Gültige Werte: `GreengrassContainer` oder `NoContainer`.

Gültiges Pattern:`^NoContainer$|^GreengrassContainer$`

Version 1 - 3

`SplunkEndpoint`

Der Endpunkt Ihrer Splunk-Instance. Dieser Wert muss das Protokoll, den Hostnamen und den Port enthalten.

Anzeigename imAWS IoT-Konsole Splunk-Endpunkt)

Erforderlich`true`

Typ: `string`

Gültiges Pattern:`^(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 imAWS IoT-Konsole Arbeitsspeichergröße

Erforderlich`true`

Typ: `string`

Gültiges Pattern:`^[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 imAWS IoT-Konsole Maximale Anzahl zu erhaltender Elemente

Erforderlich`true`

Typ: `string`

Gültiges Pattern:`^[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 imAWS IoT-Konsole Splunk Veröffentlichungsintervall

Erforderlich`true`

Typ: `string`

Gültiges Pattern:`[0-9]|[1-9]\d|[1-9]\d\d|900`

`SplunkTokenSecretArn`

Das Secret inAWS Secrets Managerdas den Splunk-Token speichert. Dies muss ein Texttyp Secret sein.

Anzeigename imAWS IoT-Konsole ARN of Splunk auth token secret (ARN des Splunk

Erforderlich`true`

Typ: `string`

Gültiges Pattern:`arn:aws:secretsmanager:[a-z]{2}-[a-z]+-\d{1}:\d{12}?:secret:[a-zA-Z0-9-_]+-[a-zA-Z0-9-_]+`

`SplunkTokenSecretArn-ResourceId`

Die geheime Ressource in der Greengrass-Gruppe, die auf das Splunk-Secret verweist.

Anzeigename imAWS IoT-Konsole Splunk auth token Ressource

Erforderlich`true`

Typ: `string`

Gültiges Pattern:`.+`

`SplunkCustomCALocation`

Der Dateipfad der benutzerdefinierten Zertifizierungsstelle (CA) für Splunk (z. B. `/etc/ssl/certs/splunk.crt`).

Anzeigename imAWS IoT-Konsole Ort der Splunk Custom Certificate Authority

Erforderlich`false`

Typ: `string`

Gültiges Pattern:`^$|/.*`

Beispiel für das Erstellen eines Konnektors (AWS CLI)

Der folgende -CLI-Befehl erstellt eine`ConnectorDefinition`mit einer Initialversion, die den Splunk Integration-Konnektor 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 einelangdauerndenLebenszyklus.

In derAWS IoT Greengrass-Konsole können Sie einen Connector aus demKonnektorenangezeigten. Weitere Informationen finden Sie unter the section called “Beginnen Sie mit Konnektoren (Konsole)”.

Eingabedaten

Dieser Konnektor akzeptiert Protokoll- und Ereignisdaten zu einem MQTT-Thema und veröffentlicht die empfangenen Daten an die Splunk-API. Eingabebotschaften 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`

Type: `object`. Nur der`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 Konnektor einen Stapel empfangener Daten an die Splunk-API sendet, veröffentlicht er eine Liste der IDs der erfolgreichen und der 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 Konnektor einen wiederholbaren Fehler erkennt (z. B. Verbindungsfehler), versucht er die Veröffentlichung im nächsten Batch erneut.

Verwendungsbeispiele

Führen Sie die folgenden allgemeinen Schritte aus, 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 Python 3.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 Konnektor sendet.

   Speichern Sie den Beispielcode als PY-Datei. Downloaden und entpacken Sie esAWS IoT GreengrassCore SDK für Python. Erstellen Sie dann ein ZIP-Paket, das die PY-Datei und den Ordner `greengrasssdk` auf Stammebene enthält. Dieses ZIP-Paket ist das Bereitstellungspaket, das Sie auf hochladenAWS Lambda.

Nachdem Sie die Python 3.7-Lambda-Funktion erstellt haben, veröffentlichen Sie eine Funktionsversion und erstellen Sie einen Alias.

3. Konfigurieren Sie Ihre Greengrass-Gruppe.

   a. Fügen Sie die Lambda-Funktion über ihren Alias hinzu (empfohlen). Konfigurieren Sie den Lambda-Lebenszyklus als langlebig (oder`"Pinned": true`in der CLI).

   b. Fügen Sie die erforderliche Secret-Ressource hinzu und gewähren Sie Lesezugriff 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 zum Anzeigen von Statusmeldungen imAWS IoTconsole.

4. Stellen Sie die Gruppe bereit.

5. In derAWS IoTder der der -KonsoleTestdas Thema Ausgabedaten abonnieren, um Statusmeldungen vom Connector anzuzeigen. Die Lambda-Beispielfunktion ist langlebig und beginnt sofort 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) und stellen Sie die Gruppe bereit. 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 wird unter demLizenzvereinbarung für die Greengrass Core-Software.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in jeder Version des Konnektors für beschrieben.

Version	Änderungen
4	Der Parameter `IsolationMode` wurde hinzugefügt, um den Containerisierungsmodus für den Konnektor zu konfigurieren.
3	Lambda -Laufzeitumgebung auf Python 3.7 aktualisiert, wodurch sich die Laufzeita nforderung ändert.
2	Beheben, um übermäßige Protokollierung zu reduzieren.
1	Erstversion.

Eine Greengrass-Gruppe kann nur eine Version des Konnektors gleichzeitig 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)”

Connector für Twilio Benachrichtigungen

> **Warning**
>
> Dieser Konnektor wurde in dieverlängerte Lebensmuster, undAWS IoT Greengrassveröffentlicht keine Updates, die Funktionen, Verbesserungen vorhandener Funktionen, Sicherheitspatches oder Fehlerbehebungen bieten. Weitere Informationen finden Sie unter AWS IoT Greengrass Version 1Wartungspolitik.

Twilio-BenachrichtigungenAnschlussführt automatisierte Telefonate 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**
>
> Ein Tutorial, das die Verwendung des Konnektors für Twilio Benachrichtigungen zeigt, finden Sie unterthe section called “Beginnen Sie mit Konnektoren (Konsole)”oderthe section called “Erste Schritte mit Konnektoren (CLI)”.

Dieser Konnektor hat die folgenden Versionen.

Version	ARN
5	`arn:aws:greengrass:region::/connectors/TwilioNotifications/versions/5`
4	`arn:aws:greengrass:region::/connectors/TwilioNotifications/versions/4`
3	`arn:aws:greengrass:region::/connectors/TwilioNotifications/versions/3`
2	`arn:aws:greengrass:region::/connectors/TwilioNotifications/versions/2`
1	`arn:aws:greengrass:region::/connectors/TwilioNotifications/versions/1`

Informationen über Versionsänderungen finden Sie im Änderungsprotokoll.

Voraussetzungen

Dieser Konnektor hat die folgenden Anforderungen:

Version 4 - 5

- AWS IoT GreengrassCore-Software v1.9.3 oder höher.AWS IoT Greengrassmuss für die Unterstützung lokaler Secrets konfiguriert sein, wie in beschriebenSecrets Anforderungen.

> **Note**
>
> Diese Anforderung umfasst den Zugriff auf Ihre Secrets Manager Manager-Schlüssel. Wenn Sie die standardmäßige Greengrass-Servicerolle verwenden, hat Greengrass die Berechtigung, die Werte von Secrets mit Namen zu erhalten, die mitgreengrass..

- PythonVersion 3.7 oder 3.8 auf dem Core-Gerät installiert und der 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 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, die nicht von Twilio stammen, zu einer Liste verifizierter Telefonnummern hinzufügen. Weitere Informationen finden Sie unterWie Sie mit Ihrem kostenlosen Twilio-Testkonto arbeiten können.

- Ein Texttyp, der in AWS Secrets Manager geheim ist und das Twilio-Authent-Token speichert. Weitere Informationen finden Sie unterErstellen eines Basis-SecretsimAWS Secrets ManagerBenutzerhandbuch.

  > **Note**
  >
  > Um das Secret in der Secrets Manager Manager-Konsole zu erstellen, geben Sie Ihr Token auf derNur-Text-Registerkarten. Verwenden Sie keine Anführungszeichen oder andere Formatierungen. Geben Sie in der API das Token als Wert für das`SecretString`Eigentum.

- 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 GreengrassCore-Software v1.7 oder höher.AWS IoT Greengrassmuss für die Unterstützung lokaler Secrets konfiguriert sein, wie in beschriebenSecrets Anforderungen.

  > **Note**
  >
  > Diese Anforderung umfasst den Zugriff auf Ihre Secrets Manager Manager-Schlüssel. Wenn Sie die standardmäßige Greengrass-Servicerolle verwenden, hat Greengrass die Berechtigung, die Werte von Secrets mit Namen zu erhalten, die mitgreengrass..

- PythonVersion 2.7 auf dem Core-Gerät installiert und der 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 Telefonnummern, die nicht von Twilio stammen, zu einer Liste verifizierter Telefonnummern hinzufügen. Weitere Informationen finden Sie unterWie Sie mit Ihrem kostenlosen Twilio-Testkonto arbeiten können.

- Ein Texttyp, der in AWS Secrets Manager geheim ist und das Twilio-Authent-Token speichert. Weitere Informationen finden Sie unterErstellen eines Basis-SecretsimAWS Secrets ManagerBenutzerhandbuch.

  > **Note**
  >
  > Um das Secret in der Secrets Manager Manager-Konsole zu erstellen, geben Sie Ihr Token auf derNur-Text-Registerkarten. Verwenden Sie keine Anführungszeichen oder andere Formatierungen. Geben Sie in der API das Token als Wert für das`SecretString`Eigentum.

- 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.

Connector-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 -KonsoleAWS IoT-Konsole Twilililililio

Erforderlich:.`true`

Typ: `string`

Gülmuster:`.+`

`TwilioAuthTokenSecretArn`

Der ARN des Secrets Manager Manager-Secrets, das das Twilio Authent-Token speichert.

> **Note**
> Dies wird verwendet, um auf den Wert des lokalen Secrets auf dem Core zuzugreifen.

Anzeigename in der -KonsoleAWS IoT-Konsole ARN of Twilio auth token secret (ARN)

Erforderlich:.`true`

Typ: `string`

Gülmuster:`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 -KonsoleAWS IoT-Konsole Twilio-Authent-Token-Ressource

Erforderlich:.`true`

Typ: `string`

Gülmuster:`.+`

`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 -KonsoleAWS IoT-Konsole Default from phone phone number

Erforderlich:.`false`

Typ: `string`

Gülmuster:`^$|\+[0-9]+`

`IsolationMode`

Der Containerisierungsmodus für diesen Konnektor. Der Standardwert ist `GreengrassContainer`. Hierbei wird der Konnektor in einer isolierten Laufzeitumgebung innerhalb des AWS IoT Greengrass-Containers ausgeführt.

> **Note**
>
> Die Standardeinstellung für Containerisierung für die Gruppe gilt nicht für Konnektoren.

Anzeigename in der -KonsoleAWS IoT-Konsole Containerisolierungsmodus

Erforderlich:.`false`

Typ: `string`

Gültige Werte: `GreengrassContainer` oder `NoContainer`.

Gülmuster:`^NoContainer$|^GreengrassContainer$`

Version 1 - 4

`TWILIO_ACCOUNT_SID`

Die SID des Twilio-Kontos, mit der die Twilio-API aufgerufen wird.

Anzeigename in der -KonsoleAWS IoT-Konsole Twilililililio

Erforderlich:.`true`

Typ: `string`

Gülmuster:`.+`

`TwilioAuthTokenSecretArn`

Der ARN des Secrets Manager Manager-Secrets, das das Twilio Authent-Token speichert.

> **Note**
>
> Dies wird verwendet, um auf den Wert des lokalen Secrets auf dem Core zuzugreifen.

Anzeigename in derAWS IoT-Konsole ARN of Twilio auth token secret (ARN)

Erforderlich:.`true`

Typ: `string`

Gülmuster:`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 derAWS IoT-Konsole Twilio-Authent-Token-Ressource

Erforderlich:.`true`

Typ: `string`

Gülmuster:`.+`

`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 -KonsoleAWS IoT-Konsole Default from phone phone number

Erforderlich:.`false`

Typ: `string`

Gülmuster:`^$|\+[0-9]+`

Beispiel für das Erstellen eines Konnektors (AWS CLI)

Der folgende -CLI-Beispiel-Befehl erstellt eine`ConnectorDefinition`mit einer Initialversion, die Konnektor für Twilio Benachrichtigungen 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 Anleitung zum Hinzufügen des Konnektors für Twilio Benachrichtigungen zu einer Gruppe finden Sie unterthe section called “Erste Schritte mit Konnektoren (CLI)”undthe section called “Beginnen Sie mit Konnektoren (Konsole)”.

Eingabedaten

Dieser Konnektor akzeptiert Twilio-Nachrichteninformationen zu zwei MQTT-Themen. Eingabemeldungen 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`, das beinhaltet die folgenden Eigenschaften:

`recipient`

Der Nachrichtenempfänger. Nur ein Empfänger wird unterstützt.

Erforderlich:.`true`

Typ:`object`, das folgende Eigenschaften beinhaltet:

`name`

Der Name des Empfängers.

Erforderlich:.`true`

Typ: `string`

Gülmuster:`.*`

`phone_number`

Die Telefonnummer des Empfängers.

Erforderlich:.`true`

Typ: `string`

Gülmuster:`\+[1-9]+`

`message`

Der Textinhalt der Textnachricht. Zu diesem Thema werden nur Textnachrichten unterstützt. Für Sprachnachrichten verwenden Sie `twilio/call`.

Erforderlich:.`true`

Typ: `string`

Gülmuster:`.+`

`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`

Gülmuster:`\+[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`

Gülmuster:`.+`

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`, das beinhaltet die folgenden Eigenschaften:

`recipient`

Der Nachrichtenempfänger. Nur ein Empfänger wird unterstützt.

Erforderlich:.`true`

Typ:`object`, das folgende Eigenschaften beinhaltet:

`name`

Der Name des Empfängers.

Erforderlich:.`true`

Typ: `string`

Gülmuster:`.+`

`phone_number`

Die Telefonnummer des Empfängers.

Erforderlich:.`true`

Typ: `string`

Gülmuster:`\+[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`

Gülmuster:`.+`

`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`

Gülmuster:`\+[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`

Gülmuster:`.+`

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: Herzlichen Glückwunsch

```
{
    "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'
    }
}
```

Nutzungsbeispiele

Führen Sie die folgenden allgemeinen Schritte aus, um eine Python 3.7-Lambda-Beispielfunktion einzurichten, mit der Sie den Konnektor ausprobieren können.

> **Note**
>
> Diethe section called “Beginnen Sie mit Konnektoren (Konsole)”undthe section called “Erste Schritte mit Konnektoren (CLI)”Themen enthalten end-to-end Schritte, die Einrichtung, Bereitstellung und das Testen des Konnektors für Twilio Benachrichtigungen zeigen.

1. Stellen Sie sicher, dass Sie die Anforderungen für den Konnektor erfüllen.
2. Erstellen und veröffentlichen Sie eine Lambda-Funktion, die Eingabedaten an den Konnektor sendet.

   Speichern Sie den Beispielcode als PY-Datei. Downloaden und entpacken Sie esAWS IoT GreengrassColio SDK für Python. Erstellen Sie dann ein ZIP-Paket, das die PY-Datei und den Ordner `greengrasssdk` auf Stammebene enthält. Dieses ZIP-Paket ist das Bereitstellungspaket, das Sie auf hochladenAWS Lambda.

   Nachdem Sie die Python 3.7-Lambda-Funktion erstellt haben, veröffentlichen Sie eine Funktionsversion und erstellen Sie einen Alias.
3. Konfigurieren Sie Ihre Greengrass-Gruppe.

a. Fügen Sie die Lambda-Funktion über ihren Alias hinzu (empfohlen). Konfigurieren Sie den Lambda-Lebenszyklus als langlebig (oder`"Pinned": true`in der CLI).

b. Fügen Sie die erforderliche Secret-Ressource hinzu und gewähren Sie Lesezugriff 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 zum Anzeigen von Statusmeldungen in derAWS IoTconsole.

4. Stellen Sie die Gruppe bereit.

5. In derAWS IoT-Konsole auf der -KonsoleTestdas Thema Ausgabedaten abonnieren, um Statusmeldungen vom Connector anzuzeigen. Die Lambda-Beispielfunktion ist langlebig und beginnt sofort 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) und stellen Sie die Gruppe bereit. 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 Konnektor für Twilio Benachrichtigungen enthält die folgende Drittanbieter-Software/Lizenz:

- twilio-python/MIT

Dieser Connector wird unter demLizenzvereinbarung für die Greengrass Core-Software.

Änderungsprotokoll

In der folgenden Tabelle werden die wichtigen Änderungen in jeder Version des -Konnektors für beschrieben.

Version	Änderungen
5	Der Parameter `IsolationMode` wurde hinzugefügt, um den Containerisierungsmodus für den Konnektor zu konfigurieren.

Version	Änderungen
4	Die Lambda-Laufzeit wurde auf Python 3.7 aktualisiert, wodurch sich die Laufzeitanforderung ändert.
3	Beheben, um übermäßige Protokollierung zu reduzieren.
2	Kleinere Fehlerbehebungen und Verbesserungen.
1	Erstversion.

Eine Greengrass-Gruppe kann nur eine Version des -Konnektors gleichzeitig 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

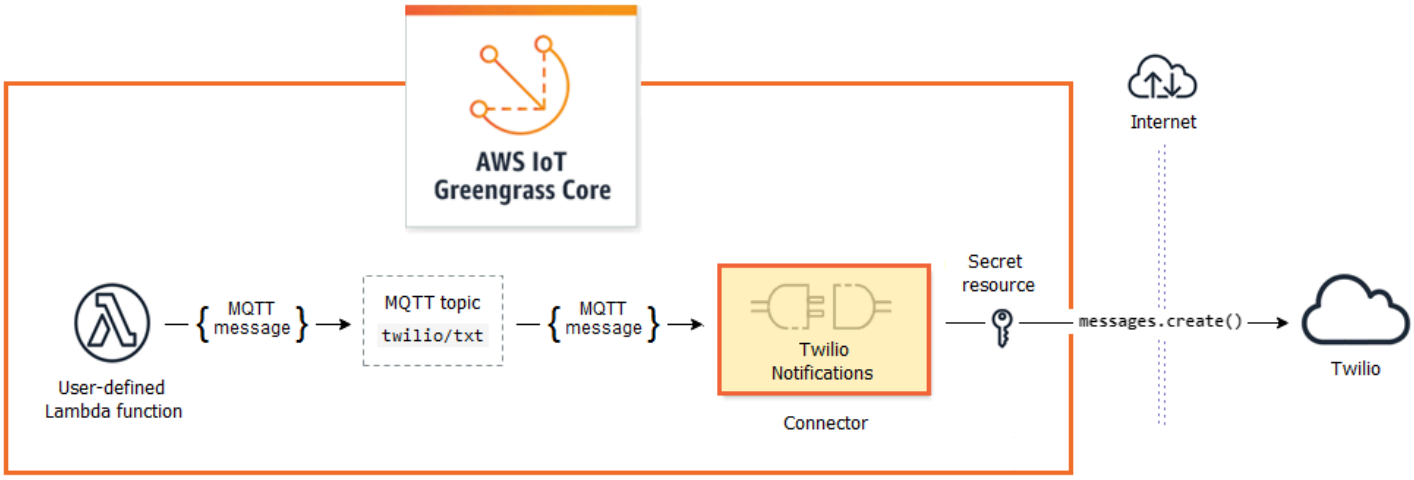
Erste Schritte mit Greengrass-Konnektoren (Konsole)

Diese Funktion ist verfügbar für diese FunktionAWS IoT GreengrassCore v1.7 und höher.

Dieses Tutorial zeigt, wie man die AWS Management Console verwendet, um mit Konnektoren 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 stellen Sie denTwilio-BenachrichtigungenKonnektor. Der Konnektor empfängt Twilio-Nachrichteninformationen als Eingangsdaten und löst dann eine Twilio-SMS aus. Der Datenfluss ist in der folgenden Abbildung dargestellt.



Nach dem Konfigurieren des Konnektors 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-Benachrichtigungs-Konnektor erfordert einen Twilio-Authentifizierungs-Token für die Interaktion mit der Twilio-API. Das Token ist ein Texttyp-Secret, das in AWS Secrets Manager erstellt und von einer Gruppenressource referenziert wird. Dies ermöglicht es AWS IoT Greengrass, eine lokale Kopie des Secrets auf dem Greengrass Core zu erstellen, wo es verschlüsselt und dem Konnektor zur Verfügung gestellt wird. Weitere Informationen finden Sie unter Bereitstellen von Secrets für den Core.

Das Tutorial enthält die folgenden allgemeinen Schritte:

1. Secrets Manager erstellen
2. Hinzufügen einer geheimen Ressource zu einer Gruppe
3. Hinzufügen eines Konnektors zur Gruppe
4. Bereitstellungspaket für die Lambda-Funktion erstellen

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 auf dem AWS IoT Greengrass Core-Gerät.
- AWS IoT Greengrassmuss für die Unterstützung lokaler Secrets konfiguriert sein, wie unterSecrets Anforderungen.

  > **Note**
  >
  > Diese Anforderung umfasst den Zugriff auf Ihre Secrets Manager Manager-Schlüssel. Wenn Sie die standardmäßige Greengrass-Servicerolle verwenden, hat Greengrass die Berechtigung, die Werte von Secrets mit Namen abzurufen, die mitgreengrass-.

- 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, die nicht von Twilio stammen, zu einer Liste verifizierter Telefonnummern hinzufügen. Weitere Informationen finden Sie unterWie Sie mit Ihrem kostenlosen Twilio-Testkonto arbeiten können.

Schritt 1: Secrets Manager erstellen

In diesem Schritt verwenden Sie die AWS Secrets Manager-Console, um ein Texttyp-Secret für Ihren Twilio-Authent-Token zu erstellen.

1. Melden Sie sich an der AWS Secrets Manager-Konsole an.

   > **Note**
   >
   > Weitere Informationen zu diesem Verfahren finden Sie unterSchritt 1: Erstellen und Speichern des Secrets inAWS Secrets ManagerimAWS Secrets ManagerBenutzerhandbuch.

2. Wählen Sie Store a new secret (Ein neues Secret speichern).
3. UnderSecret-Typ wählen, wählenOther type of secret (Andere Art von Secret).
4. UnderAngabe der Schlüssel/Wert-Paare, die für dieses Geheimnis gespeichert werden sollen), auf derNur-Text, geben Sie Ihren Twilio auth token ein. Entfernen Sie alle JSON-Formatierungen und geben Sie nur den Token-Wert ein.
5. Behalten Sieaws/secretsmanagerfür den Verschlüsselungscode ausgewählt und wählen Sie dannWeiter.

   > **Note**
   >
   > Sie werden nicht von belastetAWS KMSwenn Sie die Standardeinstellung verwendenAWSverwalteter Schlüssel, 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 die Greengrass-ServicerolleAWS IoT Greengrassum den Wert von Secrets zu erhalten, deren Namen mit beginnengreengrass-. Weitere Informationen finden Sie unter Anforderungen für Secrets.

7. Dieses Tutorial erfordert keine Rotation. Wählen Sie also Disable automatic rotation (Automatische Rotation deaktivieren) und danachWeiter.
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. In derAWS IoTNavigationsbereich der -Konsole, unterVerwalten, erweitern SieGreengrass-Geräteund wählen Sie dannGruppen (V1).
2. Wählen Sie die Gruppe, zu der Sie die geheime Ressource hinzufügen möchten.
3. Wählen Sie auf der Gruppenkonfigurationsseite die OptionRessourcenund scrollen Sie dann nach unten zur RegisterkarteSecretsAbschnitts erstellt. DieSecretszeigt die geheimen Ressourcen an, die zur Gruppe gehören. In diesem Abschnitt 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. Hierfür können Sie den -Konnektoren verwendenKonfigurationsparameter-Seite oder die Lambda-FunktionRessourcenangezeigten.

4. Klicken Sie aufAddimSecretsAbschnitts erstellt.
5. Auf derHinzufügen einer geheimen RessourceSeite, geben Sie**`MyTwilioAuthToken`**fürRessourcenname.
6. Für denSecret, wählengreengrass-TwilioAuthToken.
7. In derBeschriftungen auswählen (optional)-Abschnitt erstellt AWSCURRENT Staging-Label stellt die neueste Version des Secrets dar. Dieses Label ist immer in einer geheimen Ressource enthalten.

> **Note**
>
> Dieses Tutorial erfordert das AWSCURRENT Nur label (Bezeichnung) Sie können optional Bezeichnungen hinzufügen, die von Ihrer Lambda-Funktion oder vom -Konnektor benötigt werden.

8. Wählen Sie Add resource (Ressource hinzufügen) aus.

Schritt 3: Hinzufügen eines Konnektors zur Greengrass-Gruppe

In diesem Schritt konfigurieren Sie die Parameter für denTwilio-Konnektor fürund fügen Sie es der Gruppe hinzu.

1. Wählen Sie auf der Gruppenkonfigurationsseite die Option Connectors (Konnektoren), und wählen Sie dann Add a connector (Konnektor hinzufügen).
2. Auf derHinzufügen eines Connector-Seite wählen SieTwilio-Benachrichtigungen.
3. Wählen Sie die -Version aus.
4. In derKonfigurationAbschnitts erstellt:
   - FürTwilio auth token-Ressource), geben Sie die Ressource ein, die Sie im vorherigen Schritt erstellt haben.

     > **Note**
     >
     > Wenn Sie die Ressource eingeben, wirdARN des Twilio-auth Token-Secrets-Eigenschaft ist für Sie eingetragen.

   - 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: Bereitstellungspaket für die Lambda-Funktion erstellen

Zum Erstellen einer Lambda-Funktion müssen Sie zunächst eine Lambda-Funktion erstellenBereitstellungspaketdas enthält den Funktionscode und Abhängigkeiten. Greengrass

Lambda-Funktionen erfordern dieAWS IoT GreengrassCore-SDKfür 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. AusAWS IoT GreengrassCore-SDKDownloads-Seite, laden Sie dieAWS IoT GreengrassCore SDK für Python auf Ihrem Computer.
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 das Bereitstellungspaket Ihrer Lambda-Funktion.

Erstellen Sie nun eine Lambda-Funktion, die das Bereitstellungspaket verwendet.

Schritt 5: Erstellen Sie eine Lambda-Funktion im FeldAWS LambdaKonsole

In diesem Schritt verwenden Sie denAWS Lambda-Konsole erstellen Sie eine Lambda-Funktion und konfigurieren diese zur Verwendung des Bereitstellungspakets. Anschließend veröffentlichen Sie eine Funktionsversion und erstellen einen Alias.

1. Zunächst erstellen Sie die Lambda-Funktion.

   a. Wählen Sie in der AWS Management Console Services und öffnen Sie die AWS Lambda-Konsole.

   b. Klicken Sie aufFunktion erstellenund wählen Sie dannAuthor from scratch.

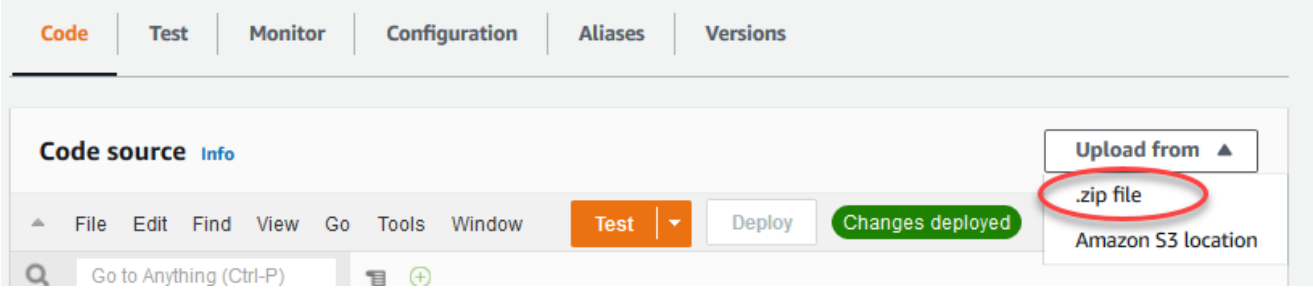
   c. Verwenden Sie im Abschnitt Basic information (Basisinformationen) folgende Werte:

      - Geben Sie für Function name (Funktionsname) **`TempMonitor`** ein.
      - Wählen Sie für Runtime (Laufzeit) die Option Python 3.7 aus.
      - FürBerechtigungenverwenden, behalten Sie die Standardeinstellung bei. Dadurch wird eine Ausführungsrolle erstellt, die grundlegende Lambda-Berechtigungen gewährt. Diese Rolle wird nicht von verwendetAWS IoT Greengrass.

d. Klicken Sie unten auf der Seite auf Create function.

2. Registrieren Sie jetzt den Handler und laden Sie Ihr Bereitstellungspaket für Lambda-Funktion hoch

   a. Auf derCodeTab, unterQuellcode, wählenHochladen von. Klicken Sie in der Dropdown-Liste aufZIP-Datei.



   b. Klicken Sie aufHochladenund wählen Sie dann Ihr`temp_monitor_python.zip`Bereitstellungspaket. Wählen Sie dann Save (Speichern) aus.

   c. Auf derCodefür die Funktion, unterRuntime-Einstellungen, wählenBearbeiten, 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 Save (Speichern) aus.

> **Note**
>
> DieTest-Schaltfläche aufAWS Lambda-Konsole funktioniert nicht mit dieser Funktion. DieAWS IoT GreengrassDas Core SDK enthält keine Module, die erforderlich sind, um Ihre Greengrass Lambda-Funktionen unabhängig imAWS Lambdaconsole. Diese Module (zum Beispiel`greengrass_common`) werden für die Funktionen bereitgestellt, nachdem sie in Ihrem Greengrass-Kern bereitgestellt wurden.

3. Veröffentlichen Sie jetzt die erste Version der Lambda-Funktion und erstellen Sie eineAlias für die Version.

> **Note**
>
> Greengrass-Gruppen können eine Lambda-Funktion per Alias (empfohlen) oder nach Version referenzieren. Mit einem Alias lassen sich Code-Updates einfacher verwalten, da die Abonnementtabelle oder Gruppendefinition nicht geändert werden muss, wenn der Funktionscode aktualisiert wird. Stattdessen verweisen Sie einfach den 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. Auf derTempMonitor: 1-Konfigurationsseite, von derAktionenMenü wählen SieErstellen eines Alias.
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 Greengrassunterstützt keine Lambda-Aliassen für$LATESTVersionen erstellt.

e. Wählen Sie Create (Erstellen) aus.

Jetzt sind Sie bereit, die Lambda-Funktion zu Ihrer Greengrass-Gruppe hinzuzufügen.

Schritt 6: Hinzufügen einer Lambda-Funktion zur Greengrass-Gruppe

In diesem Schritt fügen Sie die Lambda-Funktion der Gruppe hinzu und konfigurieren dann den Lebenszyklus und die Umgebungsvariablen. Weitere Informationen finden Sie unter the section called “Steuern der Ausführung der Greengrass-Lambda-Funktion”.

1. Wählen Sie auf der Gruppenkonfigurationsseite die OptionLambda-FunktionenRegisterkarte.
2. UnderMeine Lambda-Funktionen, wählenAdd.

3. Auf derHinzufügen von Lambda-Funktion-Seite wählen SieTempMonitorfür Ihre Lambda-Funktion.
4. FürVersioning der Lambda-Funktion, wählenAlias: GG_TempMonitor.
5. Klicken Sie aufHinzufügen von Lambda-Funktion.

Schritt 7: Hinzufügen von Abonnements zur Greengrass-Gruppe

In diesem Schritt fügen Sie ein Abonnement hinzu, mit dem die Lambda-Funktion Eingabedaten an den Konnektor senden kann. 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.

In diesem Tutorial erstellen Sie auch Abonnements, die es der Funktion ermöglichen, simulierte Temperaturmessungen von AWS IoT zu empfangen, und dem AWS IoT gestatten, Statusinformationen vom Konnektor zu empfangen.

1. Wählen Sie auf der Gruppenkonfigurationsseite die OptionAbonnementsund wählen Sie dannAbo hinzufügen.
2. Auf derErstellen eines Abonnements-Seite, konfigurieren Sie die Quelle und das Ziel wie folgt:
   a. FürRessourcentyp, wählenLambda-Funktionund wählen Sie dannTempMonitor.
   b. FürZieltyp, wählen-Konnektorund wählen Sie dannTwilio-Benachrichtigungen.
3. Für denThemenfilter, wählen**`twilio/txt`**.
4. Wählen Sie Create subscription (Abonnement erstellen) aus.
5. Wiederholen Sie die Schritte 1 - 4, um ein Abonnement zu erstellen, das es AWS IoT ermöglicht, Nachrichten an die Funktion zu veröffentlichen.
   a. FürRessourcentyp, wählenServiceund wählen Sie dannIoT Cloud.
   b. FürWählen Sie ein Ziel aus, wählenLambda-Funktionund wählen Sie dannTempMonitor.
   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. FürRessourcentyp, wählen-Konnektorund wählen Sie dannTwilio-Benachrichtigungen.
   b. FürZieltyp, wählenServiceund wählen Sie dannIoT Cloud.

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 dieAWS IoT Greengrasscore 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 in dem Pfad hängt von der AWS IoT Greengrass-Core-Softwareversion ab, die auf Ihrem Core-Gerät installiert ist.

   b. So starten Sie den Daemon:

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

2. Wählen Sie auf der Gruppenkonfigurationsseite die OptionBereitstellen.
3. a. In derLambda-Funktionenunter der RegisterkarteSystem-Lambda-FunktionenAbschnitt, wählen SieIP-Detektorund wähleBearbeiten.

   b. In derEinstellungen für IP-Detektor bearbeiten-Dialogfeld wählen SieAutomatisches Erkennen und Überschreiben von MQTT-Broker-.

   c. Wählen Sie Save (Speichern) 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.

> **Note**
>
> Erteilen Sie bei Aufforderung die Berechtigung zum Erstellen desGreengrass-Servicerolleund assoziiere es mit deinemAWS-Kontoim aktuellenAWS-Region. Diese Rolle erlaubtAWS IoT Greengrassum auf Ihre -Ressourcen in zuzugreifenAWS-Services.

Auf der Seite Deployments werden der Zeitstempel, die Versions-ID und der Status der Bereitstellung angegeben. Nach abgeschlossener Bereitstellung sollte der StatusCompleted (Abgeschlossen).

Hilfe zur Problembehebung finden Sie unter Fehlerbehebung.

> **Note**
>
> Eine Greengrass-Gruppe kann nur eine Version des Konnektors gleichzeitig enthalten. Weitere Informationen zum Aktualisieren einer Konnektorversion finden Sie unter the section called “Aktualisieren von Konnektorversionen”.

Testen der Lösung

1. Auf derAWS IoT-Konsolen-Startseite, wählen SieTest.
2. FürAbonnieren eines -Themas, verwenden Sie die folgenden Werte und wählen Sie dannAbonnieren. Die Statusinformationen zu diesem Thema veröffentlicht der Twilio-Benachrichtigungs-Konnektor.

Property (Eigenschaft)	Wert
Abonnementthema	twilio/message/status
MQTT-Nutzlast-Anzeige	Zeigt Nutzlasten als Zeichenfolgen an

3. FürVeröffentlichung für ein Thema, verwenden Sie die folgenden Werte und wählen Sie dannVeröffentlichenum die Funktion aufzurufen.

Property (Eigenschaft)	Wert
Topic	Temperatur/Input
Fehlermeldung	Ersetzen*Empfängername* mit einem Namen und*recipient-phone-number* mit der Telefonnummer des Empfängers der SMS. Beispiel: `+12345000000` `{ "to_name": " recipient-name ", "to_number": " recipient-phone-number ", "temperature": 31 }` Wenn Sie ein Testkonto verwenden, müssen Sie Telefonnummern, die nicht von Twilio stammen, zu einer Liste verifizierter Telefonnummern hinzufügen. Weitere Informationen finden Sie unterÜberprüfen Ihrer persönlichen 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 dies weniger als 30 ist, ist der TempMonitor -Funktion löst keine Twilio-Nachricht aus.

Weitere Informationen finden Sie auch unter

- Integrieren von Services und Protokollen mit Konnektoren
- the section called “AWSVon bereitgestellte Greengrass-Konnektoren”

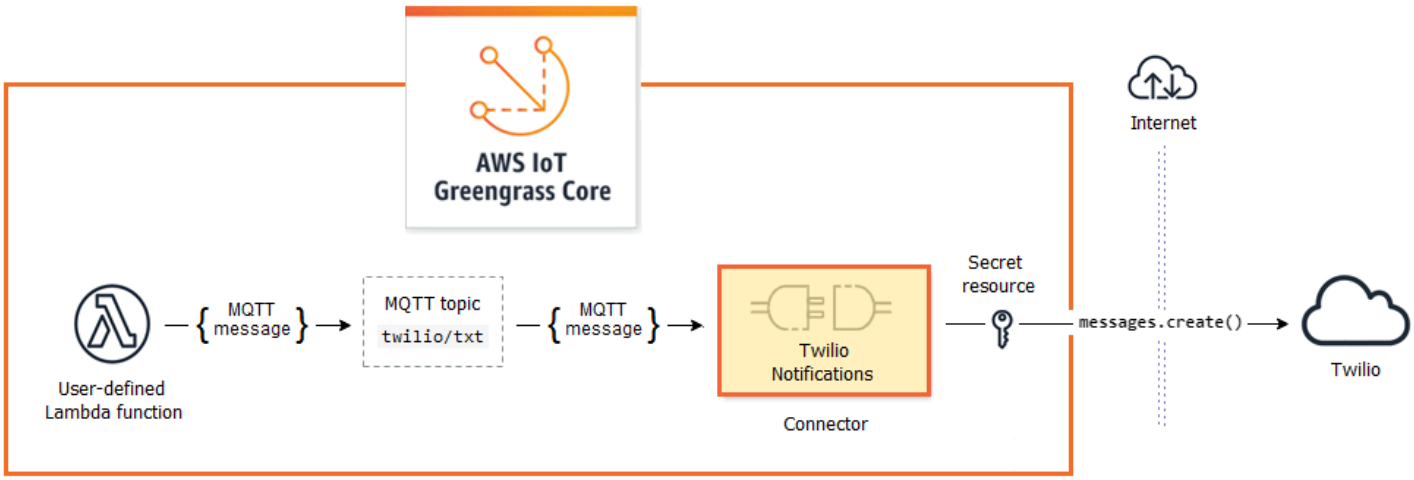
Erste Schritte mit Greengrass-Konnektoren (CLI)

Diese Funktion ist verfügbar fürAWS IoT GreengrassCore v1.7 und höher.

Dieses Tutorial zeigt, wie man die AWS CLI verwendet, um mit Konnektoren 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 stellen Sie den bereitTwilio-Benachrichtigungen-Konnektor Der Konnektor empfängt Twilio-Nachrichteninformationen als Eingangsdaten und löst dann eine Twilio-SMS aus. Der Datenfluss ist in der folgenden Abbildung dargestellt.



Nach dem Konfigurieren des Konnektors 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-Benachrichtigungs-Konnektor erfordert einen Twilio-Authentifizierungs-Token für die Interaktion mit der Twilio-API. Das Token ist ein Texttyp-Secret, das in AWS Secrets Manager erstellt

und von einer Gruppenressource referenziert wird. Dies ermöglicht es AWS IoT Greengrass, eine lokale Kopie des Secrets auf dem Greengrass Core zu erstellen, wo es verschlüsselt und dem Konnektor zur Verfügung gestellt wird. Weitere Informationen finden Sie unter Bereitstellen von Secrets für den Core.

Das Tutorial enthält die folgenden allgemeinen Schritte:

1. Secret-Manager-Secret
2. Erstellen einer Ressourcendefinition und -version
3. Erstellen einer Konnektordefinition und -version
4. Erstellen eines Bereitstellungspakets für Lambda
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.

Verwenden der AWS IoT Greengrass-API

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 beim Ausführen eines AWS CLI-Befehls einen Fehler erhalten, fügen Sie den Parameter `--debug` hinzu und führen Sie den Befehl erneut aus, um weitere Informationen über den 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 auf dem AWS IoT Greengrass Core-Gerät.
- AWS IoT Greengrassmuss für die Unterstützung lokaler Secrets konfiguriert sein, wie inSecrets Anforderungenaus.

  > **Note**
  >
  > Diese Anforderung umfasst den Zugriff auf Ihre Secrets-Manager-Secrets. Wenn Sie die standardmäßige Greengrass-Servicerolle verwenden, hat die Berechtigung, die Werte von Secrets mit Namen zu erhalten, die mit beginnengreengrassaus.

- 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, die nicht von Twilio stammen, zu einer Liste verifizierter

Telefonnummern hinzufügen. Weitere Informationen finden Sie unterSo arbeiten Sie mit Ihrem kostenlosen Twilio-Testkontoaus.

- Die AWS CLI ist auf Ihrem Computer installiert und konfiguriert. Weitere Informationen finden Sie unterInstallieren vonAWS Command Line InterfaceundKonfigurieren vonAWS CLIimAWS Command Line Interface-Benutzerhandbuchaus.

Die Beispiele in diesem Tutorial sind für Linux und andere Unix-basierte Systeme geschrieben. Wenn Sie Windows verwenden, lesen Sie Folgendes:Angeben von Parameterwerten für dieAWS Command Line Interfaceum mehr über Unterschiede in der Syntax zu erfahren.

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: Secret-Manager-Secret

In diesem Schritt verwenden Sie die AWS Secrets Manager-API, um ein Secret für Ihren Twilio-Authent-Token zu erstellen.

1. Erstellen Sie zunächst das Secret.

   - Ersetzen*twilio-auth-token*mit Ihrem Twilio-Authent-Token.

```
aws secretsmanager create-secret --name greengrass-TwilioAuthToken --secret-
string twilio-auth-token
```

> **Note**
>
> Die Greengrass-Servicerolle ermöglicht standardmäßigAWS IoT Greengrassum den Wert von Secrets mit Namen zu erhalten, die mitgreengrassaus. Weitere Informationen finden Sie unter Anforderungen für Secrets.

2. Kopieren Sie den ARN des Secrets aus der Ausgabe. Damit erstellen Sie die geheime Ressource und konfigurieren den Twilio-Benachrichtigungs-Konnektor.

Schritt 2: Erstellen einer Ressourcendefinition und -version

In diesem Schritt verwenden Sie denAWS IoT GreengrassAPI zum Erstellen einer geheimen Ressource für Ihr Secret-Manager-Secret

1. Erstellen Sie eine Ressourcendefinition, die eine Initialversion enthält.

   - Ersetzen Sie *secret-arn* durch den ARN des Secrets, das Sie im vorherigen Schritt kopiert haben.

   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 die Parameter für den Twilio Notifications -Konnektor.

1. Erstellen Sie eine Konnektordefinition mit einer Initialversion.

   - Ersetzen Sie *`account-sid`* durch Ihre Twilio-Konto-SID.
   - Ersetzen*`geheim-arn`*mit demARNIhres Secrets-Manager-Secrets Der Konnektor verwendet dies, um den Wert des lokalen Secrets zu erhalten.
   - Ersetzen Sie *`phone-number`* durch Ihre Twilio-aktivierte 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 eines Bereitstellungspakets für Lambda

Zum Erstellen einer Lambda-Funktion müssen Sie zuerst eine Lambda-Funktion erstellenBereitstellungspaketdas den Funktionscode und Abhängigkeiten enthält. Greengrass Lambda-Funktionen erfordern dieAWS IoT GreengrassCore-SDKfür 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. Aus demAWS IoT GreengrassCore-SDKDownloads-Seite, laden Sie dieAWS IoT GreengrassCore SDK für Python auf Ihrem Computer.
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 das Bereitstellungspaket Ihrer Lambda-Funktion.

Schritt 5: Erstellen einer Lambda-Funktion

Erstellen Sie nun eine Lambda-Funktion, die das Bereitstellungspaket verwendet.

1. Erstellen Sie eine IAM-Rolle, damit Sie beim Anlegen der Funktion den Rollen-ARN 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 Greengrassverwendet diese Rolle nicht, da 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 derAWS LambdaAPI zum Erstellen der TempMonitor Funktion. Der folgende Befehl geht davon aus, dass sich die Zip-Datei im aktuellen Verzeichnis befindet.

   - Ersetzen Sie *role-arn* durch den kopierten `Arn`.

   ```
   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 über einen Alias (empfohlen) oder nach Version referenzieren. Mit einem Alias lassen sich Code-Updates einfacher verwalten, da die Abonnementtabelle oder Gruppendefinition nicht geändert werden muss, wenn der Funktionscode aktualisiert wird. Stattdessen verweisen Sie einfach auf den Alias auf die neue Funktionsversion.

   > **Note**
   >
   > AWS IoT Greengrassunterstützt 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. Diesen Wert verwenden Sie bei der Konfiguration der Funktion für AWS IoT Greengrass und beim Anlegen eines Abonnements.

Jetzt sind Sie bereit, die Funktion für AWS IoT Greengrass zu konfigurieren.

Schritt 6: Erstellen einer Funktionsdefinition und -version

So verwenden Sie eine Lambda-Funktion für eineAWS IoT Greengrasscore erstellen Sie eine Funktionsdefinitionsversion, die die Lambda-Funktion über einen Alias referenziert und die Konfiguration auf Gruppenebene definiert. Weitere Informationen finden Sie unter the section called “Steuern der Ausführung der Greengrass-Lambda-Funktion”.

1. Erstellen Sie eine Funktionsdefinition, die eine Initialversion enthält.

   - Ersetzen Sie *alias-arn* durch den `AliasArn`, den Sie beim Erstellen 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, mit dem die Lambda-Funktion Eingabedaten an den -Konnektor senden kann. 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.

In diesem Tutorial erstellen Sie auch Abonnements, die es der Funktion ermöglichen, simulierte Temperaturmessungen von AWS IoT zu empfangen, und dem AWS IoT gestatten, Statusinformationen vom Konnektor zu empfangen.

1. Erstellen Sie eine Abonnementdefinition, die eine Initialversion enthält, die die Abonnements umfasst.

   - Ersetzen Sie *`alias-arn`* durch den `AliasArn`, den Sie beim Erstellen des Alias für die Funktion kopiert 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 die IDs der Greengrass-Zielgruppen und die Gruppenversion ab. Dieses Verfahren setzt voraus, dass es sich um die neueste Gruppe 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 inAWS IoTconsole. Die Gruppen-ID wird auf der Seite Einstellungen der Gruppe angezeigt. Gruppenversions-IDs werden auf denBereitstellungenRegisterkarte.

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.

- Ersetzen Sie *`group-id`* durch die kopierte `Id` für die Gruppe.
- Ersetzen*`group-version-id`*mit dem`LatestVersion`die du für die Gruppe kopiert hast.

```
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

- Ersetzen Sie *`group-id`* durch die kopierte `Id` für die Gruppe.

- Ersetzen*core-definition-version-arn*mit demCoreDefinitionVersionArndie Sie für die Core-Definitionsversion kopiert haben.
- Ersetzen*resource-definition-version-arn*mit demLatestVersionArndie Sie für die Rollenressourcendefinition kopiert haben.
- Ersetzen*connector-definition-version-arn*mit demLatestVersionArndie Sie für die Konnektordefinition kopiert haben.
- Ersetzen*function-definition-version-arn*mit demLatestVersionArndie Sie für die Funktionsdefinition kopiert haben.
- Ersetzen*subscription-definition-version-arn*mit demLatestVersionArndie 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: Eine Bereitstellung auswählen

Stellen Sie die Gruppe auf dem Core-Gerät bereit.

1. Stellen Sie in einem Core-Gerät-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. So starten Sie den Daemon:

   ```
   cd /greengrass/ggc/core/
   ```

```
sudo ./greengrassd start
```

2. Erstellen einer Bereitstellung.

   - Ersetzen Sie *group-id* durch die kopierte `Id` für die Gruppe.
   - Ersetzen*group-version-id*mit dem`Version`ondie Sie für die neue Gruppe 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.

   - Ersetzen Sie *group-id* durch die kopierte `Id` für die Gruppe.
   - Ersetzen Sie *deployment-id* durch die `DeploymentId`, die 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. Auf derAWS IoT-Konsolen-Startseite, wählen SieTestaus.
2. FürAbonnieren eines -Themas, verwenden Sie die folgenden Werte und wählen SieAbonnierenaus. Der Twilio-Benachrichtigungs-Konnektor veröffentlicht Statusinformationen zu diesem Thema.

Property (Eigenschaft)	Wert
Abonnementthema	twilio/message/status

Property (Eigenschaft)	Wert
MQTT-Nutzlast-Anzeige	Zeigt Nutzlasten als Zeichenfolgen an

3. FürVeröffentlichung für ein Thema, verwenden Sie die folgenden Werte und wählen SieVeröffentlichenum die Funktion aufzurufen.

Property (Eigenschaft)	Wert
Topic	Temperatur/Input
Fehlermeldung	Ersetzen*Empfängername* mit einem Namen und*recipient-phone-number* mit der Telefonnummer des Empfängers der SMS. Beispiel: `+12345000000` `{ "to_name": " recipient-name ", "to_number": " recipient-phone-number ", "temperature": 31 }` Wenn Sie ein Testkonto verwenden, müssen Sie Telefonnummern, die nicht von Twilio stammen, zu einer Liste verifizierter Telefonnummern hinzufügen. Weitere Informationen finden Sie unterÜberprüfen Sie Ihre persönliche Telefonnummeraus.

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 dies weniger als 30 ist, TempMonitor -Funktion löst keine Twilio-Nachricht aus.

Weitere Informationen finden Sie auch unter

- Integrieren von Services und Protokollen mit Konnektoren
- the section called “AWSVon bereitgestellte Greengrass-Konnektoren”
- the section called “Beginnen Sie mit Konnektoren (Konsole)”
- AWS Secrets ManagerKommandosimAWS CLIBefehlsreferenz
- AWS Identity and Access Management(IAM) -BefehleimAWS CLIBefehlsreferenz
- AWS LambdaKommandosimAWS CLIBefehlsreferenz
- AWS IoT GreengrassKommandosimAWS CLIBefehlsreferenz

RESTful-API für Greengrass Discovery

Alle Client-Geräte, die mit einemAWS 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-Kern erforderlich sind, der sich in derselben Greengrass-Gruppe wie das Client-Gerät befindet. Wenn ein Client-Gerät zum ersten Mal online geht, kann es eine Verbindung zumAWS IoT Greengrass Dienst herstellen und die Discovery-API verwenden, um Folgendes zu finden:

- 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 dieAWS IoT Device SDKs auch zum Auffinden von Verbindungsinformationen für einen Greengrass Core verwenden. 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 dieAWS IoT Greengrass Discovery-API finden Sie unter AWS IoT GreengrassEndpunkte und Kontingente in der Allgemeine AWS-Referenz. Dies ist nur eine Datenebenen-API. Die Endpunkte für die Gruppenverwaltung und AWS IoT Core-Operationen unterscheiden sich von den Discovery-API-Endpunkten.

Anfrage

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` als die`ProtocolName` in übergeben`ProtocolNameList`. Weitere Informationen finden Sie unter Protokolle im AWS IoTEntwicklerhandbuch.

> **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 “Service-Endpunkte müssen mit dem Zertifikattyp übereinstimmen”.

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.

Berechtigung zum Discovery

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:

```
{
    "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-Kern, 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-Kern, 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"
            }
          ]
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
      ]
    },
    {
      "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-Service, die eine Liste von Greengrass Cores enthält, enthält nur einen einzigen Greengrass Core.

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-2:1234567
89012: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

Die Sicherheit in der Cloud hat bei AWS höchste Priorität. Als AWS-Kunde profitieren Sie von einer Rechenzentrums- und Netzwerkarchitektur, die zur Erfüllung der Anforderungen von Organisationen entwickelt wurden, für die Sicherheit eine kritische Bedeutung hat.

Sicherheit ist eine übergreifende Verantwortlichkeit zwischen AWS und Ihnen. Das Modell der geteilten Verantwortung beschreibt dies als Sicherheit der Cloud selbst und als Sicherheit in der Cloud:

- Sicherheit der Cloud – AWS ist dafür verantwortlich, die Infrastruktur zu schützen, mit der AWS-Services in der AWS Cloud ausgeführt werden. AWS stellt Ihnen außerdem Services bereit, die Sie sicher nutzen können. Auditoren von Drittanbietern testen und überprüfen die Effektivität unserer Sicherheitsmaßnahmen im Rahmen der AWS-Compliance-Programme regelmäßig. Informationen zu den Compliance-Programmen, die für AWS IoT Greengrass gelten, finden Sie unter Im Rahmen des Compliance-Programms zugelassene AWS-Services.
- 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.

Wenn Sie AWS IoT Greengrass verwenden, sind Sie auch für die Sicherung Ihrer Geräte, der lokalen Netzwerkverbindung und der privaten Schlüssel verantwortlich.

Diese Dokumentation hilft Ihnen zu verstehen, wie Sie das Modell der geteilten Verantwortung bei der Verwendung von AWS IoT Greengrass einsetzen können. Die folgenden Themen veranschaulichen, wie Sie AWS IoT Greengrass zur Erfüllung Ihrer Sicherheits- und Compliance-Ziele konfigurieren können. Sie erfahren außerdem, wie Sie andere AWS-Services verwenden, um Ihre AWS IoT Greengrass-Ressourcen zu überwachen und zu schützen.

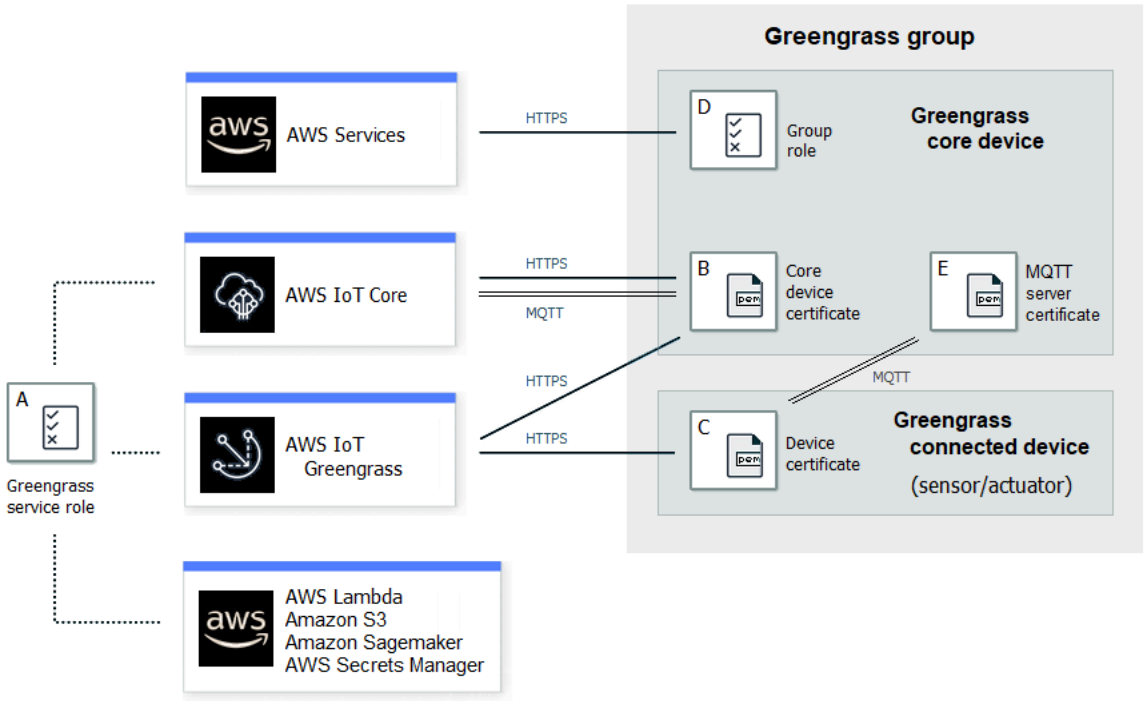
Themen

- Ausfallsicherheit in AWS IoT Greengrass
- Infrastruktursicherheit in AWS IoT Greengrass
- Konfigurations- und Schwachstellenanalyse in AWS IoT Greengrass
- AWS IoT Greengrass und Schnittstellen-VPC-Endpunkte (AWS PrivateLink)
- Bewährte Methoden für die Sicherheit 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 “Berechtigung zum Discovery”.

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 Core-Gerät.

> **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 Core-Konfigurationsdatei”.

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 rotiert den Schlüssel basierend auf Ihren 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 Nachrichtenflusses 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 finden 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
TLSv1.2	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_WITH_AES_128_CBC_SHA	Nicht unterstützt	1,0
TLS_RSA_WITH_AES_128_GCM_SHA256	Nicht unterstützt	1,0
TLS_RSA_WITH_AES_256_CBC_SHA	Nicht unterstützt	1,0
TLS_RSA_WITH_AES_256_GCM_SHA384	Nicht unterstützt	1,0
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	Unterstützt	1.9
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	Unterstützt	1.9

	Verschlüsselungsverfahren	HSM-Support	GGC-Mindestversion
TLSv1.1	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	Unterstützt	1,0
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	Unterstützt	1,0
	TLS_RSA_WITH_AES_128_CBC_SHA	Nicht unterstützt	1,0
	TLS_RSA_WITH_AES_256_CBC_SHA	Nicht unterstützt	1,0
TLSv1.0	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	Unterstützt	1,0
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	Unterstützt	1,0
	TLS_RSA_WITH_AES_128_CBC_SHA	Nicht unterstützt	1,0
	TLS_RSA_WITH_AES_256_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 im Abschnitt Datenschutz FAQ. Informationen zum Datenschutz in Europa finden Sie im AWS Shared Responsibility Model und im GDPR Blogbeitrag auf dem AWS Security Blog.

Aus Datenschutzgründen empfehlen wir, dass Sie Ihre 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 eine Multi-Faktor-Authentifizierung (MFA).
- Verwenden SieSSL/TLS, um mit AWS Ressourcen zu kommunizieren. Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Einrichtung API und Protokollierung von Benutzeraktivitäten mit AWS CloudTrail.
- 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 FIPS 140-3 validierte kryptografische Module für den Zugriff AWS über eine Befehlszeilenschnittstelle oder eine benötigenAPI, verwenden Sie einen Endpunkt. FIPS Weitere Informationen zu den verfügbaren FIPS Endpunkten finden Sie unter Federal Information Processing Standard () 140-3. FIPS

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 arbeiten AWS IoT Greengrass oder sie anderweitig AWS -Services verwenden,API, AWS CLI oder. 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 einem externen Server eine URL zur Verfügung stellen, empfehlen wir dringend, dass Sie keine Anmeldeinformationen in den angebenURL, um Ihre Anfrage an diesen Server zu überprüfen.

Weitere Informationen zum Schutz vertraulicher Informationen in AWS IoT Greengrass finden Sie unterthe section called “Keine Protokollierung sensibler Informationen”.

Weitere Informationen zum Datenschutz finden Sie im Modell der AWS geteilten Verantwortung und im GDPR Blogbeitrag im AWS Sicherheitsblog.

Themen

- Datenverschlüsselung
- Integration von Hardware-Sicherheit

Datenverschlüsselung

AWS IoT Greengrassverwendet Verschlüsselung, um Daten während der Übertragung (über das Internet oder das lokale Netzwerk) und im Ruhezustand (imAWS Cloud) enthalten.

Geräte in einer AWS IoT Greengrass-Umgebung erfassen häufig Daten, die zur weiteren Verarbeitung an AWS-Services gesendet werden. Weitere Informationen zur Datenverschlüsselung für andere AWS-Services finden Sie in der Sicherheitsdokumentation des Services.

Themen

- Verschlüsselung während der Übertragung
- Verschlüsselung im Ruhezustand
- Schlüsselverwaltung für das Greengrass Core-Gerät

Verschlüsselung während der Übertragung

AWS IoT Greengrass besitzt drei Kommunikationsmodi, bei denen Daten übertragen werden:

- the section called “Daten in Übertragung über das Internet”aus. Kommunikation zwischen einem Greengrass-Kern undAWS IoT Greengrassüber das Internet ist verschlüsselt.
- the section called “Daten in Übertragung über das lokale Netzwerk”aus. Die Kommunikation zwischen einem Greengrass-Kern und Client-Geräten über ein lokales Netzwerk ist verschlüsselt.
- the section called “Daten auf dem Core-Gerät”aus. Die Kommunikation zwischen Komponenten auf dem Greengrass-Kerngerät ist nicht verschlüsselt.

Daten in Übertragung über das Internet

AWS IoT Greengrass verwendet Transport Layer Security (TLS) zur Verschlüsselung der gesamten Kommunikation über das Internet. Alle Daten, die an die gesendet werdenAWS Cloudwird über eine TLS-Verbindung mit dem MQTT- oder HTTPS-Protokoll gesendet, sodass sie standardmäßig sicher sind.AWS IoT Greengrassverwendet von von vonAWS IoT-Transportsicherheitsmodell. Weitere Informationen finden Sie unter Transportsicherheit im AWS IoT Core-Entwicklerhandbuch.

Daten in Übertragung über das lokale Netzwerk

AWS IoT Greengrassverwendet TLS, um die gesamte Kommunikation über das lokale Netzwerk zwischen dem Greengrass-Kern und 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 sind Sie für Folgendes verantwortlich:

- Ständige Aktualität des TLS-Stacks.
- Schutz privater Schlüssel.

Daten auf dem Core-Gerät

AWS IoT Greengrass verschlüsselt keine Daten, die lokal auf dem Greengrass-Kerngerät ausgetauscht werden, da die Daten das Gerät nicht verlassen. Dazu gehört die Kommunikation zwischen benutzerdefinierten Lambda-Funktionen, Konnektoren undAWS IoT Greengrass-Kern-SDK und Systemkomponenten, z. B. dem Stream-Manager.

Verschlüsselung im Ruhezustand

AWS IoT Greengrass speichert Ihre Daten:

- the section called “Daten im Ruhezustand imAWS Cloud”aus. Diese Daten sind verschlüsselt.

- the section called “Daten im Ruhezustand auf dem Greengrass-Kern”aus. Diese Daten sind nicht verschlüsselt (außer lokaler Kopien Ihrer Secrets).

Daten im Ruhezustand imAWS Cloud

AWS IoT Greengrassverschlüsselt Kundendaten, die imAWS Cloudaus. Diese Daten werden mit AWS KMS-Schlüsseln geschützt, die von AWS IoT Greengrass verwaltet werden.

Daten im Ruhezustand auf dem Greengrass-Kern

AWS IoT Greengrass nutzt Unix-Dateiberechtigungen und Volldatenträgerverschlüsselung (falls aktiviert), um Daten zu schützen, die sich auf dem Kern in Ruhe befinden. Sie sind für den Schutz des Dateisystems und des Geräts verantwortlich.

AWS IoT Greengrass verschlüsselt jedoch lokale Kopien Ihrer Secrets, die von AWS Secrets Manager abgerufen werden. Weitere Informationen finden Sie unter the section called “Secrets-Verschlüsselung”.

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 mit dem MQTT-Serverzertifikat zur Authentifizierung von TLS-Verbindungen zwischen Kern- und Clientgeräten verwendet. 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 “Secrets-Verschlüsselung”.

Ein Greengrass-Kern unterstützt den privaten Schlüsselspeicher mithilfe von Dateisystemberechtigungen, Hardwaresicherheitsmodulen 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 mit Gerätezertifikaten verwendet, um TLS-Verbindungen mit dem AWS IoT Greengrass-Service zu authentifizieren.

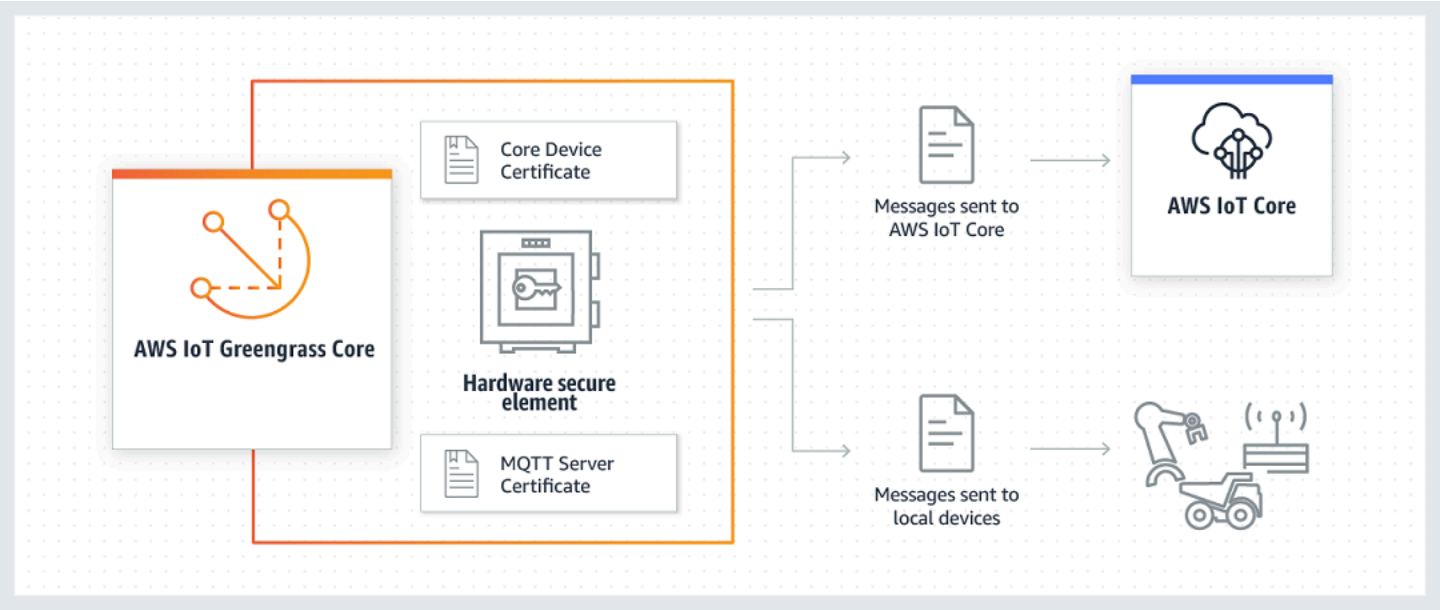
Integration von Hardware-Sicherheit

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 Gerätekatalog 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 sich um einen RSA Schlüssel mit einer Mindestlänge von 2048 Bit handeln. Weitere Informationen finden Sie unter the section called “Angeben des privaten Schlüssels für die Verschlüsselung von Secrets”.

Voraussetzungen

Bevor Sie die Hardwaresicherheit für einen Greengrass Core konfigurieren können, müssen Sie Folgendes beachten:

- Ein Hardware-Sicherheitsmodul (HSM), das Ihre private Zielschlüsselkonfiguration für die Komponenten IoT-Client, lokalen MQTT Server und Local Secrets Manager 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 Schlüsselgröße von RSA -2048 (oder größer) und das Signaturschema PKCS#1 v1.5.
  - Für EC-Schlüssel: Eine NIST P-256- oder P-384-Kurve. NIST

  > **Note**
  >
  > Suchen Sie im AWS Partner Gerätekatalog nach Geräten, die für diese Funktion qualifiziert sind.

- Eine PKCS #11 -Anbieterbibliothek, die zur Laufzeit geladen werden kann (mit libdl) und PKCS#11 -Funktionen bereitstellt.
- Das Hardwaremodul muss anhand der Steckplatzbezeichnung aufgelöst werden können, wie in der PKCS Spezifikation #11 definiert.
- Der private Schlüssel muss mithilfe der HSM vom Hersteller bereitgestellten Bereitstellungstools generiert und auf den 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 Open SSL libp11 PKCS #11 Wrapper-Bibliothek installiert sein. Weitere Informationen finden Sie unter the section called “OTAUpdates konfigurieren”.

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 das Kernelement 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 demHSM. Die Zertifikate müssen als Klartextdateien an einem Ort im Dateisystem geladen werden, der von Greengrass gelesen werden kann.

Hardwaresicherheitskonfiguration für einen AWS IoT Greengrass Kern

Die Hardware-Sicherheit wird in der Greengrass-Konfigurationsdatei konfiguriert. Dies ist die Datei `config.json`, die sich im Verzeichnis `/greengrass-root/config` befindet.

> **Note**
>
> Eine schrittweise Anleitung zum Einrichten einer HSM Konfiguration mithilfe einer reinen Softwareimplementierung finden Sie unterthe section called “Modul 7: Simulation der Hardware-Sicherheitsintegration”.
>
> > **Important**
> >
> > Die simulierte Konfiguration im Beispiel bietet keine Sicherheitsvorteile. Es soll Ihnen ermöglichen, mehr über die PKCS #11 -Spezifikation zu erfahren und erste Tests Ihrer Software durchzuführen, falls Sie HSM in future eine hardwarebasierte Software verwenden möchten.

Um die Hardwaresicherheit in zu konfigurieren AWS IoT Greengrass, bearbeiten Sie das `crypto` Objekt in. `config.json`

Bei Verwendung der Hardwaresicherheit wird das `crypto` Objekt verwendet, um Pfade zu Zertifikaten, privaten Schlüsseln und Ressourcen für die PKCS #11 -Anbieterbibliothek auf dem 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	Beschreibung	Hinweise
`caPath`	Der absolute Pfad zur AWS IoT Stammzertifizierungsstelle.	Muss eine Datei URI der Form sein:`file:///absolute/path/to/file` . **Note** Stellen Sie sicher, dass Ihre Endpunkte Ihrem Zertifikatstyp entsprechen.
`PKCS11`		
`OpenSSLEngine`	Optional. Der absolute Pfad zur Open SSL `.so` Engine-Datei, um die PKCS #11 -	Muss ein Pfad zu einer Datei im Dateisystem sein.

Feld	Beschreibung	Hinweise
	Unterstützung bei Open SSL zu aktivieren.	Diese Eigenschaft ist erforderlich, wenn Sie den Greengrass OTA Update Agent mit Hardwaresicherheit verwenden. Weitere Informationen finden Sie unter the section called “OTAUpdates konfigurieren”.
`P11Provider`	Der absolute Pfad zur PKCS libdl-loadable-Bibliothek der #11 -Implementierung.	Muss ein Pfad zu einer Datei im Dateisystem sein.
`slotLabel`	Das Slot-Label, das zur Identifizierung des Hardwaremoduls verwendet wird.	Muss den Spezifikationen des #11 -Labels entsprechen. PKCS
`slotUserPin`	Der BenutzerPIN, der verwendet wird, um den Greengrass-Kern für das Modul zu authentifizieren.	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	Beschreibung	Hinweise
`IoTCertificate.privateKeyPath`	Der Pfad zum privaten Core-Schlüssel.	Für den Dateisystemspeicher muss es sich um eine Datei URI der Form handeln:. `file:///absolute/path/to/file` Für die HSM Speicherung muss es sich um einen RFC7512 PKCS #11 -Pfad handeln, der die Objektbezeichnung angibt.
`IoTCertificate.certificatePath`	Der absolute Pfad zum Core-Gerätezertifikat.	Muss eine Datei URI der Form sein:`file:///absolute/path/to/file` .
`MQTTServerCertificate`	Optional. Der private Schlüssel, den der Kern in Kombination mit dem Zertifikat verwendet, um als MQTT Server oder Gateway zu fungieren.	

Feld	Beschreibung	Hinweise
`MQTTServerCertificate.privateKeyPath`	Der Pfad zum privaten Schlüssel des lokalen MQTT Servers.	Verwenden Sie diesen Wert, um Ihren eigenen privaten Schlüssel für den lokalen MQTT Server anzugeben. Für den Dateisystemspeicher muss es sich um eine Datei URI der Form handeln:`file:///absolute/path/to/file` . Für die HSM Speicherung muss es sich um einen RFC7512 PKCS #11 -Pfad handeln, der die Objektbezeichnung angibt. Wenn diese Eigenschaft weggelassen wird, wird der Schlüssel auf der Grundlage Ihrer Rotationseinstellungen AWS IoT Greengrass gedreht. 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	Beschreibung	Hinweise
`SecretsManager .privateKeyPath`	Der Pfad zum privaten Schlüssel des lokalen Secrets Managers.	Es wird nur ein RSA Schlüssel unterstützt. Für den Dateisystemspeicher muss es sich um eine Datei URI der Form handeln:`file:///`*`absolute/path/to/file`* . Für die HSM Speicherung muss es sich um einen RFC7512 PKCS #11 -Pfad handeln, der die Objektbezeichnung angibt. Der private Schlüssel muss mithilfe des Auffüllmechanismus PKCS#1 v1.5 generiert werden.

Feld	Beschreibung	Hinweise
`caPath`	Der absolute Pfad zur AWS IoT Root-CA.	Muss eine Datei URI der Form sein:`file:///`*`absolute/path/to/file`* . **Note** Stellen Sie sicher, dass Ihre Endpunkte Ihrem Zertifikatstyp entsprechen.
`PKCS11`		

Feld	Beschreibung	Hinweise
`OpenSSLEngine`	Optional. Der absolute Pfad zur Open SSL `.so` Engine-Datei, um die PKCS #11 - Unterstützung bei Open SSL 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 “OTAUpdates konfigurieren”.
`P11Provider`	Der absolute Pfad zur PKCS libdl-loadable-Bibliothek der #11 -Implementierung.	Muss ein Pfad zu einer Datei im Dateisystem sein.
`slotLabel`	Das Slot-Label, das zur Identifizierung des Hardwaremoduls verwendet wird.	Muss den Spezifikationen des #11 -Labels entsprechen. PKCS
`slotUserPin`	Der BenutzerPIN, der verwendet wird, um den Greengrass-Kern für das Modul zu authentifizieren.	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	Beschreibung	Hinweise
`IoTCertificate.privateKeyPath`	Der Pfad zum privaten Core-Schlüssel.	Für den Dateisystemspeicher muss es sich um eine Datei URI der Form handeln:. `file:///absolute/path/to/file` Für die HSM Speicherung muss es sich um einen RFC7512 PKCS #11 -Pfad handeln, der die Objektbezeichnung angibt.
`IoTCertificate.certificatePath`	Der absolute Pfad zum Core-Gerätezertifikat.	Muss eine Datei URI der Form sein:`file:///absolute/path/to/file` .
`MQTTServerCertificate`	Optional. Der private Schlüssel, den der Kern in Kombination mit dem Zertifikat verwendet, um als MQTT Server oder Gateway zu fungieren.	

Feld	Beschreibung	Hinweise
`MQTTServerCertificate.privateKeyPath`	Der Pfad zum privaten Schlüssel des lokalen MQTT Servers.	Verwenden Sie diesen Wert, um Ihren eigenen privaten Schlüssel für den lokalen MQTT Server anzugeben. Für den Dateisystemspeicher muss es sich um eine Datei URI der Form handeln:`file:///`*`absolute/path/to/file`* . Für die HSM Speicherung muss es sich um einen RFC7512 PKCS #11 -Pfad handeln, der die Objektbezeichnung angibt. Wenn diese Eigenschaft weggelassen wird, wird der Schlüssel auf der Grundlage Ihrer Rotationseinstellungen AWS IoT Greengrass gedreht. 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	Beschreibung	Hinweise
`SecretsManager .privateKeyPath`	Der Pfad zum privaten Schlüssel des lokalen Secrets Managers.	Es wird nur ein RSA Schlüssel unterstützt. Für den Dateisystemspeicher muss es sich um eine Datei URI der Form handeln:`file:///`*`absolute/path/to/file`* . Für die HSM Speicherung muss es sich um einen RFC7512 PKCS #11 -Pfad handeln, der die Objektbezeichnung angibt. Der private Schlüssel muss mithilfe des Auffüllmechanismus PKCS#1 v1.5 generiert werden.

Feld	Beschreibung	Hinweise
`caPath`	Der absolute Pfad zur AWS IoT Root-CA.	Muss eine Datei URI der Form sein:`file:///`*`absolute/path/to/file`* . **Note** Stellen Sie sicher, dass Ihre Endpunkte Ihrem Zertifikatstyp entsprechen.
`PKCS11`		

Feld	Beschreibung	Hinweise
`OpenSSLEngine`	Optional. Der absolute Pfad zur Open SSL `.so` Engine-Datei, um die PKCS #11 - Unterstützung bei Open SSL 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 “OTAUpdates konfigurieren”.
`P11Provider`	Der absolute Pfad zur PKCS libdl-loadable-Bibliothek der #11 -Implementierung.	Muss ein Pfad zu einer Datei im Dateisystem sein.
`slotLabel`	Das Slot-Label, das zur Identifizierung des Hardwaremoduls verwendet wird.	Muss den Spezifikationen des #11 -Labels entsprechen. PKCS
`slotUserPin`	Der BenutzerPIN, der verwendet wird, um den Greengrass-Kern für das Modul zu authentifizieren.	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	Beschreibung	Hinweise
`IoTCertificate.privateKeyPath`	Der Pfad zum privaten Core-Schlüssel.	Für den Dateisystemspeicher muss es sich um eine Datei URI der Form handeln:. `file:///absolute/path/to/file` Für die HSM Speicherung muss es sich um einen RFC7512 PKCS #11 -Pfad handeln, der die Objektbezeichnung angibt.
`IoTCertificate.certificatePath`	Der absolute Pfad zum Core-Gerätezertifikat.	Muss eine Datei URI der Form sein:`file:///absolute/path/to/file` .
`MQTTServerCertificate`	Optional. Der private Schlüssel, den der Kern in Kombination mit dem Zertifikat verwendet, um als MQTT Server oder Gateway zu fungieren.	

Feld	Beschreibung	Hinweise
`MQTTServerCertificate.privateKeyPath`	Der Pfad zum privaten Schlüssel des lokalen MQTT Servers.	Verwenden Sie diesen Wert, um Ihren eigenen privaten Schlüssel für den lokalen MQTT Server anzugeben. Für den Dateisystemspeicher muss es sich um eine Datei URI der Form handeln:`file:///`*`absolute/path/to/file`* . Für die HSM Speicherung muss es sich um einen RFC7512 PKCS #11 -Pfad handeln, der die Objektbezeichnung angibt. Wenn diese Eigenschaft weggelassen wird, wird der Schlüssel auf der Grundlage Ihrer Rotationseinstellungen AWS IoT Greengrass gedreht. 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	Beschreibung	Hinweise
`SecretsManager .privateKeyPath`	Der Pfad zum privaten Schlüssel des lokalen Secrets Managers.	Es wird nur ein RSA Schlüssel unterstützt. Für den Dateisystemspeicher muss es sich um eine Datei URI der Form handeln:`file:///`*`absolute/path/to/file`* . Für die HSM Speicherung muss es sich um einen RFC7512 PKCS #11 -Pfad handeln, der die Objektbezeichnung angibt. Der private Schlüssel muss mithilfe des Auffüllmechanismus PKCS#1 v1.5 generiert werden.

Bereitstellungspraktiken für die Hardwaresicherheit AWS IoT Greengrass

Im Folgenden werden die sicherheits- und leistungsbezogenen Bereitstellungspraktiken beschrieben.

Sicherheit

- Generieren Sie private Schlüssel direkt auf dem, HSM indem Sie den internen Hardware-Zufallszahlengenerator verwenden.

  > **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 () nicht. OAEP

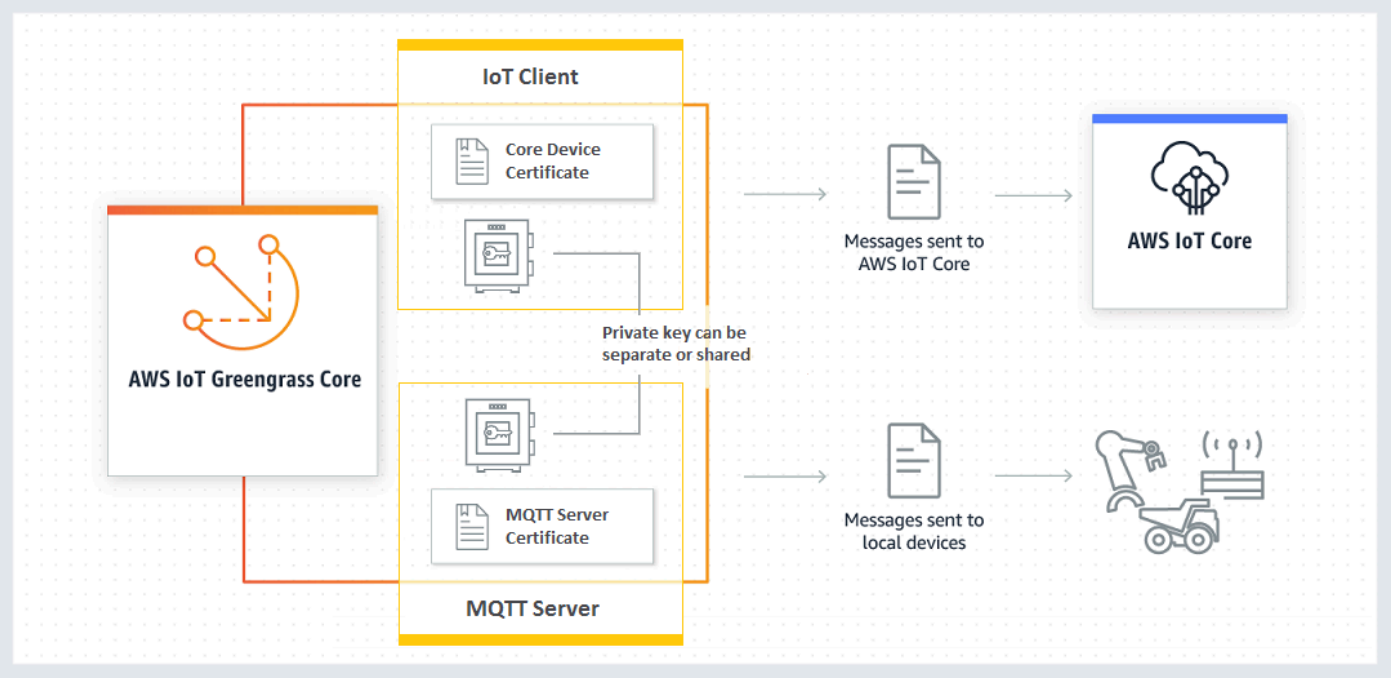
- Konfigurieren Sie private Schlüssel, um den Export zu verhindern.
- Verwenden Sie das vom Hardwareanbieter bereitgestellte Bereitstellungstool, um mithilfe des hardwaregeschützten privaten Schlüssels eine Zertifikatsignieranforderung (CSR) zu generieren, und verwenden Sie dann die AWS IoT Konsole, um ein Client-Zertifikat zu generieren.

> **Note**
>
> Die Praxis, Schlüssel zu rotieren, gilt nicht, wenn private Schlüssel auf einem generiert werden. HSM

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 denselben 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 speichern, nicht einHSM, sodass Sie private Schlüssel für die IoT-Client-, MQTT Server- und Secrets-Manager-Komponenten speichern können. Einige HSM Anbieter beschränken jedoch möglicherweise die Anzahl der Schlüssel, die Sie in einem Steckplatz speichern können.



Im Allgemeinen wird der IoT-Clientschlüssel nicht sehr häufig verwendet, da die AWS IoT Greengrass Core-Software langlebige Verbindungen zur Cloud unterhält. Der MQTT Serverschlüssel wird jedoch jedes Mal verwendet, wenn ein Greengrass-Gerät eine Verbindung zum Core herstellt. Diese Interaktionen wirken sich direkt auf die Leistung aus.

Wenn der MQTT Serverschlüssel auf dem gespeichert ist, hängt die GeschwindigkeitHSM, mit der Geräte eine Verbindung herstellen können, von der Anzahl der RSA Signaturvorgänge pro Sekunde ab, die sie ausführen HSM können. Wenn es beispielsweise 300 Millisekunden HSM dauert, eine Signatur von PKCS1 -v1.5 für einen privaten Schlüssel von RSASSA RSA -2048 auszuführen, können nur drei Geräte pro Sekunde eine Verbindung zum Greengrass-Core herstellen. Nachdem die Verbindungen hergestellt wurden, wird der nicht mehr verwendet und HSM es gelten die Standardkontingente für. AWS IoT Greengrass

Um Leistungsengpässe zu vermeiden, können Sie den privaten Schlüssel für den MQTT Server im Dateisystem statt auf dem speichern. HSM Bei dieser Konfiguration verhält sich der MQTT Server so, als ob die Hardwaresicherheit 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 Beispielkonfigurationen.

Konfiguration	IoT-Schlüssel	MQTTSchlüssel	Leistung
HSMGemeinsamer Schlüssel	HSM: Schlüssel A	HSM: Schlüssel A	Eingeschränkt durch das HSM oder CPU
HSMSeparate Schlüssel	HSM: Schlüssel A	HSM: Schlüssel B	Limitiert durch das HSM oder CPU
HSMnur für IoT	HSM: Schlüssel A	Dateisystem: Schlüssel B	Limitiert durch CPU
Veraltet	Dateisystem: Schlüssel A	Dateisystem: Schlüssel B	Limitiert durch 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 Standardschlüssel verwenden, der von generiert wurde). 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 von Greengrass-Geräten über das lokale Netzwerk eine Verbindung zum Greengrass Core herstellen, stellen Sie sicher, dass Sie eine der unterstützten Cipher Suites verwenden, um die Verbindung herzustellen. TLS

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-Wrapper-Bibliothek libp11 PKCS #11 installieren und die Greengrass-Konfigurationsdatei bearbeiten. Weitere Informationen zu Updates finden Sie unter. OTA 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 Sie die Open SSL Engine. Open SSL 1.0 oder 1.1 werden unterstützt.

   ```
   sudo apt-get install libengine-pkcs11-openssl
   ```

3. Suchen Sie den Pfad zur Open SSL 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 AWS IoT Greengrass verwendet. `coreThing.caPath` Diese Abwärtskompatibilität gilt für OTA Greengrass-Updates, die eine dateibasierte Konfiguration, die in angegeben ist, nicht überschreiben. `config.json`

PKCSHardware ohne #11 -Unterstützung

Die PKCS #11 -Bibliothek wird in der Regel vom Hardwareanbieter 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 Nutzungshandbuch für kryptografische Token-Schnittstellen Version 2.40. Herausgegeben von John Leiseboer und Robert Griffin. 16. November 2014. OASISAnmerkung des Ausschusses 02. 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.
- RFC7512
- PKCS#1: RSA Verschlüsselung 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 oder mit AWS IoT Core und AWS IoT Greengrass zu verbinden.

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 den Satz von Operationen, die für AWS IoT-Geräte zulässig sind. Insbesondere erlauben und verweigern sie den Zugriff auf AWS IoT Core- und AWS IoT Greengrass-Operationen auf Datenebene, z. B. 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 angefügten AWS IoT-Richtlinie. Geräte fallen in zwei Kategorien:

- Greengrass-Kerne. Greengrass-Kerngeräte verwenden Zertifikate und AWS IoT-Richtlinien, um eine sichere Verbindung zu AWS IoT Core herzustellen. Die Zertifikate und Richtlinien ermöglichen es auchAWS IoT Greengrass, Konfigurationsinformationen, Lambda-Funktionen, Connectors und verwaltete Abonnements auf -Core-Geräten bereitzustellen.
- Client-Geräte . Client-Geräte (auch als verbundene Geräte, Greengrass-Geräte oder Geräte bezeichnet) sind Geräte, die über MQTT eine Verbindung zu einem Greengrass-Kern herstellen. Sie verwenden Zertifikate und Richtlinien, um eine Verbindung zu AWS IoT Core und dem AWS IoT Greengrass Service herzustellen. Auf diese Weise können Client-Geräte den AWS IoT Greengrass Discovery Service verwenden, um ein Core-Gerät zu finden und eine Verbindung zu ihm herzustellen. Ein Client-Gerät verwendet dasselbe Zertifikat, um eine Verbindung mit dem AWS IoT Core Geräte-Gateway und dem Core-Gerät herzustellen. Client-Geräte verwenden auch Erkennungsinformationen für die gegenseitige Authentifizierung mit dem Core-Gerät. Weitere Informationen finden 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 und AWS IoT Core 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 verbindet sich mit AWS IoT Core und AWS IoT Greengrass über das Internet.
- Client-Geräte, die eine Verbindung zu herstellenAWS IoT Greengrass, um Kernerkennungsinformationen über das Internet abzurufen.

- Die MQTT-Serverkomponente auf dem Greengrass-Kern, die über das lokale Netzwerk eine Verbindung zu Client-Geräten in der Gruppe herstellt.

Das AWS IoT Greengrass Core-Gerät speichert Zertifikate an zwei Speicherorten:

- 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 Kern eine Verbindung mit den Services AWS IoT Core und AWS IoT Greengrass 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` steht für den Pfad, unter 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 Stammzertifizierungsstellenzertifikat herunter, das für die Authentifizierung mit - AWS IoT Core und -AWS IoT GreengrassServices 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 Stammzertifizierungsstellen-Zertifikattyp muss mit Ihrem Endpunkt übereinstimmen. Verwenden Sie ein ATS-Stammzertifizierungsstellenzertifikat mit einem ATS-Endpunkt (bevorzugt) oder ein VeriSign Stammzertifizierungsstellenzertifikat mit einem Legacy-Endpunkt. Nur einige Amazon Web Services-Regionen unterstützen Legacy-Endpunkte.

Weitere Informationen finden Sie unter the section called “Service-Endpunkte müssen mit dem Zertifikattyp übereinstimmen”.

Client-Geräte laden auch das Greengrass-Gruppenzertifizierungsstellenzertifikat 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 Zertifikatrotation stattfinden kann, muss Ihr Greengrass-Kerngerät online sein und regelmäßig direkt auf den AWS IoT Greengrass Service zugreifen können. Wenn das Zertifikat abläuft, versucht das Core-Gerät, eine Verbindung zum AWS IoT Greengrass Service 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. An diesem Punkt werden alle Client-Geräte, die mit dem Core verbunden sind, getrennt. Wenn das Core-Gerät zum Zeitpunkt des Ablaufs offline ist, erhält es das Ersatzzertifikat nicht. Alle neuen Versuche, eine Verbindung mit dem Core-Gerät herzustellen, werden abgelehnt. Bestehende Verbindungen bleiben davon unberührt. Client-Geräte können erst dann eine Verbindung zum Core-Gerät herstellen, wenn die Verbindung zum AWS IoT Greengrass Service wiederhergestellt 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 den Ablauf des Zertifikats auf einen Wert über 30 Tage festlegen möchten, wenden Sie sich an AWS Support.

In der AWS IoT Konsole können Sie das Zertifikat auf der Seite Einstellungen 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 Operationen auf Datenebene

Verwenden Sie AWS IoT-Richtlinien, um den Zugriff auf die Datenebene AWS IoT Core und AWS IoT Greengrass zu autorisieren. Die Datenebene AWS IoT Core besteht aus Operationen für Geräte, Benutzer und Anwendungen, wie z. B. die Verbindung zu 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 IoTRichtlinie 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 zugelassen oder verweigert wird.

AWS IoT -Richtlinien unterstützen * als Platzhalterzeichen und behandeln MQTT-Platzhalterzeichen (+ und #) als Literalzeichenfolgen. Weitere Informationen zum * Platzhalter finden Sie unter Verwenden von Platzhaltern in Ressourcen-ARNs im AWS Identity and Access Management - Benutzerhandbuch.

Weitere Informationen finden Sie unter AWS IoT-Richtlinien und AWS IoT-Richtlinienaktionen im AWS IoT Core-Entwicklerhandbuch.

> **Note**
>
> AWS IoT Core Mit können Sie Objektgruppen AWS IoT Richtlinien zuordnen, um Berechtigungen für Gerätegruppen zu definieren. Richtlinien für Objektgruppen erlauben keinen Zugriff auf AWS IoT Greengrass Datenebenenoperationen. Um einem Objekt 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 des Objekts hinzufügen.

AWS IoT Greengrass-Richtlinienaktionen

Greengrass-Kernaktionen

AWS IoT Greengrass definiert die folgenden Richtlinienaktionen, die Greengrass-Kerngeräte in AWS IoT-Richtlinien verwenden können:

`greengrass:AssumeRoleForGroup`

Berechtigung für ein Greengrass-Core-Gerät zum Abrufen von Anmeldeinformationen mithilfe der Lambda-Funktion des Token Exchange Service (TES)-Systems. 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-Kerngerät versucht, seine Konnektivitätsinformationen 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`

Berechtigung für ein Greengrass-Core-Gerät zum Abrufen von Bereitstellungsartefakten wie Gruppeninformationen oder Lambda-Funktionen.

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`

Berechtigung für ein Client-Gerät, die Discovery-API zum Abrufen der Kernkonnektivitätsinformationen und der Gruppenzertifizierungsstelle 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 Core-Gerät

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, Connectors 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 enthält einen Abschnitt, der AWS IoT Core zum Abrufen, Aktualisieren und Löschen des Kern-Geräteschattens berechtigt. Um die Schattensynchronisierung für Client-Geräte in der Greengrass-Gruppe zu ermöglichen, geben Sie die Amazon-Ressourcennamen (ARNs) in der `Resource` Liste an (z. B. `arn:aws:iot:region:account-id:thing/device-name`).
- Die Verwendung von Objektrichtlinien-Variablen (`iot:Connection.Thing.*`) in der AWS IoT-Richtlinie für ein Kerngerät wird nicht unterstützt. Der Core verwendet dasselbe Gerätezertifikat, um mehrere Verbindungen mit AWS IoT Core herzustellen, aber die Client-ID in einer Verbindung stimmt möglicherweise nicht exakt mit dem Namen des Core-Objekts überein.
- Für die `greengrass:UpdateCoreDeploymentStatus`-Berechtigung ist das letzte Segment im `Resource`-ARN der URL-codierte ARN des Core-Geräts.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:client/core-name-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topic/$aws/things/core-name-*"
            ]
        },
        {
            "Effect": "Allow",
```

```
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topicfilter/$aws/things/core-name-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:GetThingShadow",
                "iot:UpdateThingShadow",
                "iot:DeleteThingShadow"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:thing/core-name-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:AssumeRoleForGroup",
                "greengrass:CreateCertificate"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:GetDeployment"
            ],
            "Resource": [
                "arn:aws:greengrass:region:account-id:/greengrass/groups/group-id/
deployments/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:GetDeploymentArtifacts"
            ],
            "Resource": [
```

```
                "arn:aws:greengrass:region:account-id:/greengrass/groups/group-id/
deployments/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:UpdateCoreDeploymentStatus"
            ],
            "Resource": [
                "arn:aws:greengrass:region:account-id:/greengrass/groups/group-id/
deployments/*/cores/arn%3Aaws%3Aiot%3Aregion%3Aaccount-id%3Athing%2Fcore-name"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:GetConnectivityInfo",
                "greengrass:UpdateConnectivityInfo"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:thing/core-name-*"
            ]
        }
    ]
}
```

Note

AWS IoT -Richtlinien für Client-Geräte erfordern in der Regel ähnliche Berechtigungen für `iot:Connect`-, `-iot:Receive`, - und `-iot:SubscribeAktioniot:Publish`en. Damit ein Client-Gerät automatisch Konnektivitätsinformationen 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 enthalten. Geben Sie im `Resource` Abschnitt den ARN des Client-Geräts an, nicht den ARN des Greengrass-Core-Geräts. Beispielsweise:

```
{
    "Effect": "Allow",
    "Action": [
        "greengrass:Discover"
    ],
```

```
    "Resource": [
        "arn:aws:iot:region:account-id:thing/device-name"
    ]
}
```

Die AWS IoT Richtlinie für Client-Geräte erfordert in der Regel keine Berechtigungen für `iot:GetThingShadow`-`iot:UpdateThingShadow`, - oder -`iot:DeleteThingShadow`Aktionen, da der Greengrass-Kern Schattensynchronisierungsvorgänge für Client-Geräte übernimmt. Stellen Sie in diesem Fall sicher, dass der `Resource` Abschnitt für Schattenaktionen in der AWS IoT Richtlinie des Kerns die ARNs der Client-Geräte enthält.

In der AWS IoT Konsole können Sie die Richtlinie anzeigen und bearbeiten, die dem Zertifikat Ihres Kerns zugeordnet ist.

1. Erweitern Sie im Navigationsbereich unter Verwalten die Option Alle Geräte und wählen Sie dann Objekte aus.
2. Wählen Sie Ihren Kern aus.
3. Wählen Sie auf der Konfigurationsseite Ihres Kerns die Registerkarte 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 aus.
6. Überprüfen Sie die Richtlinie und fügen Sie Berechtigungen nach Bedarf hinzu, entfernen oder bearbeiten Sie sie.
7. Um eine neue Richtlinienversion als aktive Version festzulegen, wählen Sie unter Status der Richtlinienversion die Option Bearbeitene 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 AWS Ressourcen sicher zu kontrollieren. IAMAdministratoren kontrollieren, wer authentifiziert (angemeldet) und autorisiert werden kann (über Berechtigungen verfügt), um AWS IoT Greengrass Ressourcen zu verwenden. IAMist eine AWS -Service , die Sie ohne zusätzliche Kosten verwenden können.

> **Note**
>
> In diesem Thema werden IAM Konzepte und Funktionen beschrieben. Informationen zu den IAM Funktionen, die von unterstützt werden AWS IoT Greengrass, finden Sie unterthe section called “Wie AWS IoT Greengrass funktioniert mit IAM”.

Zielgruppe

Wie Sie AWS Identity and Access Management (IAM) verwenden, hängt von der jeweiligen Arbeit ab AWS IoT Greengrass.

Dienstbenutzer — Wenn Sie den AWS IoT Greengrass Dienst für Ihre Arbeit verwenden, stellt Ihnen Ihr Administrator die erforderlichen Anmeldeinformationen und Berechtigungen zur Verfügung. Wenn Sie für Ihre Arbeit mehr AWS IoT Greengrass Funktionen verwenden, benötigen Sie möglicherweise zusätzliche Berechtigungen. Wenn Sie die Fuktionsweise der Zugriffskontrolle nachvollziehen, wissen Sie bereits, welche Berechtigungen Sie von Ihrem Administrator anzufordern müssen. Unter Problembehandlung bei Identitäts- und Zugriffsproblemen für AWS IoT Greengrass finden Sie nützliche Informationen für den Fall, dass Sie keinen Zugriff auf eine Feature in AWS IoT Greengrass haben.

Serviceadministrator — Wenn Sie in Ihrem Unternehmen für die AWS IoT Greengrass Ressourcen verantwortlich sind, haben Sie wahrscheinlich vollen Zugriff auf AWS IoT Greengrass. Es ist Ihre Aufgabe, zu bestimmen, auf welche AWS IoT Greengrass Funktionen und Ressourcen Ihre Servicebenutzer zugreifen sollen. Anschließend müssen Sie Anfragen an Ihren IAM Administrator senden, um die Berechtigungen Ihrer Servicebenutzer zu ändern. Lesen Sie die Informationen auf dieser Seite, um die grundlegenden Konzepte von zu verstehenIAM. Weitere Informationen darüber, wie Ihr Unternehmen IAM mit verwenden kann AWS IoT Greengrass, finden Sie unterWie AWS IoT Greengrass funktioniert mit IAM.

IAMAdministrator — Wenn Sie ein IAM Administrator sind, möchten Sie vielleicht mehr darüber erfahren, wie Sie Richtlinien schreiben können, um den Zugriff darauf zu verwalten AWS IoT Greengrass. Beispiele für AWS IoT Greengrass identitätsbasierte Richtlinien, die Sie in verwenden könnenIAM, finden Sie unter. 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 als IAM Benutzer authentifiziert (angemeldet AWS) sein oder eine IAM Rolle übernehmen. Root-Benutzer des AWS-Kontos

Sie können sich AWS als föderierte Identität anmelden, indem Sie Anmeldeinformationen verwenden, die über eine Identitätsquelle bereitgestellt wurden. AWS IAM Identity Center (IAMIdentity Center-) Nutzer, die Single-Sign-On-Authentifizierung Ihres Unternehmens und Ihre Google- oder Facebook-Anmeldeinformationen sind Beispiele für föderierte Identitäten. Wenn Sie sich als föderierte Identität anmelden, hat Ihr Administrator zuvor einen Identitätsverbund mithilfe von Rollen eingerichtet. IAM Wenn Sie AWS mithilfe eines Verbunds darauf zugreifen, übernehmen Sie indirekt eine Rolle.

Je nachdem, welcher Benutzertyp Sie sind, können Sie sich beim AWS Management Console oder beim AWS Zugangsportal anmelden. Weitere Informationen zur Anmeldung finden Sie AWS unter So melden Sie sich bei Ihrem an AWS-Konto im AWS-Anmeldung Benutzerhandbuch.

Wenn Sie AWS programmgesteuert darauf zugreifen, AWS stellt es ein Software Development Kit (SDK) und eine Befehlszeilenschnittstelle (CLI) bereit, mit der Sie Ihre Anfragen mithilfe Ihrer Anmeldeinformationen kryptografisch signieren können. Wenn Sie keine AWS Tools verwenden, müssen Sie Anfragen selbst signieren. Weitere Informationen zur Verwendung der empfohlenen Methode, um Anfragen selbst zu signieren, finden Sie im IAMBenutzerhandbuch unter AWS API Anfragen signieren.

Unabhängig von der verwendeten Authentifizierungsmethode müssen Sie möglicherweise zusätzliche Sicherheitsinformationen angeben. AWS Empfiehlt beispielsweise, die Multi-Faktor-Authentifizierung (MFA) zu verwenden, um die Sicherheit Ihres Kontos zu erhöhen. Weitere Informationen finden Sie unter Multi-Faktor-Authentifizierung im AWS IAM Identity Center Benutzerhandbuch und Verwenden der Multi-Faktor-Authentifizierung (MFA) AWS im IAM Benutzerhandbuch.

AWS-Konto Root-Benutzer

Wenn Sie einen erstellen AWS-Konto, beginnen Sie mit einer Anmeldeidentität, die vollständigen Zugriff auf alle AWS -Services Ressourcen im Konto hat. Diese Identität wird als AWS-Konto Root-

Benutzer bezeichnet. Sie können darauf zugreifen, indem Sie sich mit der E-Mail-Adresse und dem Passwort anmelden, mit denen Sie das Konto erstellt haben. Wir raten ausdrücklich davon ab, den Root-Benutzer für Alltagsaufgaben zu verwenden. Schützen Sie Ihre Root-Benutzer-Anmeldeinformationen und verwenden Sie diese, um die Aufgaben auszuführen, die nur der Root-Benutzer ausführen kann. Eine vollständige Liste der Aufgaben, für die Sie sich als Root-Benutzer anmelden müssen, finden Sie im Benutzerhandbuch unter Aufgaben, für die Root-Benutzeranmeldedaten erforderlich sind. IAM

IAM-Benutzer und -Gruppen

Ein IAMBenutzer ist eine Identität innerhalb Ihres Unternehmens AWS-Konto , die über spezifische Berechtigungen für eine einzelne Person oder Anwendung verfügt. Wir empfehlen, sich nach Möglichkeit auf temporäre Anmeldeinformationen zu verlassen, anstatt IAM Benutzer mit langfristigen Anmeldeinformationen wie Passwörtern und Zugriffsschlüsseln zu erstellen. Wenn Sie jedoch spezielle Anwendungsfälle haben, für die langfristige Anmeldeinformationen von IAM Benutzern erforderlich sind, empfehlen wir, die Zugriffsschlüssel abwechselnd zu verwenden. Weitere Informationen finden Sie im Benutzerhandbuch unter Regelmäßiges Rotieren von Zugriffsschlüsseln für Anwendungsfälle, für die IAM langfristige Anmeldeinformationen erforderlich sind.

Eine IAMGruppe ist eine Identität, die eine Sammlung von IAM Benutzern spezifiziert. Sie können sich nicht als Gruppe anmelden. Mithilfe von Gruppen können Sie Berechtigungen für mehrere Benutzer gleichzeitig angeben. Gruppen vereinfachen die Verwaltung von Berechtigungen, wenn es zahlreiche Benutzer gibt. Sie könnten beispielsweise eine Gruppe benennen IAMAdminsund dieser Gruppe Berechtigungen zur Verwaltung von IAM Ressourcen erteilen.

Benutzer unterscheiden sich von Rollen. Ein Benutzer ist einer einzigen Person oder Anwendung eindeutig zugeordnet. Eine Rolle kann von allen Personen angenommen werden, die sie benötigen. Benutzer besitzen dauerhafte Anmeldeinformationen. Rollen stellen temporäre Anmeldeinformationen bereit. Weitere Informationen finden Sie unter Wann sollte ein IAM Benutzer (statt einer Rolle) erstellt werden? im IAMBenutzerhandbuch.

IAMRollen

Eine IAMRolle ist eine Identität innerhalb von Ihnen AWS-Konto , für die bestimmte Berechtigungen gelten. Sie ähnelt einem IAM Benutzer, ist jedoch keiner bestimmten Person zugeordnet. Sie können vorübergehend eine IAM Rolle in der übernehmen, AWS Management Console indem Sie die Rollen wechseln. Sie können eine Rolle übernehmen, indem Sie eine AWS CLI AWS API OR-Operation aufrufen oder eine benutzerdefinierte Operation verwendenURL. Weitere Informationen zu Methoden zur Verwendung von Rollen finden Sie unter Verwenden von IAM Rollen im IAMBenutzerhandbuch.

IAMRollen mit temporären Anmeldeinformationen sind in den folgenden Situationen nützlich:

- Verbundbenutzerzugriff – Um einer Verbundidentität Berechtigungen zuzuweisen, erstellen Sie eine Rolle und definieren Berechtigungen für die Rolle. Wird eine Verbundidentität authentifiziert, so wird die Identität der Rolle zugeordnet und erhält die von der Rolle definierten Berechtigungen. Informationen zu Rollen für den Verbund finden Sie im IAMBenutzerhandbuch unter Erstellen einer Rolle für einen externen Identitätsanbieter. Wenn Sie IAM Identity Center verwenden, konfigurieren Sie einen Berechtigungssatz. Um zu kontrollieren, worauf Ihre Identitäten nach der Authentifizierung zugreifen können, korreliert IAM Identity Center den Berechtigungssatz mit einer Rolle in. IAM Informationen zu Berechtigungssätzen finden Sie unter Berechtigungssätze im AWS IAM Identity Center -Benutzerhandbuch.
- Temporäre IAM Benutzerberechtigungen — Ein IAM Benutzer oder eine Rolle kann eine IAM Rolle übernehmen, um vorübergehend verschiedene Berechtigungen für eine bestimmte Aufgabe zu übernehmen.
- Kontoübergreifender Zugriff — Sie können eine IAM Rolle verwenden, um einer Person (einem vertrauenswürdigen Principal) in einem anderen Konto den Zugriff auf Ressourcen in Ihrem Konto zu ermöglichen. Rollen stellen die primäre Möglichkeit dar, um kontoübergreifendem Zugriff zu gewähren. Bei einigen können Sie AWS -Services jedoch eine Richtlinie direkt an eine Ressource anhängen (anstatt eine Rolle als Proxy zu verwenden). Informationen zum Unterschied zwischen Rollen und ressourcenbasierten Richtlinien für den kontenübergreifenden Zugriff finden Sie IAMim Benutzerhandbuch unter Kontoübergreifender Ressourcenzugriff. IAM
- Serviceübergreifender Zugriff — Einige AWS -Services verwenden Funktionen in anderen. AWS -Services Wenn Sie beispielsweise einen Service aufrufen, ist es üblich, dass dieser Service Anwendungen in Amazon ausführt EC2 oder Objekte in Amazon S3 speichert. Ein Dienst kann dies mit den Berechtigungen des aufrufenden Prinzipals mit einer Servicerolle oder mit einer serviceverknüpften Rolle tun.
  - Zugriffssitzungen weiterleiten (FAS) — Wenn Sie einen IAM Benutzer oder eine Rolle verwenden, um Aktionen auszuführen AWS, gelten Sie als Principal. Bei einigen Services könnte es Aktionen geben, die dann eine andere Aktion in einem anderen Service initiieren. FASverwendet die Berechtigungen des Prinzipals, der an aufruft AWS -Service, kombiniert mit der Anforderung, Anfragen AWS -Service an nachgelagerte Dienste zu stellen. FASAnfragen werden nur gestellt, wenn ein Dienst eine Anfrage erhält, für deren Abschluss Interaktionen mit anderen AWS -Services oder Ressourcen erforderlich sind. In diesem Fall müssen Sie über Berechtigungen zum Ausführen beider Aktionen verfügen. Einzelheiten zu den Richtlinien beim Stellen von FAS Anfragen finden Sie unter Zugriffssitzungen weiterleiten.

- Servicerolle — Eine Servicerolle ist eine IAMRolle, die ein Dienst übernimmt, um Aktionen in Ihrem Namen auszuführen. Ein IAM Administrator kann eine Servicerolle von innen heraus erstellen, ändern und löschenIAM. Weitere Informationen finden Sie im IAMBenutzerhandbuch unter Erstellen einer Rolle zum Delegieren von Berechtigungen AWS -Service an eine.
- Dienstbezogene Rolle — Eine dienstverknüpfte Rolle ist eine Art von Servicerolle, die mit einer verknüpft ist. AWS -Service Der Service kann die Rolle übernehmen, um eine Aktion in Ihrem Namen auszuführen. Servicebezogene Rollen erscheinen in Ihrem Dienst AWS-Konto und gehören dem Dienst. Ein IAM Administrator kann die Berechtigungen für dienstbezogene Rollen anzeigen, aber nicht bearbeiten.

- Auf Amazon ausgeführte Anwendungen EC2 — Sie können eine IAM Rolle verwenden, um temporäre Anmeldeinformationen für Anwendungen zu verwalten, die auf einer EC2 Instance ausgeführt werden und AWS API Anfragen stellen AWS CLI . Dies ist dem Speichern von Zugriffsschlüsseln innerhalb der EC2 Instance vorzuziehen. Um einer EC2 Instanz eine AWS Rolle zuzuweisen und sie allen ihren Anwendungen zur Verfügung zu stellen, erstellen Sie ein Instanzprofil, das an die Instanz angehängt ist. Ein Instanzprofil enthält die Rolle und ermöglicht Programmen, die auf der EC2 Instanz ausgeführt werden, temporäre Anmeldeinformationen abzurufen. Weitere Informationen finden Sie im IAMBenutzerhandbuch unter Verwenden einer IAM Rolle zur Erteilung von Berechtigungen für Anwendungen, die auf EC2 Amazon-Instances ausgeführt werden.

Informationen darüber, ob Sie IAM Rollen oder IAM Benutzer verwenden sollten, finden Sie im Benutzerhandbuch unter Wann sollte eine IAM Rolle (anstelle eines IAM Benutzers) erstellt werden.

Verwalten des Zugriffs mit Richtlinien

Sie steuern den Zugriff, AWS indem Sie Richtlinien erstellen und diese an AWS Identitäten oder Ressourcen anhängen. Eine Richtlinie ist ein Objekt, AWS das, wenn es einer Identität oder Ressource zugeordnet ist, deren Berechtigungen definiert. AWS wertet diese Richtlinien aus, wenn ein Prinzipal (Benutzer, Root-Benutzer oder Rollensitzung) eine Anfrage stellt. Berechtigungen in den Richtlinien bestimmen, ob die Anforderung zugelassen oder abgelehnt wird. Die meisten Richtlinien werden in AWS Form von JSON Dokumenten gespeichert. Weitere Informationen zur Struktur und zum Inhalt von JSON Richtliniendokumenten finden Sie im IAMBenutzerhandbuch unter Überblick über JSON Richtlinien.

Administratoren können mithilfe von AWS JSON Richtlinien festlegen, wer Zugriff auf was hat. Das bedeutet, welcher Prinzipal kann Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen.

Standardmäßig haben Benutzer, Gruppen und Rollen keine Berechtigungen. Um Benutzern die Erlaubnis zu erteilen, Aktionen mit den Ressourcen durchzuführen, die sie benötigen, kann ein IAM Administrator IAM Richtlinien erstellen. Der Administrator kann dann die IAM Richtlinien zu Rollen hinzufügen, und Benutzer können die Rollen übernehmen.

IAMRichtlinien definieren Berechtigungen für eine Aktion, unabhängig von der Methode, mit der Sie den Vorgang ausführen. Angenommen, es gibt eine Richtlinie, die Berechtigungen für die `iam:GetRole`-Aktion erteilt. Ein Benutzer mit dieser Richtlinie kann Rolleninformationen aus dem AWS Management Console AWS CLI, dem oder dem abrufen AWS API.

Identitätsbasierte Richtlinien

Identitätsbasierte Richtlinien sind Dokumente mit JSON Berechtigungsrichtlinien, die Sie an eine Identität anhängen können, z. B. an einen IAM Benutzer, eine Benutzergruppe oder eine Rolle. Diese Richtlinien steuern, welche Aktionen die Benutzer und Rollen für welche Ressourcen und unter welchen Bedingungen ausführen können. Informationen zum Erstellen einer identitätsbasierten Richtlinie finden Sie unter IAMRichtlinien erstellen im Benutzerhandbuch. IAM

Identitätsbasierte Richtlinien können weiter als Inline-Richtlinien oder verwaltete Richtlinien kategorisiert werden. Inline-Richtlinien sind direkt in einen einzelnen Benutzer, eine einzelne Gruppe oder eine einzelne Rolle eingebettet. Verwaltete Richtlinien sind eigenständige Richtlinien, die Sie mehreren Benutzern, Gruppen und Rollen in Ihrem System zuordnen können. AWS-Konto Zu den verwalteten Richtlinien gehören AWS verwaltete Richtlinien und vom Kunden verwaltete Richtlinien. Informationen dazu, wie Sie zwischen einer verwalteten Richtlinie oder einer Inline-Richtlinie wählen können, finden Sie im IAMBenutzerhandbuch unter Auswahl zwischen verwalteten Richtlinien und Inline-Richtlinien.

Ressourcenbasierte Richtlinien

Ressourcenbasierte Richtlinien sind JSON Richtliniendokumente, die Sie an eine Ressource anhängen. Beispiele für ressourcenbasierte Richtlinien sind IAM Rollenvertrauensrichtlinien und Amazon S3 S3-Bucket-Richtlinien. In Services, die ressourcenbasierte Richtlinien unterstützen, können Service-Administratoren sie verwenden, um den Zugriff auf eine bestimmte Ressource zu steuern. Für die Ressource, an welche die Richtlinie angehängt ist, legt die Richtlinie fest, welche Aktionen ein bestimmter Prinzipal unter welchen Bedingungen für diese Ressource ausführen kann. Sie müssen in einer ressourcenbasierten Richtlinie einen Prinzipal angeben. Zu den Prinzipalen können Konten, Benutzer, Rollen, Verbundbenutzer oder gehören. AWS -Services

Ressourcenbasierte Richtlinien sind Richtlinien innerhalb dieses Diensts. Sie können AWS verwaltete Richtlinien nicht IAM in einer ressourcenbasierten Richtlinie verwenden.

Zugriffskontrolllisten () ACLs

Zugriffskontrolllisten (ACLs) steuern, welche Principals (Kontomitglieder, Benutzer oder Rollen) über Zugriffsberechtigungen für eine Ressource verfügen. ACLsähneln ressourcenbasierten Richtlinien, verwenden jedoch nicht das JSON Richtliniendokumentformat.

Amazon S3 und AWS WAF Amazon VPC sind Beispiele für Dienste, die Unterstützung bietenACLs. Weitere Informationen finden Sie unter Übersicht über ACLs die Zugriffskontrollliste (ACL) im Amazon Simple Storage Service Developer Guide.

Weitere Richtlinientypen

AWS unterstützt zusätzliche, weniger verbreitete Richtlinientypen. Diese Richtlinientypen können die maximalen Berechtigungen festlegen, die Ihnen von den häufiger verwendeten Richtlinientypen erteilt werden können.

- Berechtigungsgrenzen — Eine Berechtigungsgrenze ist eine erweiterte Funktion, mit der Sie die maximalen Berechtigungen festlegen, die eine identitätsbasierte Richtlinie einer IAM Entität (IAMBenutzer oder Rolle) gewähren kann. Sie können eine Berechtigungsgrenze für eine Entität festlegen. Die daraus resultierenden Berechtigungen sind der Schnittpunkt der identitätsbasierten Richtlinien einer Entität und ihrer Berechtigungsgrenzen. Ressourcenbasierte Richtlinien, die den Benutzer oder die Rolle im Feld `Principal` angeben, werden nicht durch Berechtigungsgrenzen eingeschränkt. Eine explizite Zugriffsverweigerung in einer dieser Richtlinien setzt eine Zugriffserlaubnis außer Kraft. Weitere Informationen zu Berechtigungsgrenzen finden Sie im IAMBenutzerhandbuch unter Berechtigungsgrenzen für IAM Entitäten.
- Dienststeuerungsrichtlinien (SCPs) — SCPs sind JSON Richtlinien, die die maximalen Berechtigungen für eine Organisation oder Organisationseinheit (OU) in festlegen AWS Organizations. AWS Organizations ist ein Dienst zur Gruppierung und zentralen Verwaltung mehrerer Geräte AWS-Konten , die Ihrem Unternehmen gehören. Wenn Sie alle Funktionen in einer Organisation aktivieren, können Sie Richtlinien zur Servicesteuerung (SCPs) auf einige oder alle Ihre Konten anwenden. Das SCP schränkt die Berechtigungen für Entitäten in Mitgliedskonten ein, einschließlich der einzelnen Root-Benutzer des AWS-Kontos. Weitere Informationen zu Organizations und SCPs finden Sie unter Richtlinien zur Servicesteuerung im AWS Organizations Benutzerhandbuch.

- Sitzungsrichtlinien – Sitzungsrichtlinien sind erweiterte Richtlinien, die Sie als Parameter übergeben, wenn Sie eine temporäre Sitzung für eine Rolle oder einen verbundenen Benutzer programmgesteuert erstellen. Die resultierenden Sitzungsberechtigungen sind eine Schnittmenge der auf der Identität des Benutzers oder der Rolle basierenden Richtlinien und der Sitzungsrichtlinien. Berechtigungen können auch aus einer ressourcenbasierten Richtlinie stammen. Eine explizite Zugriffsverweigerung in einer dieser Richtlinien setzt eine Zugriffserlaubnis außer Kraft. Weitere Informationen finden Sie im IAMBenutzerhandbuch unter Sitzungsrichtlinien.

Mehrere Richtlinientypen

Wenn mehrere auf eine Anforderung mehrere Richtlinientypen angewendet werden können, sind die entsprechenden Berechtigungen komplizierter. Informationen darüber, wie AWS bestimmt wird, ob eine Anfrage zulässig ist, wenn mehrere Richtlinientypen betroffen sind, finden Sie im IAMBenutzerhandbuch unter Bewertungslogik für Richtlinien.

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 den Zugriff auf verwalten AWS IoT Greengrass, sollten Sie sich mit den IAM Funktionen vertraut machen, die Sie mit verwenden können AWS IoT Greengrass.

IAMMerkmal	Von Greengrass unterstützt?
Identitätsbasierte Richtlinien mit Berechtigungen auf Ressourcenebene	Ja
Ressourcenbasierte Richtlinien	Nein
Zugriffskontrolllisten (ACLs)	Nein
Auf Tags basierende Autorisierung	Ja

IAMMerkmal	Von Greengrass unterstützt?
Temporäre Anmeldeinformationen	Ja
Service-verknüpfte Rollen	Nein
Servicerollen	Ja

Einen allgemeinen Überblick darüber, wie andere AWS Dienste zusammenarbeitenIAM, finden Sie IAM im IAMBenutzerhandbuch unter AWS Dienste, die mit funktionieren.

Identitätsbasierte Richtlinien für AWS IoT Greengrass

Mit IAM identitätsbasierten Richtlinien können Sie zulässige oder verweigerte Aktionen und Ressourcen sowie die Bedingungen angeben, unter denen Aktionen zulässig 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 in der Referenz zu den IAM JSON Richtlinienelementen im IAMBenutzerhandbuch.

Aktionen

Administratoren können mithilfe von AWS JSON Richtlinien angeben, wer Zugriff auf was hat. Das bedeutet, welcher Prinzipal kann Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen.

Das `Action` Element einer JSON Richtlinie beschreibt die Aktionen, mit denen Sie den Zugriff in einer Richtlinie zulassen oder verweigern können. Richtlinienaktionen haben normalerweise denselben Namen wie der zugehörige AWS API Vorgang. Es gibt einige Ausnahmen, z. B. Aktionen, für die nur eine Genehmigung erforderlich ist und für die es keinen entsprechenden Vorgang gibt. API Es gibt auch einige Operationen, die mehrere Aktionen in einer Richtlinie erfordern. Diese zusätzlichen Aktionen werden als abhängige Aktionen bezeichnet.

Schließen Sie Aktionen in eine Richtlinie ein, um Berechtigungen zur Durchführung der zugeordneten Operation zu erteilen.

Bei Richtlinienaktionen wird das `greengrass:` Präfix vor der Aktion AWS IoT Greengrass verwendet. Um beispielsweise jemandem zu ermöglichen, die `ListGroups` API Operation zum Auflisten der Gruppen in seiner eigenen zu verwenden 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 durch Kommas, wie folgt:

```
"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 IAMBenutzerhandbuch unter Definierte Aktionen von.

Ressourcen

Administratoren können mithilfe von AWS JSON Richtlinien festlegen, wer Zugriff auf was hat. Das bedeutet, welcher Prinzipal kann Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen.

Das `Resource` JSON Richtlinienelement gibt das Objekt oder die Objekte an, für die die Aktion gilt. Anweisungen müssen entweder ein – `Resource`oder ein `NotResource`-Element enthalten. Es hat sich bewährt, eine Ressource mit ihrem Amazon-Ressourcennamen (ARN) anzugeben. Sie können dies für Aktionen tun, die einen bestimmten Ressourcentyp unterstützen, der als Berechtigungen auf Ressourcenebene bezeichnet wird.

Verwenden Sie für Aktionen, die keine Berechtigungen auf Ressourcenebene unterstützen, z. B. Auflistungsoperationen, einen Platzhalter (*), um anzugeben, dass die Anweisung für alle Ressourcen gilt.

```
"Resource": "*"
```

Die folgende Tabelle enthält die AWS IoT Greengrass RessourceARNs, die im `Resource` Element einer Richtlinienerklärung verwendet werden kann. Eine Zuordnung der unterstützten Berechtigungen für AWS IoT Greengrass Aktionen auf Ressourcenebene finden Sie AWS IoT Greengrass im IAMBenutzerhandbuch unter Definierte Aktionen von.

Ressource	ARN
Group	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/groups/${GroupId}`
GroupVersion	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/groups/${GroupId}/versions/${VersionId}`
CertificateAuthority	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/groups/${GroupId}/certificateauthorities/${CertificateAuthorityId}`
Deployment	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/groups/${GroupId}/deployments/${DeploymentId}`
BulkDeployment	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/bulk/deployments/${BulkDeploymentId}`
ConnectorDefinition	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/connectors/${ConnectorDefinitionId}`
ConnectorDefinitionVersion	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/connectors/${ConnectorDefinitionId}/versions/${VersionId}`

Ressource	ARN
CoreDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/cores/${CoreDefinitionId}
CoreDefinitionVersion	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/cores/${CoreDefinitionId}/versions/${VersionId}
DeviceDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/devices/${DeviceDefinitionId}
DeviceDefinitionVersion	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/devices/${DeviceDefinitionId}/versions/${VersionId}
FunctionDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/functions/${FunctionDefinitionId}
FunctionDefinitionVersion	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/functions/${FunctionDefinitionId}/versions/${VersionId}
LoggerDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/loggers/${LoggerDefinitionId}
LoggerDefinitionVersion	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/loggers/${LoggerDefinitionId}/versions/${VersionId}
ResourceDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/resources/${ResourceDefinitionId}

Ressource	ARN
ResourceDefinitionVersion	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/resources/${ResourceDefinitionId}/versions/${VersionId}`
SubscriptionDefinition	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/subscriptions/${SubscriptionDefinitionId}`
SubscriptionDefinitionVersion	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/subscriptions/${SubscriptionDefinitionId}/versions/${VersionId}`
ConnectivityInfo	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/things/${ThingName}/connectivityInfo`

Das folgende `Resource` Beispielelement gibt den Wert einer Gruppe in ARN der Region USA West (Oregon) in der folgenden Region an: 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 wie folgt durch Kommas:

```
"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 kann Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen.

Das Element `Condition` (oder `Condition` block) ermöglicht Ihnen die Angabe der Bedingungen, unter denen eine Anweisung wirksam ist. Das Element `Condition` ist optional. 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.

Wenn Sie mehrere `Condition`-Elemente in einer Anweisung oder mehrere Schlüssel in einem einzelnen `Condition`-Element angeben, wertet AWS diese mittels einer logischen AND-Operation aus. Wenn Sie mehrere Werte für einen einzelnen Bedingungsschlüssel angeben, AWS wertet die Bedingung mithilfe einer logischen OR Operation aus. Alle Bedingungen müssen erfüllt werden, bevor die Berechtigungen der Anweisung gewährt werden.

Sie können auch Platzhaltervariablen verwenden, wenn Sie Bedingungen angeben. Sie können einem IAM Benutzer beispielsweise nur dann Zugriff auf eine Ressource gewähren, wenn sie mit seinem IAM Benutzernamen gekennzeichnet ist. Weitere Informationen finden Sie im IAMBenutzerhandbuch unter IAMRichtlinienelemente: Variablen und Tags.

AWS unterstützt globale Bedingungsschlüssel und dienstspezifische Bedingungsschlüssel. Eine Übersicht aller AWS globalen Bedingungsschlüssel finden Sie unter Kontext-Schlüssel für AWS globale Bedingungen im IAMBenutzerhandbuch.

AWS IoT Greengrass unterstützt die folgenden globalen Bedingungsschlüssel.

Schlüssel	Beschreibung
`aws:CurrentTime`	Filtert den Zugriff durch Prüfen der Datum/Uhrzeit-Bedingungen für das aktuelle Datum und die aktuelle Uhrzeit.

Schlüssel	Beschreibung
`aws:EpochTime`	Filtert den Zugriff durch Prüfen der Datum/Uhrzeit-Bedingungen für das aktuelle Datum und die aktuelle Uhrzeit in Epoch- oder Unix-Zeit.
`aws:MultiFactorAuthAge`	Filtert den Zugriff, indem geprüft wird, wie lange es her ist (in Sekunden), dass die Sicherheitsanmeldedaten, die durch die Multi-Faktor-Authentifizierung (MFA) in der Anfrage validiert wurden, ausgestellt wurden. MFA
`aws:MultiFactorAuthPresent`	Filtert den Zugriff, indem geprüft wird, ob die Multi-Faktor-Authentifizierung (MFA) verwendet wurde, um die temporären Sicherheitsanmeldeinformationen zu validieren, mit denen die aktuelle Anfrage gestellt wurde.
`aws:RequestTag/${TagKey}`	Filtert Anfragen zum Erstellen basierend auf den zulässigen Wertesätzen für jedes einzelne obligatorische Tag.
`aws:ResourceTag/${TagKey}`	Filtert Aktionen basierend auf dem Tag-Wert, der der Ressource zugeordnet ist.
`aws:SecureTransport`	Filtert den Zugriff, indem geprüft wird, ob die Anfrage mit SSL gesendet wurde.
`aws:TagKeys`	Filtert Anfragen zum Erstellen basierend auf dem Vorhanden sein obligatorischer Tags in der Anfrage.
`aws:UserAgent`	Filtert den Zugriff nach der Client-Anwendung des Anforderers.

Weitere Informationen finden Sie im IAMBenutzerhandbuch unter Kontexttasten für AWS globale Bedingungen.

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 Basis von AWS IoT Greengrass -Tags

Sie können Tags an unterstützte AWS IoT Greengrass Ressourcen anhängen oder Tags in einer Anfrage an übergeben AWS IoT Greengrass. Um den Zugriff basierend auf Tags zu steuern, stellen Sie Tag-Informationen im Bedingungselement einer Richtlinie unter Verwendung der Bedingungsschlüssel `aws:ResourceTag/${TagKey}`, `aws:RequestTag/${TagKey}` oder `aws:TagKeys` bereit. Weitere Informationen finden Sie unter Markieren Ihrer Greengrass-Ressourcen.

IAMRollen für AWS IoT Greengrass

Eine IAMRolle ist eine Entität innerhalb von Ihnen AWS-Konto , die über bestimmte Berechtigungen verfügt.

Verwenden temporärer Anmeldeinformationen mit AWS IoT Greengrass

Temporäre Anmeldeinformationen werden verwendet, um sich bei einem Verband anzumelden, eine IAM Rolle zu übernehmen oder eine kontoübergreifende Rolle anzunehmen. Sie erhalten temporäre Sicherheitsanmeldedaten, indem Sie AWS STS API Operationen wie AssumeRoleoder GetFederationTokenaufrufen.

Auf dem Greengrass-Kern werden temporäre Anmeldeinformationen für die Gruppenrolle für benutzerdefinierte Lambda-Funktionen und -Konnektoren verfügbar gemacht. Wenn Ihre Lambda-Funktionen das verwenden AWS SDK, müssen Sie keine Logik hinzufügen, um die Anmeldeinformationen abzurufen, da der dies für Sie AWS SDK 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 dem Konto. Das 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”.

Eine IAM Rolle in der AWS IoT Greengrass Konsole auswählen

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 eineAWS Identity and Access Management(IAM) -Servicerolle, die autorisiertAWS IoT Greengrassum auf Ressourcen zuzugreifen vonAWS-Services in Ihrem Namen. Dadurch kann AWS IoT Greengrass grundlegende Aufgaben durchführen, wie z. B. das Abrufen Ihrer AWS Lambda-Funktionen und das Verwalten von AWS IoT-Schatten.

ZulassenAWS IoT Greengrassum auf Ihre Ressourcen zugreifen zu können, müssen die Greengrass-Servicerolle mit IhremAWS-Kontound spezifizierenAWS IoT Greengrassals vertrauenswürdige Entität. Die -Rolle muss die AWSGreengrassResourceAccessRolePolicyverwaltete Richtlinie oder eine benutzerdefinierte Richtlinie, die gleichwertige Berechtigungen für dieAWS IoT GreengrassFunktionen, die Sie verwenden. Diese Richtlinie wird beibehalten vonAWSund definiert den Satz von Berechtigungen, dieAWS IoT Greengrassverwendet, um auf IhreAWSRessourcen schätzen.

Sie können dieselbe Greengrass-Servicerolle inAWS-Regions, aber Sie müssen es in jedem Fall mit Ihrem Konto verknüpfenAWS-Regionwo du verwendestAWS IoT Greengrassaus. Die Gruppenbereitstellung schlägt fehl, wenn die Servicerolle im aktuellenAWS-Kontound Region.

In den folgenden Abschnitten wird beschrieben, wie Sie die Greengrass-Servicerolle in der AWS Management Console oder AWS CLI erstellen und verwalten.

- Verwaltung der Servicerolle (Konsole)
- Verwaltung der Servicerolle (CLI)

> **Note**
>
> Zusätzlich zur Servicerolle, die den Zugriff auf Service-Ebene autorisiert, können Sie eine-Gruppen-Rollezu einemAWS IoT GreengrassGruppe. Die Gruppenrolle ist eine separate IAM-Rolle, die steuert, wie Greengrass-Lambda-Funktionen und -Konnektoren in der Gruppe darauf zugreifen können.AWS-Services.

Verwalten der Greengrass-Servicerolle (Konsole)

Die AWS IoT-Konsole vereinfacht die Verwaltung Ihrer Greengrass-Servicerolle. Wenn Sie beispielsweise eine Greengrass-Gruppe erstellen oder bereitstellen, prüft die Konsole, ob IhreAWS-Kontoist an eine Greengrass-Servicerolle imAWS-Regiondie derzeit in der Konsole ausgewählt ist. 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 dasAWS IoT-Konsole für die folgenden Rollenverwaltungsaufgaben:

- 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)

Führen Sie die folgenden Schritte aus, um die Servicerolle zu findenAWS IoT Greengrassverwendet in der aktuellenAWS-Regionaus.

1. AusAWS IoTKonsoleNavigationsbereich, wählen SieEinstellungenaus.
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.

Property (Eigenschaft)	Wert
Name	`Greengrass_ServiceRole`
Trusted entity (Vertrauenswürdige Entität)	`AWS service: greengrass`
Richtlinie	AWSGreengrassResourceAccessRolePolicy

> **Note**
>
> Wenn Greengrass Device Setup die Servicerolle erstellt, lautet der Rollenname `GreengrassServiceRole_random-string`.

Wenn Sie eine Greengrass-Gruppe aus demAWS IoT-Konsole prüft die Konsole, ob eine Greengrass-Servicerolle mit IhremAWS-KontoimAWS-Regiondie derzeit in der Konsole ausgewählt ist. Andernfalls werden Sie von der Konsole aufgefordert, zuzulassenAWS IoT Greengrasslesen und schreibenAWS-Services in Ihrem Namen.

Wenn Sie die Berechtigung erteilen, prüft die Konsole, ob eine Rolle mit`Greengrass_ServiceRole`existiert in deinemAWS-Kontoaus.

- Wenn die Rolle vorhanden ist, fügt die Konsole die Servicerolle anAWS-Kontoim aktuellenAWS-Regionaus.
- Wenn die Rolle nicht vorhanden ist, erstellt die Konsole eine standardmäßige Greengrass-Servicerolle und fügt sie an Ihre anAWS-Kontoim aktuellenAWS-Regionaus.

> **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 unterErstellen einer Rolle zum Delegieren von Berechtigungen an eineAWSBedienungoderÄndern einer RolleimIAM User Guideaus. Stellen Sie sicher, dass die Rolle Berechtigungen erteilt, die der verwalteten Richtlinie `AWSGreengrassResourceAccessRolePolicy` für die verwendeten Funktionen und Ressourcen entsprechen. Wir empfehlen Ihnen, auch die`aws:SourceArn`und`aws:SourceAccount`Globale -Bedingungskontextschlüssel in Ihrer Vertrauensrichtlinie, um dieconfused StellvertreterSicherheitsproblem. Die Bedingungskontextschlüssel schränken den Zugriff so ein, dass nur Anforderungen zugelassen werden, die vom angegebenen Konto und Greengrass-Arbeitsbereich kommen. Weitere Informationen über den „verwirrten Stellvertreter“ finden Sie unterVermeidung des Problems des verwirrten Stellvertreters (dienstübergreifend)aus.

Wenn Sie eine Servicerolle erstellen, kehren Sie zurAWS IoT-Konsole und fügen Sie die Rolle an die Gruppe an. Sie können dies unter Greengrass service role (Greengrass-Servicerolle) auf der Seite Settings (Einstellungen) der Gruppe vornehmen.

Ändern der Greengrass-Servicerolle (Konsole)

Mit dem folgenden Verfahren können Sie eine andere Greengrass-Servicerolle auswählen, die an Ihre angefügt werden sollAWS-KontoimAWS-Regionist derzeit in der -Konsole ausgewählt.

1. AusAWS IoTKonsoleNavigationsbereich, wählen SieEinstellungenaus.
2. UnderGreengrass-Servicerolle, wählenRolle ändernaus.

   DieAktualisieren der Greengrass-Servicerollewird geöffnet und zeigt die IAM-Rollen in IhremAWS-Kontodie definierenAWS IoT Greengrassals vertrauenswürdige Entität.
3. Wählen Sie die Greengrass-Servicerolle aus.
4. Klicken Sie auf-Rolle hinzufügenaus.

> **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. DieErstelle eine Rolle für michLink wird nicht angezeigt, wenn eine Rolle mit dem Namen`Greengrass_ServiceRole`eist in IhremAWS-Kontoaus.

Trennen der Greengrass-Servicerolle (Konsole)

Führen Sie die folgenden Schritte aus, um die Greengrass-Servicerolle von Ihrem zu trennenAWS-KontoimAWS-Regionist derzeit in der -Konsole ausgewählt. Dadurch werden Berechtigungen fürAWS IoT GreengrassZugriff aufAWSDienstleistungen in der aktuellenAWS-Regionaus.

> **Important**
>
> Durch das Trennen der Servicerolle können aktive Operationen unterbrochen werden.

1. AusAWS IoTKonsoleNavigationsbereich, wählen SieEinstellungenaus.
2. UnderGreengrass-Servicerolle, wählenTrennen der Rolleaus.
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 erlauben möglicherweise AWS IoT Greengrass den Zugriff auf Ihre Ressourcen. So finden Sie alle Rollen, die dies zulassenAWS IoT Greengrassum in Ihrem Namen Berechtigungen anzunehmen, in der IAM-Konsole, auf derRollennach Rollen suchen, die Folgendes beinhaltenAWSservice: greengrass/rotimTrusted Entitiescolumn.

Verwalten der Greengrass-Servicerolle (CLI)

Im folgenden Verfahren gehen wir davon aus, dass dieAWS CLIist installiert und konfiguriert für die Verwendung vonAWS-KontoID. Weitere Informationen finden Sie unterInstallieren vonAWS-Befehlszeilenschnittstelleund[Konfigurieren vonAWS CLI](#)imAWS Command Line Interface-Benutzerhandbuchaus.

Sie können die AWS CLI für die folgenden Rollenverwaltungsaufgaben verwenden:

- Abrufen Ihrer Greengrass-Servicerolle
- Erstellen der Greengrass-Servicerolle
- Entfernen der Greengrass-Servicerolle

Abrufen der Greengrass-Servicerolle (CLI)

Führen Sie die folgenden Schritte aus, um herauszufinden, ob eine Greengrass-Servicerolle mit Ihrem verknüpft istAWS-Kontoin einemAWS-Regionaus.

- Rufen Sie die Servicerolle ab. Ersetzen*Region*mit IhremAWS-Region(z. B.`us-west-2`) enthalten.

  ```
  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"
  }
  ```

  Werden keine Rollenmetadaten zurückgegeben, müssen Sie die Servicerolle erstellen (sofern sie noch nicht vorhanden ist) und Ihrem -Konto imAWS-Regionaus.

Erstellen der Greengrass-Servicerolle (CLI)

Führen Sie die folgenden Schritte aus, um eine Rolle zu erstellen und sie Ihrem zuzuweisenAWS-Kontoaus.

So erstellen Sie die Servicerolle mit IAM

1. Erstellen Sie die Rolle mit einer Vertrauensrichtlinie, die es AWS IoT Greengrass erlaubt, die Rolle anzunehmen. In diesem Beispiel wird eine Rolle namens `Greengrass_ServiceRole` erstellt, aber Sie können einen anderen Namen verwenden. Wir empfehlen Ihnen, auch die`aws:SourceArn`und`aws:SourceAccount`Globale -Bedingungskontextschlüssel in Ihrer Vertrauensrichtlinie, um dieconfused StellvertreterSicherheitsproblem. Die Bedingungskontextschlüssel schränken den Zugriff so ein, dass nur Anforderungen zugelassen werden, die vom angegebenen Konto und Greengrass-Arbeitsbereich kommen. Weitere Informationen über den „verwirrten Stellvertreter“ finden Sie unterVermeidung des Problems des verwirrten Stellvertreters (dienstübergreifend)aus.

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
```

So verknüpfen Sie die Servicerolle mit IhremAWS-Konto

- Weisen Sie die Rolle Ihrem Konto zu. Ersetzen*Rollen-ARN*mit der Servicerolle ARN und*Region*mit IhremAWS-Region(z. B.`us-west-2`) enthalten.

```
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)

Mit den folgenden Schritten können Sie die Zuweisung der Greengrass-Servicerolle von IhremAWS-Kontoaus.

- Haben Sie die Zuordnung der Servicerolle bei Ihrem Konto auf. Ersetzen*Region*mit IhremAWS-Region(z. B.`us-west-2`) enthalten.

```
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 keinerAWS-Regionaus. 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 an eineAWSBedienungimIAM User Guide
- Ändern einer RolleimIAM User Guide
- Löschen von Rollen oder Instance-ProfilenimIAM User Guide
- AWS IoT Greengrass-Befehle imAWS CLIBefehlsreferenz
  - associate-service-role-to-Konto
  - disassociate-service-role-from-Konto
  - get-service-role-for-Konto
- IAM-Befehle imAWS CLIBefehlsreferenz
  - 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, zum Zugriff auf IhreAWSRessourcen schätzen. Sie erstellen die Rolle und verwalten Berechtigungen inAWS Identity and Access Management(IAM) und fügen Sie die Rolle an Ihre Greengrass-Gruppe an. Eine Greengrass-Gruppe verfügt über eine Gruppenrolle. Um Berechtigungen hinzuzufügen oder zu ändern, können Sie eine andere Rolle anfügen oder die IAM-Richtlinien ändern, die der Rolle angefügt sind.

Die Rolle muss AWS IoT Greengrass als vertrauenswürdige Entität definieren. Je nach geschäftlicher Situation kann die Gruppenrolle IAM-Richtlinien enthalten, die Folgendes definieren:

- Berechtigungen für benutzerdefinierteLambda-FunktionenZugriff aufAWS-Services.
- Berechtigungen für Konnektoren zum Zugriff auf AWS-Services.

- Berechtigungen fürStream-Managerum Streams zu exportieren nachAWS IoT Analyticsund Kinesis Data Streams.
- Berechtigungen zum Zulassen der CloudWatch -Protokollierung.

In den folgenden Abschnitten wird beschrieben, wie eine Greengrass-Gruppenrolle in der AWS Management Console oder AWS CLI angefügt oder getrennt wird.

- 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-Servicesolle zuweisen, mit der AWS IoT Greengrass in Ihrem Auftrag auf AWS-Ressourcen zugreifen kann.

Verwalten der Greengrass-Gruppenrolle (Konsole)

Sie können dasAWS IoT-Rolle für die folgenden Rollenverwaltungsaufgaben:

- 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)

Führen Sie die Rolle aus, die einer Greengrass-Gruppe zu finden.

1. In derAWS IoTNavigationsbereich der -Konsole unterVerwalten, erweiternGreengrass-Geräteund wählen Sie dann ausGruppen (V1)aus.
2. Wählen Sie die Zielgruppe aus.
3. Wählen Sie auf der Gruppenkonfigurationsseite die Option ausEinstellungen anzeigenaus.

Wenn der Gruppe eine Rolle an die Gruppe ist, wird sie unterGruppenrolleaus.

Hinzufügen oder Ändern der Greengrass-Gruppenrolle (Konsole)

Führen Sie die Schritte aus, um eine IAM-Rolle aus IhremAWS-Kontoum einer Greengrass-Gruppe zu werden.

Für eine Gruppenrolle gelten folgende Anforderungen:

- AWS IoT Greengrass ist als vertrauenswürdige Entität definiert.
- Die der Rolle zugeordneten Berechtigungsrichtlinien müssen die Berechtigungen für IhreAWS-Ressourcen, die von den Lambda-Funktionen und Konnektoren in der Gruppe und von Greengrass-Systemkomponenten erfordert werden.

> **Note**
>
> Wir empfehlen Ihnen, auch die`aws:SourceArn`und`aws:SourceAccount`Globale -Bedingungskontextschlüssel in Ihrer Vertrauensrichtlinie, dieconfused StellvertreterSicherheitsproblem. Die Bedingungskontextschlüssel schränken den Zugriff so ein, dass nur Anforderungen zugelassen werden, die vom angegebenen Konto und Greengrass-Arbeitsbereich kommen. Weitere Informationen zum „verwirrten Stellvertreter“ finden Sie unterVermeidung des Problems des verwirrten Stellvertreters (dienstübergreifend)aus.

Verwenden Sie die IAM-Konsole, um die Rolle und ihre Berechtigungen zu erstellen und zu konfigurieren. Schritte zum Erstellen einer Beispielrolle, die den Zugriff auf eine Amazon DynamoDB-Tabelle ermöglicht, finden Sie unterthe section called “Konfigurieren der Gruppenrolle”aus. Allgemeine Schritte finden Sie unterErstellen einer Rolle für eineAWSService (Konsole)imIAM User Guideaus.

Nachdem die Rolle konfiguriert wurde, verwenden Sie dieAWS IoT, um die Rolle der Gruppe der Gruppe zu werden.

> **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. In derAWS IoTNavigationsbereich der -Konsole unterVerwalten, erweiternGreengrass-Geräteund wählen Sie dann ausGruppen (V1)aus.
2. Wählen Sie die Zielgruppe aus.
3. Wählen Sie auf der Gruppenkonfigurationsseite die Option ausEinstellungen anzeigenaus.
4. UnderGruppenrolleaus, fügen oder ändern Sie die Rolle aus, ob die Rolle
   - Wählen Sie zum Hinzufügen der RolleRolle zuordnenund wählen Sie dann Ihre Rolle aus Ihrer Rollenliste aus. Dies sind die Rollen in IhremAWS-Kontodie definierenAWS IoT Greengrassals vertrauenswürdige Entität.
   - Um eine andere Rolle zu wählen, wählen SieRolle bearbeitenund wählen Sie dann Ihre Rolle aus Ihrer Rollenliste aus.
5. Wählen Sie Save (Speichern) aus.

Entfernen der Greengrass-Gruppenrolle (Konsole)

Führen Sie die folgenden Schritte aus, um die Rolle von einer Greengrass-Gruppe zu trennen.

1. In derAWS IoTNavigationsbereich der -Konsole unterVerwalten, erweiternGreengrass-Geräteund wählen Sie dann ausGruppen (V1)aus.
2. Wählen Sie die Zielgruppe aus.
3. Wählen Sie auf der Gruppenkonfigurationsseite die Option ausEinstellungen anzeigenaus.
4. UnderGruppenrolle, wählenZuordnung der Rolleaus.

5. Wählen Sie im BestätigungsdialogfeldZuordnung der Rolleaus. 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. Ersetzen Sie *`group-id`* 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.

Erstellen Sie die Gruppenrolle mit IAM

1. Erstellen Sie die Rolle mit einer Vertrauensrichtlinie, die es AWS IoT Greengrass erlaubt, die Rolle anzunehmen. In diesem Beispiel wird eine Rolle namens `MyGreengrassGroupRole` erstellt, aber Sie können einen anderen Namen verwenden. Wir empfehlen Ihnen, auch die`aws:SourceArn`und`aws:SourceAccount`Globale -Bedingungskontextschlüssel in Ihrer Vertrauensrichtlinie, dieconfused StellvertreterSicherheitsproblem. Die Bedingungskontextschlüssel schränken den Zugriff so ein, dass nur Anforderungen zugelassen werden, die vom angegebenen Konto und Greengrass-Arbeitsbereich kommen. Weitere Informationen zum „verwirrten Stellvertreter“ finden Sie unterVermeidung des Problems des verwirrten Stellvertreters (dienstübergreifend)aus.

   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 von Amazon S3 liest, können Sie die`AmazonS3ReadOnlyAccess`verwaltete Richtlinie für die Rolle an.

```
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. Ersetzen Sie *`group-id`* 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. Ersetzen Sie *`group-id`* 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 imIAM User Guide
  - Erstellen einer Rolle zum Delegieren von Berechtigungen an eineAWSBedienung
  - Ändern einer Rolle
  - Hinzufügen und Entfernen von IAM-Identitätsberechtigungen
  - Löschen von Rollen oder Instance-Profilen
- AWS IoT Greengrass-Befehle imAWS CLIBefehlsreferenz
  - list-groups
  - associate-role-to-group
  - disassociate-role-from-group
  - get-associated-role

- IAM-Befehle imAWS CLIBefehlsreferenz
  - attach-role-policy
  - create-role
  - delete-role
  - delete-role-policy

Vermeidung des Problems des verwirrten Stellvertreters (dienstübergreifend)

Das Confused-Deputy-Problem ist ein Sicherheitsproblem, bei dem eine Entität, die nicht über die Berechtigung zum Ausführen einer Aktion verfügt, eine Entität mit größeren Rechten zwingen kann, die Aktion auszuführen. In AWS kann der dienstübergreifende Identitätswechsel zu Confused-Deputy-Problem 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 Verwendung der globalen Bedingungskontext-Schlüssel `aws:SourceArn` und `aws:SourceAccount` in ressourcenbasierten Richtlinien, um die Berechtigungen, die AWS IoT Greengrass einem anderen Service erteilt, auf eine bestimmte Ressource zu beschränken. 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 mit dem`sts:AssumeRole`request.

Der effektivste Weg, um sich vor dem Confused-Deputy-Problem zu schützen, ist die Verwendung des globalen Bedingungskontextschlüssels `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 Kontextbedingungsschlüssel mit Platzhaltern (`aws:SourceArn`) * für die unbekannten Teile des ARN. Beispiel: `arn:aws:greengrass:`*`region`*`:`*`account-id`*`:*`.

Beispiele für Richtlinien, die`aws:SourceArn`und`aws:SourceAccount`Globale - Bedingungskontextschlüssel finden Sie in den folgenden Themen:

- Erstellen der Greengrass-Servicerolle
- Erstellen der Greengrass-Gruppenrolle
- Erstellen und Konfigurieren einer IAM-Ausführungsrolle für Sammelbereitstellungen

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 ausführen, die die AWS Management Console-, AWS CLI- oder AWS-API benutzen. 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.

Bewährte Methoden 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. Befolgen Sie beim Erstellen oder Bearbeiten identitätsbasierter Richtlinien die folgenden Anleitungen und Empfehlungen:

- Erste Schritte mit AWS-verwaltete Richtlinien und Umstellung auf Berechtigungen mit den geringsten Berechtigungen – Um Ihren Benutzern und Workloads Berechtigungen zu gewähren, verwenden Sie die AWS-verwaltete Richtlinien die Berechtigungen für viele allgemeine Anwendungsfälle gewähren. Sie sind in Ihrem AWS-Konto verfügbar. Wir empfehlen Ihnen, die Berechtigungen weiter zu reduzieren, indem Sie AWS-kundenverwaltete Richtlinien definieren, die speziell auf Ihre Anwendungsfälle zugeschnitten sind. Weitere Informationen finden Sie unter AWS-verwaltete Richtlinien oder 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 Service-Aktionen zu gewähren, wenn diese durch ein bestimmtes AWS -Service, wie beispielsweise AWS CloudFormation, verwendet werden. 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 Richtlinienvalidierung zum IAM Access Analyzer im IAM-Benutzerhandbuch.
- Bedarf einer Multi-Faktor-Authentifizierung (MFA) – Wenn Sie ein Szenario haben, das IAM-Benutzer oder Root-Benutzer in Ihrem AWS-Konto erfordert, 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 Konfigurieren eines MFA-geschützten API-Zugriffs im IAM-Benutzerhandbuch.

Weitere Informationen zu bewährten Methoden in IAM finden Sie unter Bewährte Methoden für die Sicherheit in IAM im IAM-Benutzerhandbuch.

AWS Von verwaltete Richtlinien für AWS IoT Greengrass

AWS IoT Greengrassverwaltet die folgendenAWS verwalteten Richtlinien, mit denen Sie IAM-Benutzern und -Rollen Berechtigungen gewähren können.

Richtlinie	Beschreibung
AWSGreengrassFullAccess	Ermöglicht alle AWS IoT Greengrass-Aktionen für alle Ihre AWS-Ressourcen. Diese Richtlini

Richtlinie	Beschreibung
	e wird fürAWS IoT Greengrass Servicead ministratoren oder zu Testzwecken empfohlen.
AWSGreengrassReadOnlyAccess	Ermöglicht `List` und `Get` AWS IoT Greengras s-Aktionen für alle Ihre AWS-Ressourcen.
AWSGreengrassResourceAccessRolePolicy	Ermöglicht den Zugriff auf Ressourcen vonAWS Diensten wieAWS LambdaAWS IoT Device Shadow. Dies ist die Standardr ichtlinie, die für die Greengrass-Servicerolle verwendet wird. Diese Richtlinie dient allgemein zur Erleichterung des Zugriffs. Sie können eine benutzerdefinierte Richtlinie einrichten, die restriktiver ist.
Grünes Gras OtaUpdateArtifactAccess	Ermöglicht den schreibgeschützten Zugriff auf over-the-air (OTA) -Aktualisierungsartefakte für dieAWS IoT Greengrass Core-Software in allenAWS-Region Versionen.

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 enthält Berechtigungen für die Ausführung dieser Aktion auf der Konsole oder für die programmgesteuerte Ausführung über die AWS CLI oder die AWS-API.

```
{
    "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": "*"
        }
    ]
}
```

Problembehandlung bei 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 derFehler: 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-KontoAWS 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 derFehler: 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. Weitere Informationen finden Sie unter 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 Greengrassübergeben 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-KontoAWS 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).

Compliance-Validierung 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. AWS stellt die folgenden Ressourcen zur Verfügung, die Sie bei der Einhaltung der Vorschriften unterstützen:

- Schnellstartanleitungen zu Sicherheit und Compliance — In diesen Bereitstellungsleitfäden werden architektonische Überlegungen erörtert und Schritte für die Bereitstellung von Basisumgebungen beschrieben AWS , bei denen Sicherheit und Compliance im Mittelpunkt stehen.
- Architecting for HIPAA Security and Compliance on Amazon Web Services — In diesem Whitepaper wird beschrieben, wie Unternehmen Anwendungen erstellen HIPAA können, die AWS für sie in Frage kommen.

> **Note**
>
> Nicht alle sind berechtigt AWS -Services . HIPAA Weitere Informationen finden Sie in der Referenz für HIPAA qualifizierte Dienste.

- AWS Ressourcen zur AWS von Vorschriften — Diese Sammlung von Arbeitsmappen und Leitfäden kann auf Ihre Branche und Ihren Standort zutreffen.
- AWS Leitfäden zur Einhaltung von Vorschriften für Kunden — Verstehen Sie das Modell der gemeinsamen Verantwortung aus dem Blickwinkel der Einhaltung von Vorschriften. In den Leitfäden werden die bewährten Verfahren zur Sicherung zusammengefasst AWS -Services und die Leitlinien für Sicherheitskontrollen in verschiedenen Frameworks (einschließlich des National Institute of Standards and Technology (NIST), des Payment Card Industry Security Standards Council (PCI) und der International Organization for Standardization (ISO)) zusammengefasst.
- Evaluierung von Ressourcen anhand von Regeln im AWS Config Entwicklerhandbuch — Der AWS Config Service bewertet, wie gut Ihre Ressourcenkonfigurationen den internen Praktiken, Branchenrichtlinien und Vorschriften entsprechen.
- AWS Security Hub— Auf diese AWS -Service Weise erhalten Sie einen umfassenden Überblick über Ihren internen Sicherheitsstatus. AWS Security Hub verwendet Sicherheitskontrollen, um Ihre AWS -Ressourcen zu bewerten und Ihre Einhaltung von Sicherheitsstandards und bewährten Methoden zu überprüfen. Eine Liste der unterstützten Services und Kontrollen finden Sie in der Security-Hub-Steuerungsreferenz.
- Amazon GuardDuty — Dies AWS -Service erkennt potenzielle Bedrohungen für Ihre Workloads AWS-Konten, Container und Daten, indem es Ihre Umgebung auf verdächtige und böswillige Aktivitäten überwacht. GuardDuty kann Ihnen helfen, verschiedene Compliance-Anforderungen zu erfüllen PCIDSS, z. B. durch die Erfüllung der Anforderungen zur Erkennung von Eindringlingen, die in bestimmten Compliance-Frameworks vorgeschrieben sind.
- AWS Audit Manager— Auf diese AWS -Service Weise können Sie Ihre AWS Nutzung kontinuierlich überprüfen, um das Risikomanagement und die Einhaltung von Vorschriften und Industriestandards zu vereinfachen.

Ausfallsicherheit in AWS IoT Greengrass

DieAWSIm Zentrum der globalen -Infrastruktur stehen die Regionen und Availability Zones (Verfügbarkeitszonen EACHAWS-Regionstellt mehrere physisch getrennte und isolierte Availability Zones bereit, die über hoch redundante Netzwerke mit niedriger Latenz und hohen Durchsätzen

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 über Amazon Web Services Services-Regionen und Availability Zones finden Sie unterAWSGlobale -Infrastrukturaus.

Zusätzlich zur globalen AWS-Infrastruktur stellt AWS IoT Greengrass verschiedene Funktionen bereit, um Ihren Anforderungen in Bezug auf Ausfallsicherheit und Datensicherung zu erfüllen.

- Wenn der Kern seine Verbindung zum Internet verliert, können die Client-Geräte weiterhin über das lokale Netzwerk kommunizieren.
- Sie können den Core so konfigurieren, dass nicht verarbeitete Nachrichten gespeichert werden, die fürAWS CloudZiele in einem lokalen Speichercache anstatt im Speicher gespeichert werden. Der lokale Speichercache kann über Neustarts des Kerns hinweg bestehen bleiben (z. B. nach einer Gruppenbereitstellung oder einem Neustart des Geräts), so dass AWS IoT Greengrass weiterhin Nachrichten verarbeiten kann, die für AWS IoT Core bestimmt sind. Weitere Informationen finden Sie unter the section called “MQTT-Nachrichtenwarteschlange”.
- Sie können den Kern 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 Protokolle in das lokale Dateisystem und in CloudWatch protokolle. Wenn der Kern seine Konnektivität verliert, kann die lokale Protokollierung fortgesetzt werden,aber CloudWatch -Protokolle werden mit einer begrenzten Anzahl von Wiederholungsversuchen gesendet. Wenn die Zahl der zulässigen Wiederholungen erreicht ist, wird das Ereignis entfernt. YDu solltestebenfallsseien Sie sich bewusstEinschränkungen für die Protokolleaus.
- Sie können Lambda-Funktionen erstellen, die lesenStream-Managerstreamt und sendet die Daten an lokale Speicherziele.

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:

- Sicherheit auf Transportschicht (TLS). Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Cipher-Suites mit perfekter Vorwärtsgeheimhaltung (PFS) wie (Ephemeral Diffie-Hellman) oder DHE (Elliptic Curve Ephemeral Diffie-Hellman). ECDHE Die meisten modernen Systeme wie Java 7 und höher unterstützen diese Modi.

Darüber hinaus müssen Anfragen mithilfe einer Zugriffsschlüssel-ID und eines geheimen Zugriffsschlüssels, der einem Prinzipal zugeordnet ist, signiert werden. IAM Alternativ können Sie mit AWS Security Token Service (AWS STS) temporäre Sicherheitsanmeldeinformationen erstellen, um die Anforderungen zu signieren.

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”.

Konfigurations- 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, indem es Tools bereitstellt, die Sicherheitsprobleme und Abweichungen von bewährten Methoden identifizieren. Sie können mit AWS IoT Device Defender verbundene Geräte analysieren, prüfen und überwachen, um abnormales Verhalten zu erkennen und Sicherheitsrisiken zu mindern. AWS IoT Device Defender kann Geräte prüfen, um sicherzustellen, dass sie die bewährten Sicherheitsmethoden einhalten, und abnormales Verhalten auf Geräten erkennen. Dies ermöglicht es, konsistente Sicherheitsrichtlinien für alle Ihre

Geräte durchzusetzen und schnell zu reagieren, wenn Geräte gefährdet sind. In Verbindung mit AWS IoT Core generiert AWS IoT Greengrass prognostizierbare Client-IDs, die 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 die folgenden Überlegungen berücksichtigen:

- Es ist Ihre 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 möglich, dass Lambda-Funktionen mit jedem anderen Prozess kommunizieren, der im System oder außerhalb des Netzwerks ausgeführt wird.

Wenn Sie die Kontrolle über ein Greengrass-Core-Gerät verlieren und 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 Seite Einstellungen der Gruppe rotieren. 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 herstellenAWS IoT GreengrassSteuerebene durch Erstellen einerSchnittstellen-VPC-Endpunktaus. Sie können diesen Endpunkt verwenden, um Gruppen, Lambda-Funktionen, Bereitstellungen und andere Ressourcen imAWS IoT GreengrassService-Service. Schnittstellenendpunkte werden unterstützt vonAWS PrivateLink, einer Technologie, die den Zugriff ermöglichtAWS IoT Greengrass-APIs privat ohne Internet-Gateway, NAT-Gerät, VPN-Verbindung oderAWSDirect Connect-Verbindung. Die Instances in Ihrer VPC benötigen für die Kommunikation mit AWS IoT Greengrass-APIs keine öffentlichen

IP-Adressen. Datenverkehr zwischen Ihrer VPC und AWS IoT Greengrass verlässt das Amazon-Netzwerk nicht.

> **Note**
>
> Derzeit können Sie Greengrass-Kerngeräte nicht so konfigurieren, dass sie vollständig innerhalb Ihrer VPC arbeiten.

Jeder Schnittstellenendpunkt wird durch eine oder mehrere Elastic Network-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 eines Schnittstellen-VPC-Endpunkts fürAWS IoT GreengrassOperationen auf Steuerebene
- Erstellen einer VPC-Endpunktrichtlinie für AWS IoT Greengrass

Überlegungen zu AWS IoT Greengrass-VPC-Endpunkten

Bevor Sie einen Schnittstellen-VPC-Endpunkt für einrichtenAWS IoT Greengrass, PrüfenEigenschaften und Beschränkungen von SchnittstellenendpunktenimAmazon VPC User Guideaus. Dabei sollten Sie außerdem Folgendes beachten:

- AWS IoT Greengrassunterstützt Aufrufe all seiner API-Aktionen auf Steuerebene aus Ihrer VPC. Die Steuerebene umfasst Operationen wieCreateDeploymentundStartBulkDeploymentaus. Die - Steuerebene tutnichtbeinhalten Operationen wieGetDeploymentundErkennen, die Operationen auf der Datenebene sind.
- VPC-Endpunkte fürAWS IoT Greengrasswerden derzeit nicht unterstützt inAWSChina-Regionen.

Erstellen eines Schnittstellen-VPC-Endpunkts fürAWS IoT GreengrassOperationen auf Steuerebene

Sie können VPC-Endpunkte fürAWS IoT GreengrassSteuerebene mit der Amazon VPC-Konsole oderAWS Command Line Interface(AWS CLI) enthalten. Weitere Informationen finden Sie unter Erstellung eines Schnittstellenendpunkts im Benutzerhandbuch für Amazon VPC.

Erstellen Sie einen VPC-Endpunkt für AWS IoT Greengrass mit dem folgenden Servicenamen:

- com.amazonaws.*region*.greengrass

Wenn Sie einen privaten DNS für den Endpunkt aktivieren, können Sie mittels seines standardmäßigen DNS-Namen für die Region, beispielsweise `greengrass.us-east-1.amazonaws.com`, API-Anforderungen an AWS IoT Greengrass senden. 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 eine Endpunktrichtlinie an Ihren VPC-Endpunkt anhängen, der den Zugriff aufAWS IoT GreengrassOperationen auf Steuerebene. Die Richtlinie gibt die folgenden Informationen an:

- Prinzipal, der die Aktionen ausführen kann
- Die Aktionen, die der Prinzipal ausführen kann.
- Die Ressourcen, auf denen der Prinzipal Aktionen ausführen kann.

Weitere Informationen finden Sie unter Steuerung des Zugriffs auf Services mit VPC-Endpunkten im Amazon-VPC-Benutzerhandbuch.

Example Beispiel: VPC-Endpunktrichtlinie fürAWS IoT GreengrassAktionen

Im Folgenden finden Sie ein Beispiel für eine Endpunktrichtlinie für AWS IoT Greengrass. Wenn diese Richtlinie an einen Endpunkt angefügt wird, gewährt sie Zugriff auf die aufgelisteten AWS IoT Greengrass-Aktionen für alle Prinzipale auf allen Ressourcen.

```
{
```

```
    "Statement": [
        {
            "Principal": "*",
            "Effect": "Allow",
            "Action": [
                "greengrass:CreateDeployment",
                "greengrass:StartBulkDeployment"
            ],
            "Resource": "*"
        }
    ]
}
```

Bewährte Methoden für die Sicherheit für AWS IoT Greengrass

Dieses Thema behandelt bewährte Methoden in Bezug auf die Sicherheit für AWS IoT Greengrass.

Erteilen von Mindestberechtigungen

Folgen Sie dem Prinzip der geringsten Berechtigung, indem Sie den Mindestsatz von 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 “Bewährte Methoden für Richtlinien”.

Die bewährte Methode mit den geringsten Berechtigungen gilt auch für AWS IoT Richtlinien, die Sie an Ihre Greengrass-Kern- und Clientgeräte anfügen.

Keine Hartcodierung von Anmeldeinformationen in Lambda-Funktionen

Kodieren Sie keine Anmeldeinformationen in Ihren benutzerdefinierten Lambda-Funktionen fest. So schützen Sie Ihre Anmeldeinformationen besser:

- Um mit AWS-Services zu interagieren, definieren Sie Berechtigungen für bestimmte Aktionen und Ressourcen in der Greengrass-Gruppenrolle.
- Verwenden Sie lokale Secrets, um Ihre Anmeldeinformationen zu speichern. Wenn die Funktion das AWS SDK verwendet, verwenden Sie auch Anmeldeinformationen aus der standardmäßigen Anbieterkette für Anmeldeinformationen.

Keine Protokollierung sensibler Informationen

Sie sollten die Protokollierung von Anmeldeinformationen und anderen persönlich identifizierbaren Informationen (PII) verhindern. Wir empfehlen Ihnen, die folgenden Sicherheitsmaßnahmen zu implementieren, obwohl für den Zugriff auf lokale Protokolle auf einem Core-Gerät Stammrechte erforderlich sind und für den Zugriff auf CloudWatch Protokolle IAM-Berechtigungen erforderlich sind.

- Verwenden Sie keine sensiblen Informationen in MQTT-Themenpfaden.
- Verwenden Sie keine sensiblen Informationen in Gerätenamen (Objektnamen), Typen und Attributen in der AWS IoT Core-Registrierung.
- Protokollieren Sie keine sensiblen Informationen in Ihren benutzerdefinierten Lambda-Funktionen.
- Verwenden Sie keine vertraulichen Informationen in den Namen und IDs von Greengrass-Ressourcen:
  - 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 Services, 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 Geräte-SDK oder die AWS IoT Greengrass Discovery-API verwenden, um Erkennungsinformationen abzurufen, die für die Verbindung und Authentifizierung mit dem Kern in derselben Greengrass-Gruppe verwendet werden. Zu den Erkennungsinformationen gehören:

- Verbindungsinformationen für den Greengrass-Kern, 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 Gruppenzertifizierungsstellenzertifikat, 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-Kern. 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 Validierung 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 Erkennungsvorgänge verlassen, um Kernkonnektivitätsinformationen und das CA-Gruppenzertifikat abzurufen, auch wenn der Kern eine statische IP-Adresse verwendet. Client-Geräte sollten diese Erkennungsinformationen nicht fest codieren.

Aktualisieren Sie die Erkennungsinformationen regelmäßig.

Client-Geräte sollten regelmäßig Erkennungen ausführen, um die Kernkonnektivitätsinformationen und das CA-Gruppenzertifikat zu aktualisieren. Wir empfehlen, dass Client-Geräte diese Informationen aktualisieren, bevor sie eine Verbindung mit dem Core herstellen. Da kürzere Dauern zwischen Erkennungsvorgängen Ihre potenzielle Risikozeit minimieren können, empfehlen wir Client-Geräte, die Verbindung regelmäßig zu trennen und erneut zu verbinden, um das Update auszulösen.

Wenn Sie die Kontrolle über ein Greengrass-Core-Gerät verlieren und 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 Seite Einstellungen der Gruppe rotieren. In der AWS IoT Greengrass -API können Sie die -CreateGroupCertificateAuthorityAktion 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 Methoden für die Sicherheit in AWS IoT Core im -AWS IoTEntwicklerhandbuch
- Zehn goldene Sicherheitsregeln für industrielle IoT-Lösungen im Internet der Dinge im AWS offiziellen Blog

Protokollieren und Überwachen in AWS IoT Greengrass

Die Überwachung ist ein wichtiger Teil der Aufrechterhaltung von Zuverlässigkeit, Verfügbarkeit und Performance von AWS IoT Greengrass und Ihren AWS-Lösungen. Sie sollten Überwachungsdaten aller Bestandteile Ihrer AWS-Lösung sammeln, damit Sie auftretende Multipunkt-Fehler leichter beheben können. Bevor Sie mit der Überwachung von AWS IoT Greengrass beginnen, sollten Sie einen Überwachungsplan mit Antworten auf die folgenden Fragen erstellen:

- Was sind Ihre Ziele bei der Überwachung?
- 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 bietet verschiedene Tools für die Überwachung von 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 Tools zur Überwachung von verwendenAWS IoT Greengrassund Probleme melden:

- Amazon CloudWatch Protokolle- Überwachen, Speichern und Zugriff auf Ihre Protokolldateien vonAWS CloudTrailoder andere Quellen. Weitere Informationen finden Sie unterÜberwachen von Protokolldateienim*Amazon CloudWatch -Benutzerhandbuch*aus.
- AWS CloudTrail-Protokollüberwachung- Protokolldateien zwischen Konten teilen, überwachen CloudTrail -Protokolldateien in Echtzeit, indem Sie sie an senden CloudWatch Protokolliert, schreiben Sie Anwendungen zur Protokollverarbeitung in Java und vergewissern Sie sich, dass nach der Lieferung durch CloudTrail keine Änderungen an den Protokolldaten vorgenommen wurden. Weitere Informationen finden Sie unterArbeiten mit CloudTrail Protokolldateienim*AWS CloudTrail-Benutzerhandbuch*aus.
- Amazon EventBridge— Verwendung von EventBridge -Ereignisregeln, um Benachrichtigungen über Statusänderungen für Ihre Greengrass-Gruppenbereitstellungen oder API-Aufrufe zu erhalten,

die mit CloudTrail protokolliert wurden. Weitere Informationen finden Sie unterthe section called “Abrufen von Bereitstellungsbenachrichtigungen”oderWas ist Amazon EventBridge?imAmazon EventBridge -Benutzerhandbuchaus.

- Greengrass Systemgesundheitstelemetrie— Abonnieren Sie, um Telemetriedaten zu erhalten, die vom Greengrass-Kern gesendet wurden. Weitere Informationen finden Sie unter the section called “Erfassung von Telemetriedaten zum Systemzustand”.
- Lokale Zustandsprüfung- Verwenden Sie die Zustandsprüfung, um einen Snapshot des Zustands vonAWS IoT Greengrass-Prozesse auf dem Core-Gerät. 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 “Protokollierung von AWS IoT Greengrass-API-Aufrufen mit AWS CloudTrail”
- the section called “Abrufen von Bereitstellungsbenachrichtigungen”

Überwachen mit AWS IoT Greengrass-Protokollen

AWS IoT Greengrass besteht aus dem Cloud-Service und der AWS IoT Greengrass Core-Software. Die AWS IoT Greengrass Core-Software kann Protokolle in Amazon CloudWatch und in das lokale Dateisystem Ihres Core-Geräts schreiben. Lambda-Funktionen und Konnektoren, die auf dem Core ausgeführt werden, können auch Protokolle in CloudWatch 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, die Protokollebene sowie 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 der Protokollierung für AWS IoT Greengrass”.

Zugreifen auf CloudWatch Protokolle

Wenn Sie die CloudWatch Protokollierung konfigurieren, können Sie die Protokolle auf der Seite Protokolle der Amazon- CloudWatch Konsole anzeigen. Protokollgruppen für AWS IoT Greengrass-Protokolle verwenden die folgenden Namenskonventionen:

```
/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 Protokollen gelten die folgenden Überlegungen:

- Protokolle werden mit einer begrenzten Anzahl von Wiederholungen an CloudWatch Logs 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 zulassenAWS IoT Greengrass, dass in CloudWatch Protokolle schreibt. Zum Erteilen der Berechtigungen betten Sie die folgende Inline-Richtlinie in Ihre Gruppenrolle ein.

  ```
  {
      "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 von identitätsbasierten Richtlinien (IAM-Richtlinien) für CloudWatch Protokolle im Amazon- CloudWatch Benutzerhandbuch.

Die Gruppenrolle ist eine IAM-Rolle, die Sie erstellen und an Ihre Greengrass-Gruppe anfügen. Sie können die Gruppenrolle mithilfe der Konsole oder der AWS IoT Greengrass-API verwalten.

Verwenden der Konsole

1. Erweitern Sie im Navigationsbereich der AWS IoTKonsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
2. Wählen Sie die Zielgruppe aus.
3. Wählen Sie Einstellungen anzeigen aus. Unter Gruppenrolle können Sie die Gruppenrolle anzeigen, zuordnen oder die 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 anzufügen.
- 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**
>
> Standardmäßig ist *greengrass-root* das /greengrass-Verzeichnis. Wenn ein Schreibverzeichnis konfiguriert wurde, finden Sie auch die Protokolle dort.

Die folgenden Hinweise gelten für die Verwendung von Dateisystemprotokollen:

- Für das Lesen von AWS IoT Greengrass-Protokollen auf dem Dateisystem sind Root-Berechtigungen erforderlich.
- AWS IoT Greengrass unterstützt eine größenbasierte Rotation und, wenn die Menge der Protokolldaten nahe am konfigurierten Limit ist, die automatische Bereinigung von Protokolldaten.
- Die crash.log-Datei ist nur in Dateisystemprotokollen vorhanden. Dieses Protokoll wird nicht in -CloudWatch Protokolle geschrieben.
- Es gelten Einschränkungen im Hinblick auf die Festplattennutzung. Weitere Informationen finden Sie unter the section called “Einschränkungen für die Protokollierung”.

> **Note**
>
> Protokolle für AWS IoT Greengrass Core-Software v1.0 werden im Verzeichnis *greengrass-root*/var/log gespeichert.

Standardkonfiguration für die Protokollierung

Wenn die Einstellungen für die Protokollierung nicht explizit konfiguriert sind, verwendet AWS IoT Greengrass die folgende Standardkonfiguration für die Protokollierung nach der ersten Gruppenbereitstellung.

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 der Protokollierung für AWS IoT Greengrass

Sie können die -AWS IoTKonsole oder die -AWS IoT GreengrassAPIs verwenden, um die AWS IoT Greengrass Protokollierung zu konfigurieren.

> **Note**
>
> Damit Protokolle AWS IoT Greengrass in CloudWatch Protokolle schreiben kann, muss Ihre Gruppenrolle die erforderlichen CloudWatch Protokollaktionen zulassen.

Konfigurieren der Protokollierung (Konsole)

Sie können die Protokollierung auf der Seite Settings (Einstellungen) der Gruppe konfigurieren.

1. Erweitern Sie im Navigationsbereich der AWS IoTKonsole 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 Seite Gruppenkonfiguration die Registerkarte Protokolle 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 aus.
   - 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 Konfigurationsmodal Protokolle bearbeiten die Option Greengrass-Systemprotokollebene oder Protokollebene der 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 die AWS IoT Greengrass-Logger-APIs verwenden, um die Protokollierung 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 verwendet `AWSCloudWatch` wird, werden Protokollereignisse an - CloudWatch Protokolle gesendet. 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 Protokollierung des Dateisystems `ERROR` und höher für AWS IoT Greengrass Systemkomponenten.
- Aktiviert die Protokollierung des Dateisystems `INFO` (und höher) für benutzerdefinierte Lambda-Funktionen.
- Aktiviert (und höher) die CloudWatch `INFO` 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

Bei AWS IoT Greengrass sind folgende Einschränkungen für die Protokollierung vorhanden:

Transaktionen pro Sekunde

Wenn die Protokollierung in aktiviert CloudWatch ist, stapelt die Protokollierungskomponente Ereignisse lokal, bevor sie an gesendet werden CloudWatch, sodass Sie mit einer Rate von mehr als fünf Anforderungen pro Sekunde pro Protokollstream protokollieren können.

Arbeitsspeicher

Wenn so konfiguriert AWS IoT Greengrass ist, dass Protokolle an gesendet werden, CloudWatch und eine Lambda-Funktion für einen längeren Zeitraum mehr als 5 MB/Sekunde protokolliert, füllt sich die interne Verarbeitungspipeline schließlich. Der theoretische Worst-Case beträgt 6 MB pro Lambda-Funktion.

Taktversatz

Wenn die Protokollierung in aktiviert CloudWatch ist, signiert die Protokollierungskomponente Anforderungen an CloudWatch mithilfe des normalen Signature Version 4-Signaturprozesses. Wenn die Systemzeit auf dem AWS IoT Greengrass Core-Gerät um mehr als 15 Minuten nicht synchron ist, werden die Anforderungen 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 gilt:

`greengrass-system-component-space`

Die maximale Größe des lokalen Speichers für AWS IoT Greengrass-Systemkomponentenprotokolle.

`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 Core-Gerät so konfiguriert ist, dass es sich nur bei anmeldet CloudWatch und keine Internetverbindung besteht, 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 geschrieben CloudWatch.

CloudTrail -Protokolle

AWS IoT Greengrass wird mit ausgeführtAWS CloudTrail, einem Service, der die Aktionen eines Benutzers, einer Rolle oder eines -AWSServices in aufzeichnetAWS IoT Greengrass. Weitere Informationen finden Sie unter the section called “Protokollierung von AWS IoT Greengrass-API-Aufrufen mit AWS CloudTrail”.

Protokollierung von AWS IoT Greengrass-API-Aufrufen mit AWS CloudTrail

AWS IoT Greengrass ist in integriert, einem ServiceAWS CloudTrail, der die Aktionen eines Benutzers, einer Rolle oder eines -AWSServices in aufzeichnetAWS IoT Greengrass. CloudTrail erfasst alle API-Aufrufe für AWS IoT Greengrass als Ereignisse. Zu den erfassten Aufrufen gehören Aufrufe von der AWS IoT Greengrass-Konsole und Code-Aufrufe der AWS IoT Greengrass-API-Operationen. Wenn Sie einen Trail erstellen, können Sie die kontinuierliche Bereitstellung von CloudTrail Ereignissen an einen Amazon S3-Bucket aktivieren, einschließlich Ereignissen für AWS IoT Greengrass. Wenn Sie keinen Trail konfigurieren, können Sie trotzdem die neuesten Ereignisse in der CloudTrail Konsole unter Ereignisverlauf anzeigen. Anhand der von CloudTrailgesammelten Informationen können Sie die an gestellte AnfrageAWS IoT Greengrass, die IP-Adresse, von der die Anfrage gestellt wurde, den Initiator der Anfrage, den Zeitpunkt der Anfrage und zusätzliche Details bestimmen.

Weitere Informationen zu CloudTrailfinden Sie im AWS CloudTrail -Benutzerhandbuch.

AWS IoT Greengrass -Informationen in CloudTrail

CloudTrail wird beim Erstellen des Kontos AWS-Konto auf Ihrem aktiviert. Wenn eine Aktivität in auftrittAWS IoT Greengrass, wird diese Aktivität in einem - CloudTrail Ereignis zusammen mit anderen -AWSServiceereignissen im Ereignisverlauf aufgezeichnet. Sie können in Ihrem AWS-Konto die neusten Ereignisse anzeigen, suchen und herunterladen. Weitere Informationen finden Sie unter Anzeigen von Ereignissen mit dem CloudTrail Ereignisverlauf .

Zur kontinuierlichen Aufzeichnung von Ereignissen in Ihrem AWS-Konto, einschließlich Ereignissen für AWS IoT Greengrass, erstellen Sie einen Trail. Ein Trail ermöglicht CloudTrail die Bereitstellung von Protokolldateien an einen Amazon S3-Bucket. Wenn Sie einen Trail in der Konsole erstellen, gilt dieser für alle AWS-Regionen. Der Trail protokolliert Ereignisse aus allen Regionen in der AWS-Partition und stellt die Protokolldateien in dem von Ihnen angegebenen Amazon-S3-Bucket bereit. Darüber hinaus können Sie andere -AWSServices konfigurieren, um die in den CloudTrail Protokollen erfassten Ereignisdaten weiter zu analysieren und entsprechend zu agieren. Weitere Informationen finden Sie hier:

- Übersicht zum Erstellen eines Trails
- CloudTrail Von unterstützte Services und Integrationen
- Konfigurieren von Amazon SNS-Benachrichtigungen für CloudTrail

- Empfangen von CloudTrail Protokolldateien aus mehreren Regionen und Empfangen von CloudTrail Protokolldateien aus mehreren Konten

Alle -AWS IoT GreengrassAktionen werden von protokolliert CloudTrail und sind in der AWS IoT Greengrass API-Referenz dokumentiert. Aufrufe der `CreateFunctionDefinition` Aktionen `AssociateServiceRoleToAccount`, `GetConnectivityInfo`, und generieren beispielsweise Einträge in den `GetGroupVersion` CloudTrail Protokolldateien.

Jeder Ereignis- oder Protokolleintrag enthält Informationen zu dem Benutzer, der die Anforderung generiert hat. Anhand der Identitätsinformationen zur Benutzeridentität können Sie Folgendes bestimmen:

- Ob die Anfrage mit Stammbenutzer- oder AWS Identity and Access Management (IAM)-Benutzeranmeldeinformationen ausgeführt wurde.
- Ob die Anforderung mit temporären Sicherheitsanmeldeinformationen für eine Rolle oder einen Verbundbenutzer ausgeführt wurde.
- Ob die Anforderung aus einem anderen AWS-Service gesendet wurde

Weitere Informationen finden Sie unter CloudTrail -Element userIdentity.

Grundlagen zu AWS IoT Greengrass-Protokolldateieinträgen

Ein Trail ist eine Konfiguration, die die Bereitstellung von Ereignissen als Protokolldateien an einen von Ihnen angegebenen Amazon 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, das Datum und die Uhrzeit der Aktion, Anforderungsparameter usw. CloudTrail Protokolldateien sind kein geordnetes Stacktrace der öffentlichen API-Aufrufe und erscheinen daher nicht in einer bestimmten Reihenfolge.

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 mit im Amazon-Benutzerhandbuch ausgelöst CloudTrail wird EventBridge
- AWS IoT Greengrass API-Referenz

Erfassung von Telemetriedaten zur Systemintegrität vonAWS IoT Greengrass Kerngeräten

Telemetriedaten zur Systemintegrität sind Diagnosedaten, mit denen Sie die Leistung kritischer Vorgänge auf Ihren Greengrass-Core-Geräten überwachen können. Der Telemetrie-Agent auf dem Greengrass-Kern sammelt lokale Telemetriedaten und veröffentlicht sie an Amazon, EventBridge ohne dass eine Kundeninteraktion erforderlich ist. Die Kerngeräte veröffentlichen Telemetriedaten EventBridge auf bestmögliche Weise. Beispielsweise können Kerngeräte Telemetriedaten möglicherweise nicht liefern, wenn sie offline sind.

> **Note**
>
> Amazon EventBridge ist ein Event Bus-Service, mit dem Sie Ihre Anwendungen mit Daten aus verschiedenen Quellen verbinden können. 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 Berichte zu erstellen. Fachexperten wie Verfahrenstechniker können diese Anwendungen nutzen, um Einblicke in den Zustand der Flotte zu gewinnen.

Um sicherzustellen, dass die Edge-Komponenten von Greengrass ordnungsgemäß funktionieren,AWS IoT Greengrass verwendet Greengrass die Daten für Entwicklungs- und Qualitätsverbesserungszwecke. Diese Funktion hilft auch bei der Entwicklung neuer und verbesserter Edge-Funktionen. AWS IoT Greengrassspeichert Telemetriedaten nur bis zu 7 Tage.

Diese Funktion ist in derAWS IoT Greengrass Core-Software v1.11.0 verfügbar und standardmäßig für alle Greengrass-Kerne aktiviert, einschließlich vorhandener Kerne. Sie erhalten automatisch Daten, sobald Sie auf dieAWS IoT Greengrass Core-Software v1.11.0 oder höher upgraden.

Weitere Informationen zum Zugriff auf oder die Verwaltung von veröffentlichten Telemetriedaten finden Sie unterthe section called “Abonnieren des Empfangs von Telemetriedaten”.

Der Telemetrie-Agent sammelt und veröffentlicht die folgenden Systemmetriken.

Telemetrie-Metrik

Name	Beschreibung	Quelle
`SystemMemUsage`	Die Speichermenge, 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 Durchläufe, die dazu führen, dass der Lambda-Funktion der Speicher ausgeht.	System (System)
`DroppedMessageCount`	Die Anzahl der verworfenen Nachrichten, für die bestimmt sindAWS IoT Core.	`GGCloudSpooler` Systemkomponente

Name	Beschreibung	Quelle
`LambdaTimeout`	Die Anzahl der Timeouts für die Ausführung der benutzerdefinierten Lambda-Funktion.	Benutzerdefinierte Lambda-Funktion und SystemAWS Cloud
`LambdaUngracefullyKilled`	Die Anzahl der Läufe, die die benutzerdefinierte Lambda-Funktion nicht abschließen kann.	Benutzerdefinierte Lambda-Funktion und SystemAWS Cloud
`LambdaError`	Die Anzahl der Läufe, die dazu führen, dass die benutzerdefinierte Lambda-Funktion Fehlerprotokolle schreibt.	Benutzerdefinierte Lambda-Funktion und SystemAWS Cloud
`BytesAppended`	Die Anzahl der Bytes der Daten, die an den Stream-Manager angehängt wurden.	`GGStreamManager` Systemkomponente
`BytesUploadedToIoTAnalytics`	Die Anzahl der Bytes der Daten, die der Stream-Manager in Kanäle exportiertAWS IoT Analytics.	`GGStreamManager` Systemkomponente
`BytesUploadedToKinesis`	Die Anzahl der Datenbytes, die Stream Manager in Streams in Amazon Kinesis Data Streams exportiert.	`GGStreamManager` Systemkomponente
`BytesUploadedToIoTSiteWise`	Die Anzahl der Bytes der Daten, in den der Stream Manager in Asset-Eigenschaften exportiertAWS IoT SiteWise.	`GGStreamManager` Systemkomponente

Name	Beschreibung	Quelle
`BytesUploadedToS3ExportTaskExecutor`	Die Anzahl der Bytes der Daten, die der Stream-Manager in Objekte in Amazon S3 exportiert.	`GGStreamManager` Systemkomponente
`BytesUploadedToHTTP`	Die Anzahl der Bytes der Daten, die der Stream-Manager nach HTTP exportiert.	`GGStreamManager` Systemkomponente

Konfigurieren der Telemetrie-Einstellungen

Die Greengrass-Telemetrie verwendet die folgenden Einstellungen:

- Der Telemetrie-Agent aggregiert die Telemetriedaten jede Stunde.
- Der Telemetrie-Agent veröffentlicht alle 24 Stunden eine Telemetrie-Nachricht.

> **Note**
>
> Die Einstellungen sind unveränderbar.

Sie können die Telemetrie-Funktion für ein Greengrass Core-Gerät aktivieren oder deaktivieren. AWS IoT Greengrassverwendet Schatten, um die Telemetriekonfiguration zu verwalten. Ihre Änderungen werden sofort wirksam, wenn der Core eine Verbindung zu hatAWS IoT Core.

Der Telemetrie-Agent veröffentlicht Daten mithilfe des MQTT-Protokolls mit einer Dienstgütestufe (QoS) von 0. Das bedeutet, dass die Zustellung nicht bestätigt wird oder die Veröffentlichungsversuche nicht erneut versucht werden. Telemetrie-Nachrichten teilen sich eine MQTT-Verbindung mit anderen Nachrichten für Abonnements, für die bestimmt sindAWS IoT Core.

Abgesehen von den Kosten für die DatenverbindungAWS IoT Core ist der Datentransfer vom Kern zum Server kostenlos. Das liegt daran, dass der Agent zu einemAWS reservierten Thema veröffentlicht. Je nach Anwendungsfall können Ihnen jedoch Kosten entstehen, wenn Sie die Daten erhalten oder verarbeiten.

Voraussetzungen

Für die Konfiguration der Telemetrieeinstellungen gelten die folgenden Anforderungen:

- Sie müssen dieAWS 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 Konfigurations-APIs aufzurufen, bevor Sie die Telemetrieeinstellungen aktualisieren.

  Mit der folgenden IAM-Beispielrichtlinie können Sie die Shadow- und Laufzeitkonfiguration eines bestimmten Kerns verwalten:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowManageShadow",
            "Effect": "Allow",
            "Action": [
                "iot:GetThingShadow",
                "iot:UpdateThingShadow",
                "iot:DeleteThingShadow",
                "iot:DescribeThing"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:thing/core-name-*"
            ]
        },
        {
            "Sid": "AllowManageRuntimeConfig",
            "Effect": "Allow",
            "Action": [
                "greengrass:GetCoreRuntimeConfiguration",
                "greengrass:UpdateCoreRuntimeConfiguration"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:thing/core-name"
```

```
            ]
        }
    ]
}
```

Sie können granularen oder bedingten Zugriff auf Ressourcen gewähren, indem Sie beispielsweise ein* Platzhalter-Benennungsschema verwenden. Weitere Informationen finden Sie unter Hinzufügen und Entfernen von IAM-Richtlinien im IAM-Benutzerhandbuch.

Konfigurieren Sie die Telemetrie-Einstellungen (Konsole)

Im Folgenden wird gezeigt, wie Sie die Telemetrieeinstellungen eines Greengrass-Kerns in derAWS IoT Greengrass Konsole aktualisieren.

1. Erweitern Sie im Navigationsbereich derAWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1).
2. Wählen Sie unter Greengrass-Gruppen Ihre Zielgruppe aus.
3. Wählen Sie auf der Gruppenkonfigurationsseite im Abschnitt Übersicht Ihren Greengrass-Kern aus.
4. Wählen Sie auf der Konfigurationsseite des Cores die Registerkarte Telemetrie.
5. Wählen Sie im Abschnitt Telemetrie zur Systemintegrität 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 dieAWS IoT Greengrass Core-Software v1.11.0 oder höher aktiviert.

Die Änderungen werden zur Laufzeit wirksam. Sie müssen die Gruppe nicht bereitstellen.

Telemetrieeinstellungen konfigurieren (CLI)

In derAWS IoT Greengrass API stellt das`TelemetryConfiguration` Objekt die Telemetrieeinstellungen eines Greengrass-Kerns dar. Dieses Objekt ist Teil des`RuntimeConfiguration` Objekts, das dem Kern zugeordnet ist. Sie können dieAWS IoT

Greengrass API oder dasAWS SDK verwendenAWS CLI, um die Greengrass-Telemetrie zu verwalten. Für die Beispiele in diesem Abschnitt wird die AWS CLI verwendet.

Um die Telemetrieeinstellungen zu überprüfen

Mit dem folgenden Befehl werden die Telemetrieeinstellungen eines Greengrass-Kerns abgerufen.

- *core-thing-name*Ersetzen Sie durch den Namen des Zielkerns.

  Um den Namen des Dings zu erhalten, verwenden Sie den get-core-definition-versionBefehl. 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"
        }
    }
}
```

So konfigurieren Sie die Telemetrieeinstellungen

Mit dem folgenden Befehl werden die Telemetrieeinstellungen für einen Greengrass Core aktualisiert.

- *core-thing-name*Ersetzen Sie durch den Namen des Zielkerns.

  Um den Namen des Dings zu erhalten, verwenden Sie den get-core-definition-versionBefehl. 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\"}}"
```

Falls ja, wurden Änderungen an den`ConfigurationSyncStatus` Telemetrieeinstellungen vorgenommen`InSync`. Die Änderungen werden zur Laufzeit wirksam. Sie müssen die Gruppe nicht bereitstellen.

TelemetryConfiguration Objekt

Das`TelemetryConfiguration` Objekt hat die folgenden Eigenschaften:

`ConfigurationSyncStatus`

Prüft, ob die Telemetrieeinstellungen synchronisiert sind. Sie können keine Änderungen an dieser Eigenschaft vornehmen.

Typ: Zeichenfolge

Gültige Werte: `InSync` oder `OutOfSync`.

`Telemetry`

Schaltet die Telemetrie ein oder aus. Der Standardwert ist `On`.

Typ: Zeichenfolge

Gültige Werte: `On` oder `Off`.

Abonnieren des Empfangs von Telemetriedaten

Sie können in Amazon Regeln erstellen EventBridge , die definieren, wie die vom Greengrass-Core-Gerät veröffentlichten Telemetriedaten verarbeitet werden. Beim EventBridge Empfang der Daten werden die in Ihren Regeln definierten Zielaktionen aufgerufen. Sie können zum Beispiel

Ereignisregeln erstellen, Ereignisinformationen speichern, Korrekturmaßnahmen ausführen oder andere Ereignisse auslösen.

Telemetrie-Event

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 aggregierter Datenpunkte, die die folgenden Eigenschaften haben:

`TS`

Erforderlich. Der Zeitstempel der Aggregation der Daten.

`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 metrische Wert. Der Telemetrie-Agent fügt der vorherigen Summe neue Werte hinzu, sodass die Summe ein ständig steigender Wert ist. Sie können den Zeitstempel verwenden, um den Wert einer bestimmten Aggregation zu ermitteln. Um beispielsweise den letzten aggregierten Wert zu finden, subtrahieren Sie den vorherigen Zeitstempelwert vom letzten Zeitstempelwert.

`U`

Die Einheit des metrischen Werts.

`ThingName`

Erforderlich. Der Name des Objekts, auf das Sie es abgesehen haben.

Voraussetzungen für das Erstellen von EventBridge Regeln

Führen Sie die folgenden Schritte ausAWS IoT Greengrass, bevor Sie eine EventBridge Regel für erstellen:

- Machen Sie sich mit den Ereignissen, Regeln und Zielen in vertraut EventBridge.
- Erstellen und konfigurieren Sie die Ziele, die anhand Ihrer 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)

Gehen Sie wie folgt vor, um eine EventBridge RegelAWS Management Console 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 aus einerAWS 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-Service.
4. Wählen Sie den Regeltyp und wählen Sie Regel mit einem Ereignismuster.
5. Wählen Sie Next (Weiter).
6. Wählen Sie als EreignisquelleAWS Ereignisse oder EventBridge Partnerveranstaltungen aus.
7. Wählen Sie unter Beispielereignis die Option AWSEreignisse und anschließend Greengrass Telemetry Data aus.
8. Treffen Sie unter Ereignismuster die folgenden Optionen:

   a. Als Event source (Ereignisquelle) wählen Sie AWS-Services aus.

b. Wählen Sie Greengrass für den AWSService.

c. Wählen Sie als Ereignistyp Greengrass Telemetry Data aus.

9. Wählen Sie Next (Weiter).
10. Wählen Sie für Ziel 1AWS Service aus.
11. Wählen Sie unter Ziel auswählen die Option SQS-Warteschlange aus.
12. Wählen Sie für Queue Ihre Funktion aus.

Erstellen Sie eine Ereignisregel zum Abrufen von Telemetriedaten (CLI)

Gehen Sie wie folgt vor, um eine EventBridge RegelAWS 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 dingname durch den Dingnamen* des Kerns.

     Um den Namen des Dings zu erhalten, verwenden Sie den get-core-definition-versionBefehl. 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.

   - Ersetzen Sie *queue-arn* 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 beiAWS IoT Greengrass Telemetrie

Verwenden Sie die folgenden Informationen, um Probleme mit der Konfiguration derAWS IoT Greengrass Telemetrie zu beheben.

Fehler: Die Antwort enthält "ConfigurationStatus„:"OutOfSync ", nachdem Sie den get-thing-runtime-configuration Befehl ausgeführt haben

Lösungen:

- DerAWS IoT Device Shadow-Dienst benötigt Zeit, um Aktualisierungen der Laufzeitkonfiguration zu verarbeiten und die Updates für das Greengrass-Core-Gerät bereitzustellen. Sie können warten und später überprüfen, ob die Telemetrieeinstellungen synchronisiert sind.
- Stellen Sie sicher, dass Ihr Kerngerät online ist.
- Aktivieren Sie Amazon CloudWatch Logs inAWS IoT Core, um den Schatten zu überwachen.
- Verwende AWS IoTMetriken, um dein Ding zu überwachen.

Aufruf der lokalen Health Check-API

AWS IoT Greengrassenthält eine lokale HTTP-API, die eine Momentaufnahme des aktuellen Status der lokalen Arbeitsprozesse liefert, die vonAWS IoT Greengrassaus. Dieser Snapshot umfasst benutzerdefinierte Lambda-Funktionen und Lambda-Systemfunktionen. System-Lambda-Funktionen sind Teil derAWS IoT GreengrassCore-Software. Sie laufen als lokale Worker Prozesse auf dem Core-Gerät und verwalten Operationen wie z. B. dem Nachrichtenrouting, der Synchronisierung des lokalen Schattens und der automatischen IP-Adressenerkennung.

Die Health Check-API unterstützt die folgenden Anfragen:

- Senden Sie eine`GET`Trequest anGesundheitsinformationen für alle Arbeitnehmer abrufenaus.
- Senden Sie eine`POST`Trequest anGesundheitsinformationen für spezifizierte Arbeitnehmer abrufenaus.

Anfragen werden lokal auf dem Gerät gesendet und benötigen keine Internetverbindung.

Holen Sie sich Gesundheitsinformationen für alle Arbeitnehmer

Senden Sie eine`GET`Anfrage, Gesundheitsinformationen über alle laufenden Arbeiter zu erhalten.

- Ersetzen*`Hafen`*mit der 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`file. Weitere Informationen finden Sie unter AWS IoT Greengrass Core-Konfigurationsdatei.

Beispielanforderung

Das folgende Beispiel`curl`request ruft Gesundheitsinformationen für alle Arbeitnehmer ab.

```
curl http://localhost:8000/2016-11-01/health/workers
```

JSON-Antwort

Diese Anforderung gibt ein Array vonInformationen zur Gesundheit der Arbeitnehmer&S3;

Beispielantwort

Die folgende Beispielantwort listet Integritätsinformationsobjekte für alle Arbeitsprozesse auf, die vonAWS IoT Greengrassaus.

```
[
    {
      "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"
    },
    ...
]
```

Holen Sie sich Gesundheitsinformationen über bestimmte Mitarbeiter

Senden Sie einePOSTAnfrage, Gesundheitsinformationen über bestimmte Arbeitnehmer zu erhalten. Ersetzen*Hafen*mit der Portnummer des IPC. Der Standardwert ist 8000.

```
POST http://localhost:port/2016-11-01/health/workers
```

Beispielanforderung

Das folgende Beispiel`curl`request ruft Gesundheitsinformationen für bestimmte Arbeitnehmer ab.

```
curl --data "@body.json" http://localhost:8000/2016-11-01/health/workers
```

Ein Beispiel`body.json`Anfrage Body:

```
{
    "FuncArns": [
        "arn:aws:lambda:::function:GGShadowService:1",
        "arn:aws:lambda:us-west-2:123456789012:function:my-lambda-function:3"
```

```
    ]
}
```

Der Anfragetext enthält eine`FuncArns`Array.

`FuncArns`

Eine Liste der Amazon-Ressourcennamen (ARNs) für die Lambda-Funktionen, die die Ziel-Worker darstellen.

- 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 ARNs auswählen, nach denen Sie abfragen möchten.
- Geben Sie für System-Lambda-Funktionen den ARN der entsprechenden Lambda-Funktion an. Weitere Informationen finden Sie unter the section called “System-Lambda-Funktionen”.

Typ: Zeichenfolge-Array

Mindestlänge: 1

Höchstlänge: Die Gesamtanzahl der Worker begann vonAWS IoT Greengrassauf dem Core-Gerät.

JSON-Antwort

Diese Anfrage gibt eine`Workers`-Array und ein`InvalidArns`Array.

`Workers`

Eine Liste von Objekten mit Gesundheitsinformationen für die angegebenen Arbeitskräfte.

Typ: Zeichenfolge-ArrayObjekte mit Gesundheitsinformationen

`InvalidArns`

Eine Liste von Funktions-ARNs, die ungültig sind, einschließlich Funktions-ARNs, denen keine Worker zugeordnet sind.

Typ: Zeichenfolge-Array

Beispielantwort

Die folgenden Beispielantwortlisten[Objekte mit Gesundheitsinformationen]für die angegebenen Arbeiter.

```
{
    "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 Anforderung gibt die folgenden Fehler zurück:

400 Ungültige Anfrage

Der Anfragetext ist schlecht geformt. Um dieses Problem zu beheben, senden Sie die Anfrage erneut:

```
{"FuncArns":["function-1-arn","function-2-arn"]}
```

400 Anfrage überschreitet die maximale Anzahl von Arbeitern

Die Anzahl der ARNs, die in der`FuncArns`-Array überschreitet die Anzahl der Worker.

Informationen zur Gesundheit der Arbeitnehmer

Ein Gesundheitsinformation-Objekt enthält die folgenden Eigenschaften:

`FuncArn`

Der ARN der System-Lambda-Funktion, die den Worker darstellt.

Typ: `string`

`WorkerId`

Die ID des Worker. 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 hochfährt.

Typ: `string`

`ProcessId`

Die Prozess-ID (PID) des Worker Prozesses.

Typ: `int`

`WorkerState`

Der Zustand des Worker.

Typ: `string`

Folgende Worker Stati 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, loslegen.

`FailedInitialization`

Initialisierung fehlgeschlagen.

`Terminated`

Stoppen von vom Greengrass-Daemon

`NotStarted`

Fehler beim Starten, erneuter Startversuch.

`Initialized`

Erfolgreich initialisiert.

System-Lambda-Funktionen

Sie können Gesundheitsinformationen für die folgenden Lambda-Funktionen des Systems anfordern:

`GGCloudSpooler`

Verwaltet die Warteschlange für MQTT-Nachrichten, dieAWS IoT Coreals Quelle oder Ziel.

ARN: `arn:aws:lambda:::function:GGCloudSpooler:1`

`GGConnManager`

Leitet MQTT-Nachrichten zwischen Greengrass Core- und Client-Geräten weiter.

ARN: `arn:aws:lambda:::function:GGConnManager`

`GGDeviceCertificateManager`

Hört aufAWS IoTshadow für Änderungen an den IP-Endpunkten des Cores und generiert das von GG verwendete serverseitige ZertifikatConnManager für die gegenseitige Authentifizierung.

ARN: `arn:aws:lambda:::function:GGDeviceCertificateManager`

`GGIPDetector`

Verwaltet die automatische Erkennung von IP-Adressen, die Geräte der Greengrass-Gruppe in die Lage versetzt, 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 durch benutzerdefinierte Lambda und Konnektoren.

ARN: `arn:aws:lambda:::function:GGSecretManager:1`

`GGShadowService`

Verwaltet lokale Schatten für Client-Geräte.

ARN: `arn:aws:lambda:::function:GGShadowService`

`GGShadowSyncManager`

Synchronisiert lokale Schatten mitAWS Cloudfür das Kerngerät und die Client-Geräte, wenn das Gerät`syncShadow`-Eigenschaft ist festgelegt auf`true`eaus.

ARN: `arn:aws:lambda:::function:GGShadowSyncManager`

`GGStreamManager`

Verarbeitet Datenströme lokal und führt automatische Exporte in dieAWS Cloudaus.

ARN: `arn:aws:lambda:::function:GGStreamManager:1`

`GGTES`

Der lokale Token-Austauschdienst, der IAM-Anmeldeinformationen abruft, die in der Greengrass-Gruppenrolle definiert sind und auf die lokaler Code zugreiftAWS-Services.

ARN: `arn:aws:lambda:::function:GGTES`

Markieren Ihrer AWS IoT Greengrass-Ressourcen

Mit Tags 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 nicht für die AWS IoT-Fakturierungsgruppen oder Kostenzuordnungsberichte unterstützt.

Grundlagen zu Tags (Markierungen)

Mit Tags können Sie Ihre AWS IoT Greengrass-Ressourcen kategorisieren (z. B. nach Zweck, Eigentümer und Umgebung). 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.

Markierungsunterstützung in der AWS IoT-Konsole

Sie können für die `Group`-Ressourcen in Greengrass in der AWS IoT-Konsole Markierungen (Tags) 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 die Markierung 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 auswählen. Wählen Sie im Abschnitt Tags für die Gruppe die Option Tags verwalten aus, um Gruppen-Tags hinzuzufügen, zu bearbeiten oder zu entfernen.

Markierungsunterstützung in der AWS IoT Greengrass-API

Sie können die AWS IoT Greengrass-API verwenden, um Tags für AWS IoT Greengrass-Ressourcen zu erstellen, aufzulisten und zu verwalten, die Markierungen 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 hinzufügen 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 die Ressourcen aufgeführt, die Sie in der AWS IoT Greengrass-API markieren können. Außerdem finden Sie die entsprechenden `Get`- und `Create`-Aktionen.

Ressource	Erstellen	Get
`Group`	`CreateGroup`	`GetGroup`
`ConnectorDefinition`	`CreateConnectorDefinition`	`GetConnectorDefinition`
`CoreDefinition`	`CreateCoreDefinition`	`GetCoreDefinition`
`DeviceDefinition`	`CreateDeviceDefinition`	`GetDeviceDefinition`

Ressource	Erstellen	Get
FunctionDefinition	CreateFunctionDefinition	GetFunctionDefinition
LoggerDefinition	CreateLoggerDefinition	GetLoggerDefinition
ResourceDefinition	CreateResourceDefinition	GetResourceDefinition
SubscriptionDefinition	CreateSubscriptionDefinition	GetSubscriptionDefinition
BulkDeployment	StartBulkDeployment	GetBulkDeploymentStatus

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-Objekten zuweisen können. Obwohl es sich bei Greengrass-Cores um AWS IoT-Objekte handelt,

werden die in diesem Thema beschriebenen Ressourcen-Tags dem `CoreDefinition`- und nicht dem Core-Objekt angefügt.

Verwenden von Tags mit IAM-Richtlinien

In Ihren IAM-Richtlinien können Sie Ressourcen-Tags verwenden, um den Benutzerzugriff und die Berechtigungen zu steuern. 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.

Bedingungskontextschlüssel und -werte können nur mit AWS IoT Greengrass-Aktionen verwendet werden, die bei einer markierbaren Ressource ausgeführt werden. 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 IAM-JSON-Richtlinienreferenz im IAM-Benutzerhandbuch. Die JSON-Richtlinienreferenz enthält detaillierte Syntax, Beschreibungen und Beispiele für die Elemente, Variablen und Auswertungslogik 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, auf Ressourcen zu reagieren, die nur das Tag env=beta haben.
- Die zweite Anweisung verhindert, dass ein IAM-Benutzer das Tag env=beta 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 hinzufügen 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.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "greengrass:*",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "greengrass: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.

```
{
    "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

- Markieren von AWS-Ressourcen in Allgemeine Amazon Web Services-Referenz.

AWS CloudFormation-Unterstützung für AWS IoT Greengrass

AWS CloudFormation ist ein Service, mit dessen Hilfe Sie Ihre AWS-Ressourcen erstellen, verwalten und replizieren können. Sie können -AWS CloudFormationVorlagen 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 AWS CloudFormation Vorlagen und Funktionen finden Sie unter Was ist AWS CloudFormation? im AWS CloudFormation -Benutzerhandbuch.

Erstellen von -Ressourcen.

Bei AWS CloudFormation-Vorlagen handelt es sich um JSON- oder YAML-Dokumente, in denen die Eigenschaften und Beziehungen von AWS-Ressourcen beschrieben werden. Folgende 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)

Die Struktur und Syntax von Greengrass-Ressourcen basieren in AWS CloudFormation-Vorlagen auf der AWS IoT Greengrass-API. Die Beispielvorlage ordnet beispielsweise eine oberste Ebene einem `DeviceDefinition` zu`DeviceDefinitionVersion`, der ein einzelnes Client-Gerät enthält. Weitere Informationen finden Sie unter the section called “Übersicht über das Gruppenobjektmodell”.

Die AWS IoT Greengrass Ressourcentypreferenz im AWS CloudFormation -Benutzerhandbuch beschreibt die Greengrass-Ressourcen, die Sie mit verwalten könnenAWS CloudFormation.

Wenn Sie AWS CloudFormation-Vorlagen verwenden, um Greengrass-Ressourcen zu erstellen, empfehlen wir, dass Sie sie nur aus AWS CloudFormation verwalten. Sie sollten Ihre Vorlage beispielsweise aktualisieren, wenn Sie ein Gerät hinzufügen, ändern oder entfernen möchten (anstatt die AWS IoT Greengrass API oder AWS IoTKonsole zu verwenden). Auf diese Weise können Sie Rollback und andere Funktionen der AWS CloudFormation-Änderungsverwaltung verwenden. Weitere Informationen zur Verwendung von AWS CloudFormation zum Erstellen und Verwalten Ihrer Ressourcen und Stacks finden Sie unter Arbeiten mit Stacks im AWS CloudFormation - Benutzerhandbuch.

Eine exemplarische Vorgehensweise, die zeigt, wie Sie AWS IoT Greengrass Ressourcen in einer -AWS CloudFormationVorlage erstellen und bereitstellen, finden Sie unter Automatisieren der AWS IoT Greengrass Einrichtung mit AWS CloudFormation im AWS offiziellen Blog zum Internet der Dinge.

Bereitstellen von Ressourcen

Nachdem Sie einen -AWS CloudFormationStack erstellt haben, der Ihre Gruppenversion enthält, können Sie die - AWS CLI oder -AWS IoTKonsole verwenden, um sie bereitzustellen.

> **Note**
>
> Um eine Gruppe bereitzustellen, müssen Sie Ihrem eine Greengrass-Servicerolle zugeordnet habenAWS-Konto. Die Servicerolle ermöglicht AWS IoT Greengrass den Zugriff auf Ihre Ressourcen in AWS Lambda und anderen -AWSServices. Diese Rolle sollte vorhanden sein, wenn Sie bereits eine Greengrass-Gruppe in der aktuellen bereitgestellt habenAWS-Region. 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 bei Stack-Erstellung mit Ihrer Gruppe und den Gruppenversions-IDs ausgeben.

So stellen Sie die Gruppe bereit (Konsole)

1. Erweitern Sie im Navigationsbereich der AWS IoTKonsole 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 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. Die Vorlage folgt dazu 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 die Zertifikat-ARNs für den Kern und das Gerät sowie den Versions-ARN der Lambda-Quellfunktion angeben können (bei der es sich um eine -AWS LambdaRessource handelt). In der Vorlage werden die intrinsischen Funktionen `GetAtt` und `Ref` verwendet, um auf IDs, ARNs sowie andere Attribute zu verweisen, die zum Erstellen von Greengrass-Ressourcen erforderlich sind.

Die Vorlage definiert auch zwei AWS IoT Geräte (Objekte), die das Core- und Client-Gerät darstellen, die der Greengrass-Gruppe hinzugefügt werden.

Nachdem Sie den Stack mit Ihren Greengrass-Ressourcen erstellt haben, können Sie die AWS CLI oder die -AWS IoTKonsole verwenden, um die Gruppe bereitzustellen.

> **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": "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": {
                        "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-Regions

Derzeit können Sie -AWS IoT GreengrassRessourcen nur in den folgenden erstellen und verwaltenAWS-Region:

- 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)
- Europe (London)
- AWS GovCloud (USA-West)

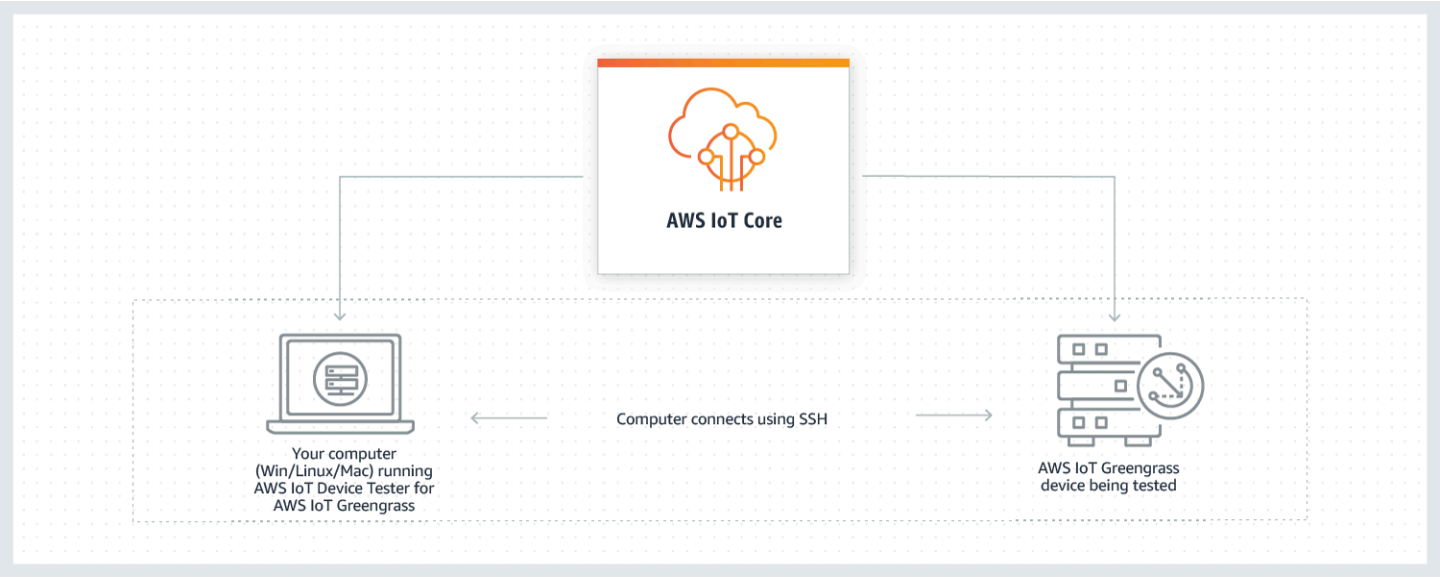
Verwenden von AWS IoT Device Tester für AWS IoT Greengrass V1

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 denAWS IoT Greengrass-Qualifizierungsprogramm.

Benutzerdefinierte Testsuiten

Ab IDT v4.0.0 AWS IoT Greengrass kombiniert IDT for 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 Test-Suiten zu entwickeln und zu betreiben.

Unterstützte Versionen von AWS IoT Device Tester für AWS IoT Greengrass V1

Da in den Wartungsmodus verschoben AWS IoT Greengrass Version 1 wurde, generiert IDT für AWS IoT Greengrass V1 keine signierten Qualifizierungsberichte mehr. Wir empfehlen, IDT für zu AWS IoT Greengrass V2verwenden.

Weitere Informationen zu IDT für AWS IoT Greengrass V2 finden Sie unter Verwenden von AWS IoT Device Tester für AWS IoT Greengrass V2 im AWS IoT Greengrass V2 -Entwicklerhandbuch.

> **Note**
>
> Beim Start eines Testlaufs erhalten Sie eine Benachrichtigung, wenn IDT für AWS IoT Greengrass nicht mit der von Ihnen verwendeten AWS IoT Greengrass-Version kompatibel ist.

Indem Sie diese Software herunterladen, stimmen Sie der Lizenzvereinbarung für AWS IoT Device Tester zu.

Nicht unterstützte IDT-Versionen für für AWS IoT Greengrass

In diesem Thema werden nicht unterstützte IDT-Versionen für AWS IoT Greengrass aufgeführt. 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 AWS IoT Greengrass Versionen v1.11.6, v1.10.5

Versionshinweise:

- Ermöglicht die Validierung und Qualifizierung von Geräten, auf denen die AWS IoT Greengrass -Core-Software v1.11.6 und v1.10.5 ausgeführt wird.
- Enthält kleinere Fehlerbehebungen.

Testsuite-Version:

`GGQ_1.3.1`

- Veröffentlicht am 2021.12.20

IDT v4.1.0 für AWS IoT Greengrass Versionen v1.11.4, v1.10.4

Versionshinweise:

- Ermöglicht die Validierung und Qualifizierung von Geräten, auf denen die AWS IoT Greengrass -Core-Software v1.11.4 und v1.10.4 ausgeführt wird.
- Behebt ein Problem, das dazu führte, dass die während eines Testlaufs angezeigten Protokolle redundante Tags verwendeten.

Testsuite-Version:

`GGQ_1.3.0`

- Veröffentlicht am 2021.06.23

- Fügt Wiederholungen für API-Aufrufe an Lambda, IAM und hinzu, AWS STS um die Handhabung von Drosselungs- oder Serverproblemen zu verbessern.
- Fügt Unterstützung für Python 3.8 zu den ML- und Docker-Testfällen hinzu.

IDT v4.0.2 für AWS IoT Greengrass die Versionen v1.11.1, v1.11.0, v1.10.3

Versionshinweise:

- Es wurde ein Problem behoben, das dazu führte, dass IDT Hardware Security Integration (HSI)-Fehler maskierte.
- Ermöglicht Ihnen die Entwicklung und Ausführung Ihrer benutzerdefinierten Testsuiten mit AWS IoT Device Tester für AWS IoT Greengrass. Weitere Informationen finden Sie unter Verwenden Sie IDT, um Ihre eigenen Test-Suiten zu entwickeln und zu betreiben.
- Stellt codesignierte IDT-Anwendungen für macOS und Windows bereit. Wenn unter macOS eine Sicherheitswarnung angezeigt wird, müssen Sie möglicherweise eine Sicherheitsausnahme für IDT erteilen. Weitere Informationen finden Sie unter Sicherheitsausnahme auf macOS.

> **Note**
>
> AWS IoT Greengrass stellt kein Dockerfile oder Docker-Image für Version 1.11.1 der -AWS IoT GreengrassCore-Software bereit. Verwenden Sie eine frühere Version der -AWS IoT GreengrassCore-Software, um Ihr Gerät auf Docker-Qualifizierung zu testen.

IDT v3.2.0 für AWS IoT Greengrass die Versionen v1.11.0, v1.10.1, v1.10.0

Versionshinweise:

- Standardmäßig führt IDT nur erforderliche Tests für die Qualifizierung durch. Um sich für zusätzliche Funktionen zu qualifizieren, können Sie die -`device.json`Datei ändern.
- Portnummer in hinzugefügt`device.json`, die Sie für SSH-Verbindungen konfigurieren können.
- Docker unterstützt nur Stream Manager und Machine Learning (ML) ohne Containerisierung. Container, Docker und Hardware Security Integration (HSI) sind für Docker-Geräte nicht verfügbar.
- Wir haben `device-ml.json` und `device-hsm.json` in zusammengeführt`device.json`.

IDT v3.1.3 für AWS IoT Greengrass Versionen: v1.10.x, v1.9.x, v1.8.x

Versionshinweise:

- Unterstützung für die ML-Funktionsqualifizierung für AWS IoT Greengrass v1.10.x und v1.9.x hinzugefügt. 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 ausgeführt wird. Wir empfehlen, diese Option während der Debugging-Phase auf Testgruppenebene zu verwenden.
- Es wurde eine Überprüfung der Uhrabweichung für MQTT-Tests hinzugefügt, um sicherzustellen, dass das getestete Gerät die richtige Systemzeit verwendet. Die verwendete Zeit muss innerhalb eines akzeptablen Zeitraums 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. Sie können diese Option verwenden, um die Antwort für die Aufforderung zum Aktualisieren der verwalteten Richtlinie festzulegen.
- 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 AWS IoT Greengrass v1.10.1 hinzugefügt.
- Automatische Updates der IDT-Test-Suite-Versionen. IDT kann die neuesten Test-Suites 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 for AWS IoT Greengrass v1.10, v1.9.x und v1.8.x

Beim Testen auf einem physischen Gerät werden AWS IoT Greengrass v1.10, v1.9.x und v1.8.x unterstützt.

Beim Testen in einem Docker-Container werden AWS IoT Greengrass v1.10 und v1.9.x unterstützt.

Versionshinweise:

- Unterstützung für the section called “Ausführen von AWS IoT Greengrass in einem Docker-Container” hinzugefügt. Sie können jetzt IDT verwenden, um zu qualifizieren und zu überprüfen, ob Ihre Geräte AWS IoT Greengrass in einem Docker-Container ausgeführt werden können.
- Es wurde eine von AWS verwaltete Richtlinie (`AWSIoTDeviceTesterForGreengrassFullAccess`) hinzugefügt, die die Berechtigungen definiert, die zum Ausführen von AWS IoT Device Tester erforderlich sind. Wenn neue Versionen zusätzliche Berechtigungen erfordern, AWS fügt sie 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 Greengras-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 den AWS IoT Greengrass-Stream-Manager hinzugefügt.
- Unterstützung für AWS IoT Greengrass in der 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.
- Es wurde Unterstützung für Linux-ARMv6l-Geräte hinzugefügt.

IDT v2.0.0 für AWS IoT Greengrass v1.9.3, v1.9.2, v.1.9.1, 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 Nachrichten nach der Diagnose zur schnelleren Behebung von Fehlern.
- 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 hinzugefügt OpenWrt.
- SSH-Benutzername und Passwort für die Geräteanmeldung hinzugefügt.

- Der native Testfehlerbehebung für die OpenWrt-ARMv7l-Plattform wurde hinzugefügt.

IDT v1.2 für AWS IoT Greengrass v1.8.1

Versionshinweise:

- Ein konfigurierbarer Timeout-Multiplikator wurde hinzugefügt, um Timeout-Probleme (z.B. bei Verbindungen mit niedriger Bandbreite) zu beheben.

IDT v1.1 für AWS IoT Greengrass v1.8.0

Versionshinweise:

- Zusätzliche Unterstützung für AWS IoT Greengrass-Hardware Security Integration (HSI) hinzugefügt.
- Zusätzliche Unterstützung für AWS IoT Greengrass-Container und keine Container hinzugefügt.
- Automatisierte AWS IoT Greengrass-Servicerollenerstellung hinzugefügt.
- Test-Ressourcenbereinigung verbessert.
- Zusammenfassungsbericht für die Testausführung hinzugefügt.

IDT v1.1 für AWS IoT Greengrass v1.7.1

Versionshinweise:

- Zusätzliche Unterstützung für AWS IoT Greengrass-Hardware Security Integration (HSI) hinzugefügt.
- Zusätzliche Unterstützung für AWS IoT Greengrass-Container und keine Container hinzugefügt.
- Automatisierte AWS IoT Greengrass-Servicerollenerstellung hinzugefügt.
- Test-Ressourcenbereinigung verbessert.
- Zusammenfassungsbericht für die Testausführung hinzugefügt.

IDT v1.0 für AWS IoT Greengrass v1.6.1

Versionshinweise:

- OTA-Testfehlerbehebung für zukünftige 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 denAWS IoT Greengrass-Qualifizierungsprogramm

Sie können Folgendes verwendenAWS IoTGerätetester (IDT) fürAWS IoT Greengrassum zu überprüfen, dass dieAWS IoT GreengrassCore-Software läuft auf Ihrer Hardware und kann mit demAWS Cloudaus. Es führt auch end-to-end testet mitAWS IoT Coreaus. So wird beispielsweise überprüft, ob Ihr Gerät MQTT-Nachrichten senden und empfangen und korrekt verarbeiten kann.

Da es sich beiAWS IoT Greengrass Version 1wurde eingezogen inWartungsmodus, IDT fürAWS IoT Greengrass V1generiert keine signierten Qualifikationsberichte mehr. Wenn Sie Ihre Hardware zumAWS PartnerGerätekatalog, führen Sie denAWS IoT Greengrass V2Qualification Suite zum Generieren von Testberichten, die Sie einreichen könnenAWS IoTaus. Weitere Informationen finden Sie unterAWS-QualifizierungsprogrammundUnterstützte Versionen von IDT fürAWS IoT Greengrass V2aus.

Neben Testgeräten bietet IDT fürAWS IoT Greengrasserstellt Ressourcen (z. B.AWS IoTthingsAWS IoT GreengrassGruppen, Lambda-Funktionen usw.) in IhremAWS-Kontoum den Qualifizierungsprozess zu erleichtern.

Um diese Ressourcen zu erstellen, verwendet IDT fürAWS IoT Greengrassverwendet dasAWSAnmeldeinformationen, die in der`config.json`um API-Aufrufe in Ihrem Auftrag zu tätigen. Diese Ressourcen werden zu verschiedenen Zeiten während eines Tests bereitgestellt.

Wenn Sie IDT für verwendenAWS IoT Greengrassum das aus:AWS IoT Greengrass-Qualifiaktionshandbuch führt IDT die folgenden Schritte aus:

1. Laden und überprüfen Sie die Konfiguration Ihres Geräts und Ihrer Anmeldeinformationen.
2. Führen Sie ausgewählte Tests mit den erforderlichen lokalen und Cloud-Ressourcen durch.

3. Bereinigen Sie lokale und Cloud-Ressourcen.
4. Erstellen Sie Testberichte, die anzeigen, ob Ihr Gerät die für die Qualifikation erforderlichen Tests bestanden hat.

Test-Suite-Versionen

IDT für 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 `list-supported-products` ausführen um die Versionen von AWS IoT Greengrass und Testsuites aufzulisten, die von Ihrer aktuellen IDT-Version 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 “Konfigurieren der IDT-Einstellungen”.

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 Ihre zu qualifizierenAWS IoT GreengrassGerät für dasAWS PartnerGeräteKatalog.

AWS IoT Greengrass-Core-Abhängigkeiten

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 dasAWS IoT GreengrassNachrichtenrouter-Funktionalität durch Überprüfung der lokalen Kommunikation zwischen dem Greengrass Core- und Client-Geräten, bei denen es sich um lokale IoT-Geräte handelt.

Over-the-Air (OTA)

Überprüft, ob Ihr Gerät erfolgreich ein OTA-Update der AWS IoT Greengrass Core-Software durchführen kann.

Diese Testgruppe ist beim Testen in einem Docker-Container nicht anwendbar.

Version

Prüft, ob die bereitgestellte AWS IoT Greengrass-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 imAWS PartnerGeräteKatalog.

Container-Abhängigkeiten

Überprüft, ob das Gerät alle Software- und Hardwareanforderungen zum Ausführen von Lambda-Funktionen im Containermodus auf einem Greengrass Core-Gerät erfüllt.

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 Core-Gerät 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 Anwendungsbereitstellungs-Konnektor für Greengrass Docker zum Ausführen 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 freigegebene HSI-Bibliothek eine Schnittstelle mit dem Hardwaresicherheitsmodul (HSM) herstellen kann und die erforderlichen PKCS# 11-APIs korrekt implementiert. 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 den AWS IoT Greengrass-Stream-Manager ausführen zu können.

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 Ihre zu qualifizierenAWS IoT GreengrassGerät für dasAWS PartnerGeräteKatalog.

AWS IoT Greengrass-Core-Abhängigkeiten

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 rotiert das AWS IoT Greengrass-Serverzertifikat und prüft, ob AWS IoT Greengrass Verbindungen zulässt.

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 diese sich kurz vor dem Ablauf befinden.

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 dasAWS IoT Greengrass-Protokollierungsservice kann mit einer in Python geschriebenen Lambda-Funktion in eine Protokolldatei schreiben.

MQTT

Überprüft die Funktionalität des AWS IoT Greengrass-Nachrichtenrouters durch Senden von Nachrichten zu einem Thema, das an zwei Lambda-Funktionen weitergeleitet wird.

Nativ

Überprüft, ob AWS IoT Greengrass native (kompilierte) Lambda-Funktionen ausführen kann.

Over-the-Air (OTA)

Überprüft, ob Ihr Gerät erfolgreich ein OTA-Update der AWS IoT Greengrass Core-Software durchführen kann.

Penetration

Überprüft, ob das Starten der AWS IoT Greengrass Core-Software fehlschlägt, wenn Hard-Link-/Soft-Link-Schutz und seccomp deaktiviert 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, obAWS IoT Greengrasskann sein Core-Zertifikat gegen gültig eintauschenAWS-Anmeldeinformationen.

Version

Prüft, ob die bereitgestellte AWS IoT Greengrass-Version 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 imAWS PartnerGeräteKatalog.

Container-Abhängigkeiten

Überprüft, ob das Gerät alle erforderlichen Abhängigkeiten erfüllt, um Lambda-Funktionen im Containermodus auszuführen.

Integration von Hardware-Sicherheit (Hardware Security Integration, HSI)

Überprüft, ob die bereitgestellte freigegebene HSI-Bibliothek eine Schnittstelle mit dem Hardwaresicherheitsmodul (HSM) herstellen kann und die erforderlichen PKCS# 11-APIs korrekt implementiert. 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 Funktion für den lokalen Ressourcenzugriff (LRA) vonAWS IoT Greengrassindem sie containerisierten Lambda-Funktionen über den Zugriff auf lokale Dateien und Verzeichnisse im Besitz von verschiedenen Linux-Benutzern und -Gruppen ermöglicht wirdAWS IoT GreengrassLRA-APIs. Lambda-Funktionen sollte Zugriff auf lokale Ressourcen basierend auf der Konfiguration des lokalen Ressourcenzugriffs erteilt 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.

Diese Kontoberechtigungen ermöglichen es IDT, in Ihrem Namen auf AWS Dienste zuzugreifen und AWS Ressourcen wie AWS IoT Dinge, Greengrass-Gruppen und Lambda-Funktionen zu erstellen.

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.

> **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 fortthe 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/signup.
2. Folgen Sie den Online-Anweisungen.

   Bei der Anmeldung müssen Sie auch einen Telefonanruf entgegennehmen und einen Verifizierungscode über die Telefontasten eingeben.

   Wenn Sie sich für eine anmelden AWS-Konto, Root-Benutzer des AWS-Kontoswird eine erstellt. Der Root-Benutzer hat Zugriff auf alle AWS -Services und Ressourcen des Kontos. Aus Sicherheitsgründen sollten Sie einem Benutzer Administratorzugriff zuweisen und nur den Root-Benutzer verwenden, um Aufgaben auszuführen, für die Root-Benutzerzugriff erforderlich ist.

AWS sendet Ihnen nach Abschluss des Anmeldevorgangs eine Bestätigungs-E-Mail. Sie können jederzeit Ihre aktuelle Kontoaktivität anzeigen und Ihr Konto verwalten. Rufen Sie dazu https://aws.amazon.com/ auf und klicken Sie auf Mein Konto.

Erstellen Sie einen Benutzer 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 Management Consoleals 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 Sie einen Benutzer 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 Benutzer in IAM Identity Center Administratorzugriff.

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

Melden Sie sich als Benutzer mit Administratorzugriff an

- 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 Zugriffsportal.

Weisen Sie weiteren Benutzern Zugriff zu

1. Erstellen Sie in IAM Identity Center einen Berechtigungssatz, der der bewährten Methode zur Anwendung von Berechtigungen mit den geringsten Rechten folgt.

Anweisungen finden Sie im Benutzerhandbuch unter Einen Berechtigungssatz erstellen.AWS IAM Identity Center

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

   Anweisungen finden Sie im AWS IAM Identity Center Benutzerhandbuch unter Gruppen hinzufügen.

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 Management Console 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 Ihres Geräts 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 Policies (Richtlinien) und dann Create policy (Richtlinie erstellen).

   b. Ersetzen Sie auf der Registerkarte JSON den Platzhalterinhalt durch die folgende Richtlinie.

```
{
    "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"
            ]
        }
    ]
}
```

> **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
> - GreenGrassota UpdateArtifactAccess
> - 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 IDTGreenGrassiamPermissions-Richtlinie, die Sie im vorherigen Schritt erstellt haben. Markieren Sie das Kontrollkästchen.

      iii. Suchen Sie nach der AWSIoTDeviceTesterForGreengrassFullAccessRichtlinie. Markieren Sie das Kontrollkästchen.

      > **Note**
      >
      > Die AWSIoTDeviceTesterForGreengrassFullAccessist 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: Markierungen.

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 Show (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 Ihres Geräts 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::*:role/idt-*",
                "arn:aws:iam::*: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 AWSIoTDeviceTesterForGreengrassFullAccessist 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 AWSIoTDeviceTesterForGreengrassFullAccessverwaltete 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 Ihres Geräts für die Ausführung von IDT-Tests

Um Ihr Gerät zu konfigurieren, müssen Sie die AWS IoT Greengrass-Abhängigkeiten installieren, die AWS IoT Greengrass Core-Software konfigurieren, Ihren Host-Computer für den Zugriff auf Ihr Gerät konfigurieren und Benutzerrechte auf Ihrem Gerät erstellen.

Überprüfen von AWS IoT Greengrass-Abhängigkeiten auf dem zu testenden Gerät

Bevor Sie Ihre Geräte mit dem IDT für AWS IoT Greengrass testen können, stellen Sie sicher, dass Sie Ihr Gerät so eingerichtet haben, wie in Erste Schritte mit AWS IoT Greengrass beschrieben. Weitere Informationen zu unterstützten Plattformen finden Sie unter Unterstützte Plattformen.

Konfigurieren der AWS IoT Greengrass-Software

IDT für AWS IoT Greengrass testet Ihr Gerät auf Kompatibilität mit einer bestimmten Version von AWS IoT Greengrass. IDT bietet zwei Optionen zum Testen von 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 bereits auf Ihrem Gerät installierten AWS IoT Greengrass Core-Software.

> **Note**
>
> Jede Version von AWS IoT Greengrass verfügt über eine entsprechende IDT-Version. Sie müssen die Version von IDT herunterladen, die der von Ihnen verwendeten AWS IoT Greengrass-Version entspricht.

In den folgenden Abschnitten werden diese Optionen beschrieben. Sie müssen nur eine der Optionen ausführen.

Option 1: Herunterladen desAWS IoT GreengrassCore-Software und konfigurierenAWS IoTGerätetester, um es zu benutzen

Sie können dieAWS IoT GreengrassCore-Software von derAWS IoT GreengrassCore-SoftwareSeite Downloads.

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 tar.gz-Datei von AWS IoT Greengrass. 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 einer vorhandenen Installation vonAWS IoT GreengrassmitAWS IoTDevice Tester

Konfigurieren Sie IDT zum Testen der auf Ihrem Gerät installierten AWS IoT Greengrass Core-Software indem Sie das Attribut `greengrassLocation` zu der Datei `device.json` im *<device-tester-extract-location>*/configs-Ordner hinzufügen. Zum Beispiel:

```
"greengrassLocation" : "<path-to-greengrass-on-device>"
```

Weitere Informationen zur Datei `device.json` finden Sie unter Konfigurieren von device.json.

Auf Linux-Geräten ist der Standardspeicherort der AWS IoT Greengrass Core-Software `/greengrass`.

> **Note**
>
> Auf Ihrem Gerät sollte eine Installation der AWS IoT Greengrass Core-Software vorhanden sein, die aber 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, 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. Zum Beispiel:

   $ ssh-copy-id ***`<remote-ssh-user>`***@***`<remote-device-ip>`***

   Dabei ist *`remote-ssh-user`* der Benutzername für die Anmeldung bei dem zu testenden Gerät und *`remote-device-ip`* ist die IP-Adresse des zu testenden Geräts für die Ausführung der Tests. Zum 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.

So erstellen Sie einen SSH-Schlüssel mit PuTTYgen (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 für den öffentlichen Schlüssel aus dem PuTTYgen-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). Zum Autorisieren dieser Operationen muss IDT für AWS IoT Greengrass Befehle mit sudo ausführen können, ohne zur Eingabe eines Passworts aufgefordert 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 “Konfigurieren der IDT-Einstellungen” fort.

Themen

- Optional: Konfigurieren des Docker-Containers für IDT fürAWS IoT Greengrass
- Optional: Konfigurieren des Geräts für die ML-Qualifizierung

Optional: Konfigurieren des Docker-Containers für IDT fürAWS IoT Greengrass

AWS IoT Greengrass stellt ein Docker-Image und eine Docker-Datei bereit, die das Ausführen der AWS IoT Greengrass Core-Software in einem Docker-Container erleichtern. 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.

Das Einrichten des Docker-Containers für die Ausführung von IDT-Tests hängt davon ab, ob Sie das Docker-Image oder die Docker-Datei verwenden, die von AWS IoT Greengrass bereitgestellt wird.

- Verwenden des Docker-Images. Das Docker-Image hat die AWS IoT Greengrass Core-Software und Abhängigkeiten installiert.
- Verwenden der Docker-Datei. Die Docker-Datei 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 Greengrassbietet keine Dockerfiles- oder Docker-Images fürAWS IoT GreengrassCore-Software, Version 1.11.1. Um IDT-Tests für eigene benutzerdefinierte Container-Images auszuführen, muss Ihr Image die Abhängigkeiten enthalten, die in der von AWS IoT Greengrass bereitgestellten Docker-Datei definiert sind .

Die folgenden Funktionen sind nicht verfügbar, wenn 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 “AWSVon bereitgestellte Greengrass-Konnektoren”. 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 des von AWS IoT Greengrass bereitgestellten Docker-Images

Führen Sie die folgenden Schritte aus, um das AWS IoT Greengrass-Docker-Image für die Ausführung von IDT-Tests zu konfigurieren.

Voraussetzungen

Führen Sie die folgenden Schritte durch, bevor Sie mit diesem Tutorial beginnen.

- Sie müssen die folgende Software und die folgenden Versionen auf Ihrem Hostcomputer basierend auf derAWS Command Line Interface(AWS CLIWählen Sie die Version aus, die Sie wählen.

  AWS CLI version 2

  - DockerVersion 18.09 oder höher. Frühere Versionen funktionieren möglicherweise auch, aber wir empfehlen 18.09 oder höher.
  - AWS CLIVersion 2.0.0 oder höher.
    - So installieren Sie dasAWS CLIVersion 2, sieheInstallieren vonAWS CLIVersion 2aus.
    - Konfigurieren derAWS CLIfinden Sie unter,Konfigurieren vonAWS CLIaus.

    > **Note**
    >
    > So aktualisieren Sie auf ein späteresAWS CLIVersion 2 auf einem Windows-Computer müssen Sie dieMSI-Installation-Prozess.

  AWS CLI version 1

  - DockerVersion 18.09 oder höher. Frühere Versionen funktionieren möglicherweise auch, aber wir empfehlen 18.09 oder höher.
  - PythonVersion 3.6 oder höher.
  - pip, Version 18.1 oder höher.
  - AWS CLIVersion 1.17.10 oder höher
    - So installieren Sie dasAWS CLIVersion 1 finden Sie unter, sieheInstallieren vonAWS CLIVersion 1aus.
    - Konfigurieren derAWS CLIfinden Sie unter,Konfigurieren vonAWS CLIaus.
    - So führen Sie ein Upgrade auf die neueste Version derAWS CLIFühren Sie den folgenden Befehl aus:

      ```
      pip install awscli --upgrade --user
      ```

    > **Note**
    >
    > Wenn Sie dasMSI-InstallationderAWS CLIVerwenden Sie unter Windows Folgendes:

- Wenn das SymbolAWS CLIVersion 1 installiert nicht, um Botocore zu installieren, versuchen Sie es mit derInstallation von Python- und Pip-Installationaus.
- So aktualisieren Sie auf ein späteresAWS CLIVersion 1 müssen Sie den MSI-Installationsvorgang wiederholen.

- Um auf Amazon Elastic Container Registry (Amazon ECR) -Ressourcen zuzugreifen, müssen Sie die folgende Berechtigung erteilen.
  - Amazon ECR verlangt von Benutzern, dass die`ecr:GetAuthorizationToken`Erlaubnis durch einAWS Identity and Access Management(IAM) -Richtlinie, bevor sie sich bei einer Registrierung authentifizieren und Images aus einem Amazon ECR-Repository pushen oder pullen können. Weitere Informationen finden Sie unterBeispiele für Amazon ECR-Repository-RichtlinienundZugriff auf ein Amazon ECR-RepositoryimAmazon Elastic Container-Registry Benutzerhandbuchaus.

1. Laden Sie das Docker-Image herunter, und konfigurieren Sie den Container. Sie können das vordefinierte Image von herunterladenDocker HuboderAmazon Elastic Container-RegistryFühren Sie es (Amazon ECR) aus und führen Sie es auf Windows-, macOS- und Linux (x86_64) -Plattformen aus.

   Führen Sie alle Schritte in aus, um das Docker-Image von Amazon ECR herunterzuladenthe section called “Abrufen desAWS IoT Greengrass Container-Image von Amazon ECR”aus. 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 verfügt, Docker-Befehle auszuführen. Weitere Informationen finden Sie unter Verwalten von Docker durch andere Benutzer als Stammbenutzer in der Docker-Dokumentation.
3. Um den AWS IoT Greengrass-Container auszuführen, verwenden Sie den Befehl für Ihr Betriebssystem:

   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>
   ```

- Ersetzen Sie *<host-path-to-kernel-config-file>* durch den Pfad zur Kernel-Konfigurationsdatei auf dem Host und *<container-path>* durch den Pfad, in dem das Volume im Container gemountet wird.

  Die Kernel-Konfigurationsdatei auf dem Host befindet sich normalerweise in `/proc/config.gz` oder `/boot/config-`*<kernel-release-date>*. Sie können `uname -r` ausführen, 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>* in dem Befehl durch den Namen des Repositorys und des Tags des Zielimages.

  Beispiel: Zum Verweis auf die neueste Version derAWS IoT GreengrassCore-Software

  ```
  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>* in dem Befehl durch den Namen des Repositorys und des Tags des Zielimages.

  Beispiel: Zum Verweis auf die neueste Version derAWS IoT GreengrassCore-Software

  ```
  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>* in dem Befehl durch den Namen des Repositorys und des Tags des Zielimages.

  Beispiel: Zum Verweis auf die neueste Version derAWS IoT GreengrassCore-Software

  ```
  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 ein, mit dem das Image zur allgemeinen AWS IoT Greengrass-Verwendung ausgeführt wird.

4. Nächster Schritt: Konfigurieren Sie IhrenAWSAnmeldedaten und`device.json`Ordneraus.

Konfigurieren der von AWS IoT Greengrass bereitgestellten Docker-Datei

Führen Sie die folgenden Schritte aus, um das aus der AWS IoT Greengrass-Docker-Datei 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 für Linux-Benutzer: Schließen Sie das abAktivieren Sie Symlink und Hardlink-SchutzundAktivieren der IPv4-NetzwerkweiterleitungSchritte.
5. Führen Sie alle Schritte in aus, um das Docker-Image zu erstellenSchritt 1. Bauen Sie dasAWS IoT GreengrassDocker-Imageaus. Kehren Sie dann zu diesem Thema zurück, um die Konfiguration fortzusetzen.
6. Um den AWS IoT Greengrass-Container auszuführen, verwenden Sie den Befehl für Ihr Betriebssystem:

   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>
   ```

   - Ersetzen Sie *<host-path-to-kernel-config-file>* durch den Pfad zur Kernel-Konfigurationsdatei auf dem Host und *<container-path>* durch den Pfad, in dem das Volume im Container gemountet wird.

Die Kernel-Konfigurationsdatei auf dem Host befindet sich normalerweise in `/proc/config.gz` oder `/boot/config-<kernel-release-date>`. Sie können `uname -r` ausführen, 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>* in dem Befehl durch den Namen des Repositorys und des Tags des Zielimages.

  Beispiel: Zum Verweis auf die neueste Version derAWS IoT GreengrassCore-Software

  ```
  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>* in dem Befehl durch den Namen des Repositorys und des Tags des Zielimages.

  Beispiel: Zum Verweis auf die neueste Version derAWS IoT GreengrassCore-Software

```
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>* in dem Befehl durch den Namen des Repositorys und des Tags des Zielimages.

  Beispiel: Zum Verweis auf die neueste Version derAWS IoT GreengrassCore-Software

  ```
  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 ein, mit dem das Image zur allgemeinen AWS IoT Greengrass-Verwendung ausgeführt wird.

7. Nächster Schritt: Konfigurieren Sie IhrenAWSAnmeldedaten und`device.json`Ordneraus.

Fehlerbehebung für das Docker-Container-Setup für IDT für AWS IoT Greengrass

Verwenden Sie die folgenden Informationen zur Behebung von Problemen beim Ausführen eines Docker-Containers zum Testen von IDT für AWS IoT Greengrass.

WARNUNG: Fehler beim Laden der Konfigurationsdatei: /home/user/config.json - stat /home/ / home/ /home/ /home///home/ /home/ <user>/home/ /home/ /home/ /home/ /home/ /home/ /home/

Wenn Sie diesen Fehler beim Ausführen von `docker`-Befehlen unter Linux erhalten, führen Sie folgenden Befehl aus. Ersetzen Sie *<user>* im folgenden Befehl durch den Benutzer, der IDT ausführt.

```
sudo chown <user>:<user> /home/<user>/.docker -R
sudo chmod g+rwx /home/<user>/.docker -R
```

Optional: Konfigurieren des Geräts für die ML-Qualifizierung

IDT für AWS IoT Greengrass stellt Machine-Learning(ML)-Qualifizierungstests bereit, mit denen Sie überprüfen können, ob Ihre Geräte ML-Inferenzen mit Cloud-geschulten Modellen lokal ausführen können.

Um ML-Qualifizierungstests ausführen zu können, müssen Sie zunächst Ihre Geräte wie unter the section called “Konfigurieren Ihres Geräts 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 mitContainerisierung(inGreengrass-Containermode), Symlinks für Python-Bibliotheken erstellen unter`/usr/local/lib/`

`python3.x`wird nicht unterstützt. 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 von MXNet-Abhängigkeiten
- the section called “Install TensorFlow Abhängigkeiten”
- Installieren von DLR-Abhängigkeiten

Installieren von Apache MXNet-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.

Installieren von MXNet

Befolgen Sie die Anweisungen in der MXNet-Dokumentation, um MXNet 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 MXNet-Installation

Wählen Sie eine der folgenden Optionen, um die MXNet-Installation zu validieren.

Option 1: Verwenden Sie 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.

Install 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.

Installieren von TensorFlow

Folgen Sie den Anweisungen im TensorFlow zu installierende Dokumentation TensorFlow 1.xmit pipoderaus der Quelleaus.

> **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.

Validierung des TensorFlow Installation

Wählen Sie eine der folgenden Optionen, um das TensorFlow installation.

Option 1: Führen Sie SSH in Ihr Gerät ein und führen Sie ein Skript aus

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 von Amazon SageMaker Abhängigkeiten von 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 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: Verwenden Sie 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 von Amazon SageMaker um das Modell zu kompilieren

Gehen Sie folgendermaßen vor, um SageMaker um das von IDT bereitgestellte ML-Modell zu kompilieren. Dieses Modell wurde mit Apache MXNet vorgeschult.

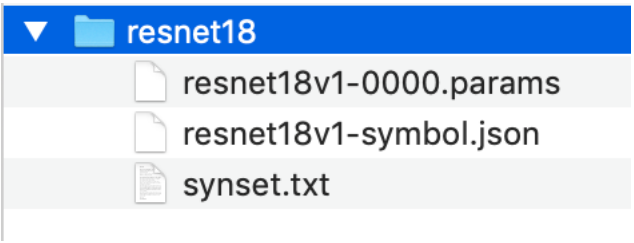
1. Vergewissern Sie sich, dass Ihr Gerät von SageMaker unterstützt wird. Weitere Informationen finden Sie im .Optionen des Zielgeräts dieAmazon SageMaker -API-Referenzaus. Wenn Ihr Gerätetyp derzeit nicht von SageMaker unterstützt wird, führen Sie die Schritte unter aus the section called “Option 2: Kompilieren des DLR-Modells mithilfe von TVM”aus.

   > **Note**
   >
   > Ausführen des DLR-Tests mit einem von kompilierten Modell SageMaker kann 4 oder 5 Minuten dauern. Halten Sie IDT während dieser Zeit nicht an.

2. Laden Sie die Tarball-Datei herunter, die das unkompilierte, vorgeschulte 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. Hochladen der komprimierten Datei in einen Amazon S3 S3-Bucket in IhremAWS-Kontound führen Sie anschließend die Schritte in aus Kompilieren eines Modells (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:

- FürS3 Output location locationgeben Sie den Pfad zum Amazon S3 S3-Bucket oder Ordner ein, in dem das kompilierte Modell gespeichert werden soll.
- 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

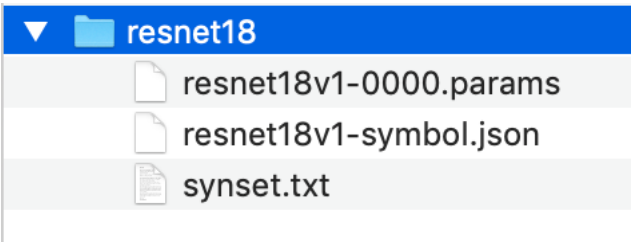
Gehen Sie folgendermaßen vor, um das von IDT bereitgestellte ML-Modell mithilfe von TVM zu kompilieren. Dieses Modell wurde mit Apache MXNet vorgeschult. Daher müssen Sie MXNet auf dem Computer oder Gerät installieren, auf dem Sie das Modell kompilieren. Befolgen Sie die Anweisungen in der MXNet-Dokumentation, um MXNet zu installieren.

> **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, vorgeschulte MXNet-Modell für DLR enthält:

   - dlr-noncompiled-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 von`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`relay_quick_start.py`file. 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 tvm.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:

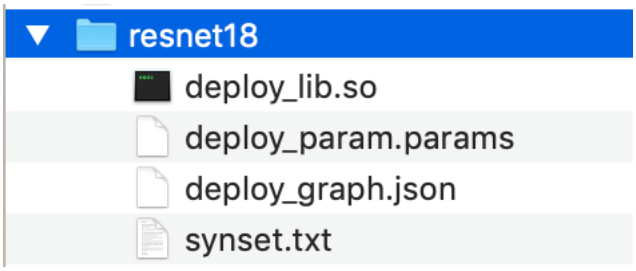
```
python3 tutorials/relay_quick_start.py --build-dir ./model
```

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.



Weiter mitKonfigurieren Sie IhrenAWSAnmeldedaten und`device.json`Ordneraus.

Konfigurieren Sie IDT-Einstellungen zum Ausführen desAWS IoT GreengrassQualifizierungs-Paket

Bevor Sie Tests ausführen, müssen Sie Einstellungen für konfigurierenAWS-Anmeldeinformationen und Geräte auf Ihrem Hostcomputer.

Konfigurieren Ihrer AWS-Anmeldeinformationen

Sie müssen Ihre IAM-Benutzeranmeldeinformationen in der*<device-tester-extract-location>* `/configs/config.json`file. Verwenden Sie die IDT-Anmeldeinformationen für AWS IoT Greengrass-Benutzer, die Sie in the section called “Erstellen und konfigurieren Sie ein AWS-Konto” erstellt haben. Sie können Ihre Anmeldeinformationen auf zwei Arten angeben:

- Anmeldeinformationen-Datei
- Umgebungsvariablen

KonfigurationAWS-Anmeldeinformationen mit einer Anmeldein

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 IhreAWS-Anmeldeinformationen für`credentials`Datei im folgenden Format:

```
[default]
aws_access_key_id = <your_access_key_id>
aws_secret_access_key = <your_secret_access_key>
```

So konfigurieren Sie IDT fürAWS IoT Greengrasszu verwendenAWS-Anmeldeinformationen von Ihrem`credentials`Datei, bearbeite deine`config.json`Datei wie folgt:

```
{
 "awsRegion": "us-west-2",
 "auth": {
  "method": "file",
  "credentials": {
   "profile": "default"
  }
 }
}
```

> **Note**
>
> Wenn Sie den`default` AWSÄndern Sie unbedingt den Profilnamen in Ihrem`config.json`file. Weitere Informationen hierzu finden Sie unter Benannte Profile.

KonfigurationAWS-Anmeldeinformationen mit Umgebungs

Umgebungsvariablen sind Variablen, die vom Betriebssystem gepflegt und von Systembefehlen verwendet werden. Sie werden nicht gespeichert, wenn Sie die SSH-Sitzung schließen. IDT fürAWS IoT GreengrassSie können die`AWS_ACCESS_KEY_ID`und`AWS_SECRET_ACCESS_KEY`-Umgebungsvariablen zum Speichern IhrerAWS-Anmeldeinformationen.

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 zuAWS-Anmeldeinformationen, IDT fürAWS IoT Greengrassbenötigt Informationen zu den Geräten, 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 Board im AWS Partner-Gerätekatalog auflisten möchten, muss die hier angegebene SKU mit der SKU übereinstimmen, die Sie während des Einreichungsprozesses verwenden.

`features`

Ein Array, das die Funktionen enthält, die das Gerät unterstützt. Alle Funktionen sind erforderlich.

`os` und `arch`

Unterstützte Kombinationen von Betriebssystemen (OS) und Architektur:

- `linux`, `x86_64`
- `linux`, `armv6l`
- `linux`, `armv7l`
- `linux`, `aarch64`
- `ubuntu`, `x86_64`
- `openwrt`, `armv7l`
- `openwrt`, `aarch64`

> **Note**
>
> Wenn Sie IDT zum Testen verwendenAWS IoT GreengrassWenn in einem Docker-Container ausgeführt wird, wird nur die x86_64 Docker-Architektur unterstützt.

`container`

Überprüft, ob das Gerät alle Software- und Hardwareanforderungen zum Ausführen von Lambda-Funktionen im Containermodus auf einem Greengrass-Kern erfüllt.

Der gültige Wert ist`yes`oder`no`aus.

`docker`

Überprüft, ob das Gerät alle erforderlichen technischen Abhängigkeiten erfüllt, um den Greengrass-Docker-Anwendungsbereitstellungs-Konnektor zum Ausführen von Containern verwenden zu können

Der gültige Wert ist`yes`oder`no`aus.

`streamManagement`

Überprüft, ob das Gerät alle erforderlichen technischen Abhängigkeiten erfüllt, um den AWS IoT Greengrass-Stream-Manager ausführen zu können.

Der gültige Wert istyesodernoaus.

`hsi`

Überprüft, ob die bereitgestellte freigegebene HSI-Bibliothek eine Schnittstelle mit dem Hardwaresicherheitsmodul (HSM) herstellen kann und die erforderlichen PKCS# 11-APIs korrekt implementiert. 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 istyesodernoaus.

`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 von`mxnet`,`tensorflow`,`dlr`, undno(z. B.`mxnet`,`mxnet,tensorflow`,`mxnet,tensorflow,dlr`, oderno) enthalten.

`mlLambdaContainerizationMode`

Überprüft, ob das Gerät alle erforderlichen technischen Abhängigkeiten erfüllt, um die ML-Inferenz im Containermodus auf einem Greengrass-Gerät durchführen zu können.

Der gültige Wert ist`container`,`process`, oder`both`aus.

`processor`

Überprüft, ob das Gerät alle Hardwareanforderungen für den angegebenen Prozessortyp erfüllt.

Der gültige Wert ist`cpu`oder`gpu`aus.

> **Note**
>
> Wenn Sie den`container`,`docker`,`streamManager`,`hsi`, oder`ml`-Funktion können Sie die entsprechende Option festlegen`value`zunoaus.
> Docker unterstützt nur Feature-Qualifikation für`streamManagement`und`ml`aus.

`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-Sicherheitsmodul (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, mit demAWS IoT GreengrassKern des Moduls.

`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-Aktualisierungsagent 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`connectivity.protocol`wird auf gesetzt`ssh`aus.

`greengrassLocation`

Der Speicherort der AWS IoT Greengrass Core-Software auf Ihren Geräten.

Dieser Wert wird für physische Geräte nur verwendet, wenn Sie eine bestehende Installation von AWS IoT Greengrass verwenden. 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 aus dem von AWS IoT Greengrass bereitgestellten Docker-Image oder Dockerfile ausführen, setzen Sie diesen Wert auf `/greengrass`.

`kernelConfigLocation`

Optional. Der Pfad zur Kernel-Konfigurationsdatei.AWS IoT Device Tester überprüft anhand dieser Datei, ob für die Geräte die erforderlichen Kernel-Funktionen aktiviert sind. Wenn nicht angeben, verwendet IDT die folgenden Pfade, um nach der Kernel-Konfigurationsdatei zu suchen:`/proc/config.gz`und`/boot/config-`*`<kernel-version>`*aus.AWS IoT Device Tester verwendet den ersten Pfad, der gefunden wird.

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. Zum Beispiel:

```
{
    "name": "ml",
    "value": "mxnet"
}
```

- Wenn Ihr Board mehrere Frameworks unterstützt, geben Sie die Frameworks als durch Kommas getrennte Liste an. Zum 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.

- Klicken Sie auf`process`um ML-Inferenzcode mit einer nicht containerisierten Lambda-Funktion auszuführen. Diese Option erfordert AWS IoT Greengrass v1.10.x oder höher.
- Klicken Sie auf`container`um ML-Inferenzcode mit einer containerisierten Lambda-Funktion auszuführen.
- Wählen Sie `both`, um ML-Inferenzcode mit beiden Modi auszuführen. Diese Option erfordert AWS IoT Greengrass v1.10.x oder höher.

`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.

`dlrModelPath`

Erforderlich, um das Framework `dlr` 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. Zum Ausführen von MXNet-Inferenztests sind auf einigen Geräten möglicherweise die folgenden Umgebungsvariablen erforderlich.

```
"environmentVariables": [
    ...
    {
        "key": "PYTHONPATH",
        "value": "$MXNET_HOME/python:$PYTHONPATH"
    },
    {
        "key": "MXNET_HOME",
        "value": "$HOME/mxnet/"
    },
    ...
]
```

> **Note**
>
> Das Feld `value` kann je nach Ihrer MXNet-Installation variieren.

Wenn Sie Lambda-Funktionen testen, die mitContainerisierungFügen Sie auf GPU-Geräten 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. EnthältLokale Ressourcenauf die 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.

Ausführen des sAWS IoT GreengrassQualifiaktionshaus

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 Standard-Update-Verhalten 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.

Auf Greengrass-Abhängigkeiten prüfen

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 Standard-Update-Verhalten 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. Folgende Werte sind zulässig:

- yaus. 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`Wenn nicht angegeben ist, verwendet IDT die neueste Version im`tests`folder.

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 AWS IoT Greengrass-Befehle

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 auf, in diesem Fall AWS IoT Greengrass-Versionen und Testsuite-Versionen für die aktuelle IDT-Version.

`list-test-cases`

Listet die Testfälle in einer bestimmten Testgruppe auf. Die folgende Option wird unterstützt:

- `group-id`aus. 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`aus. Die auszuführende Testsuite-Version. Wenn nicht angegeben, verwendet IDT die neueste Version im `tests`-Ordner.
- `group-id`aus. Die auszuführenden Testgruppen als kommagetrennte Liste. Bei fehlender Angabe führt IDT alle Testgruppen in der Testsuite aus.
- `test-id`aus. Die auszuführenden Testfälle als kommagetrennte Liste. Wenn angegeben, muss `group-id` eine einzelne Gruppe angeben.
- `pool-id`aus. Der zu testende Gerätepool. Sie müssen einen Pool angeben, wenn mehrere Gerätepools in der `device.json`-Datei definiert sind.
- `upgrade-test-suite`aus. Steuert, wie Updates der Testsuite-Version gehandhabt werden. 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`aus. Konfiguriert IDT so, dass die Ausführung beim ersten Fehler beendet 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`aus. Legt die Antwort für die Aufforderung zum Aktualisieren von IDT fest.YWenn die Eingabe beendet die Testausführung, wenn IDT erkennt, dass eine neuere Version vorhanden ist.Nwährend die Eingabe die Testausführung fortsetzt.
- `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.

Die`awsiotdevicetester_report.xml`list der Qualifikationsprüfungsbericht, an den Sie sendenAWSum Ihr Gerät imAWS PartnerGeräte-Katalog. Die Bericht enthält die folgenden Elemente:

- Die IDT-Version.
- Die AWS IoT Greengrass-Version, die getestet wurde.
- Die SKU- und der Gerätename, 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 Bericht `GGQ_Result.xml` wird im JUnit-XML-Format erstellt. Sie können ihn in Continuous Integration and Deployment-Plattformen wie Jenkins, Bamboo usw. integrieren. Die Bericht enthält die folgenden Elemente:

- Eine aggregierte Zusammenfassung der Testergebnisse.
- Eine Aufschlüsselung der Testergebnisse nach der getesteten AWS IoT Greengrass-Funktionalität.

IDT-Berichte interpretieren

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 in Tests bei den erforderlichen Funktionen keine Fehler auftreten, entspricht Ihr Gerät den technischen Anforderungen zur Ausführung von AWS IoT Greengrass und kann mit AWS IoT-Services interagieren. Wenn Sie Ihr Gerät imAWS PartnerGerätekatalog 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`

Protokolle, die aus der Komponente Test Manager des AWS IoT Device Tester generiert wurden (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 Fehlerbehebung in IDT für AWS IoT Greengrass.

Verwenden Sie IDT, um Ihre eigenen Test-Suiten zu entwickeln und zu betreiben

Ab IDT v4.0.0, IDT forAWS IoT Greengrasskombiniert ein standardisiertes Konfigurations-Setup und ein Ergebnisformat mit einer Test-Suite-Umgebung, in der Sie benutzerdefinierte Test-Suiten 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 Test-Suiten wie folgt zu entwickeln und auszuführen:

So entwickeln Sie benutzerdefinierte Test-Suiten

- Erstellen Sie Testsuiten mit benutzerdefinierter Testlogik für das Greengrass-Gerät, das Sie testen möchten.
- Stellen Sie IDT Ihre benutzerdefinierten Test-Suiten zur Verfügung, um Läufer zu testen. Fügen Sie Informationen zu bestimmten Einstellungskonfigurationen für Ihre Test-Suites hinzu.

So führen Sie benutzerdefinierte Test-Suiten aus

- Richten Sie das Gerät ein, das Sie testen möchten.
- Implementieren Sie die Einstellungskonfigurationen wie von den Test-Suiten erforderlich, die Sie verwenden möchten.
- Verwenden Sie IDT, um Ihre benutzerdefinierten Test-Suiten auszuführen.
- Zeigen Sie die Testergebnisse und Ausführungsprotokolle für die von IDT ausgeführten Tests an.

Herunterladen der neuesten Version von AWS IoT Device Tester für AWS IoT Greengrass

Herunterladen desAktuelle VersionIDT und extrahieren Sie die Software an einem Speicherort auf Ihrem Dateisystem, an dem Sie Lese- und Schreibrechte 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 Testsuite-Erstellung

Testsuiten bestehen aus drei Dateitypen:

- JSON-Konfigurationsdateien, die IDT Informationen zur Ausführung der Testsuite liefern.
- Testen Sie ausführbare Dateien, die IDT zum Ausführen 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-Konfigurationsdateienfür Ihre Testsuite.
2. Erstellen von ausführbaren Testfalldateiendie die Testlogik für Ihre Testsuite enthalten.
3. Verifizieren und dokumentieren SieKonfigurationsinformationen für Testläufer erforderlichum die Testsuite auszuführen.
4. Stellen Sie sicher, dass IDT Ihre Testsuite ausführen und produzieren kannTestergebnissewie erwartet.

Um schnell eine benutzerdefinierte Beispiel-Suite zu erstellen und auszuführen, folgen Sie den Anweisungen unterTutorial: Erstellen und führen Sie die Beispiel-IDT-Testsuite ausaus.

Informationen zum Erstellen einer benutzerdefinierten Testsuite in Python finden Sie unterTutorial: Entwickeln Sie eine einfache IDT-Testsuiteaus.

Tutorial: Erstellen und führen Sie die Beispiel-IDT-Testsuite aus

DieAWS IoTDer Download des Gerätetesters enthält den Quellcode für eine Beispiel-Testsuite. Sie können dieses Tutorial ausfüllen, um die Beispiel-Testsuite zu erstellen und auszuführen, um zu verstehen, wie Sie es verwenden könnenAWS IoTDevice Tester fürAWS IoT GreengrassSo führen Sie benutzerdefinierte Test-Suites aus.

In diesem Tutorial führen Sie die folgenden Schritte aus:

1. Bauen Sie die Beispiel-Testsuite
2. Verwenden Sie IDT, um die Beispiel-Testsuite auszuführen

Voraussetzungen

Zum Durcharbeiten dieses Tutorials ist Folgendes erforderlich:

- -Hostcomputer -Anforderungen
  - Neueste Version vonAWS IoTDevice Tester
  - Python3.7 oder höher

    Führen Sie den folgenden Befehl aus, um die Version von Python zu überprüfen, die auf dem Computer installiert ist:

    ```
    python3 --version
    ```

    Wenn die Verwendung dieses Befehls unter Windows einen Fehler zurückgibt, verwenden Sie`python --version`Stattdessen geschieht Folgendes. Wenn die zurückgegebene Versionsnummer 3.7 oder höher ist, führen Sie den folgenden Befehl in einem Powershell-Terminal aus, um festzulegen`python3`als Alias für`python`befehl.

    ```
    Set-Alias -Name "python3" -Value "python"
    ```

    Wenn keine Versionsinformationen zurückgegeben werden oder die Versionsnummer kleiner als 3.7 ist, befolgen Sie die Anweisungen unterPythonSo installieren Sie Python 3.7+. Weitere Informationen finden Sie im .Python-Dokumentationaus.

- urllib3

  So überprüfen Sie dasurllib3Führen Sie den folgenden Befehl aus:

  ```
  python3 -c 'import urllib3'
  ```

  Wennurllib3Führen 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 zum selben Netzwerk wie Ihr Host-Computer.

    Wir empfehlen Ihnen, zu verwendenRaspberry Pimit Raspberry Pi OS. Stellen Sie sicher, dass Sie einrichtenSSHSo verbinden Sie sich auf Ihrem Raspberry Pi aus der Ferne.

Geräteinformationen für IDT konfigurieren

Konfigurieren Sie Ihre Geräteinformationen, damit IDT den Test ausführen kann. Sie müssen diedevice.jsonVorlage befindet sich im*<device-tester-extract-location>*/configsOrdner mit den folgenden Informationen.

```
[
  {
    "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>"
```

```
            }
          }
        }
      }
    ]
  }
]
```

In derdevicesGeben Sie die folgenden Informationen an:

`id`

Eine benutzerdefinierte eindeutige Kennung für das 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 bei Ihrem Gerät angemeldet haben.

`connectivity.auth.credentials.privKeyPath`

Der vollständige Pfad zum privaten Schlüssel, der für die Anmeldung bei Ihrem Gerät verwendet wird.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `pki` festgelegt ist.

`devices.connectivity.auth.credentials.password`

Das Passwort für die Anmeldung bei Ihrem Gerät.

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.

Bauen Sie die Beispiel-Testsuite

Die*<device-tester-extract-location>*`/samples/python`-Ordner enthält Beispielkonfigurationsdateien, Quellcode und das IDT Client SDK, das Sie mit den 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
```

Dies erstellt die Beispiel-Testsuite im`IDTSampleSuitePython_1.0.0`Ordner innerhalb des`<device-tester-extract-location>/tests`folder. Überprüfen Sie die Dateien im`IDTSampleSuitePython_1.0.0`, um zu verstehen, wie die Beispiel-Test-Suite strukturiert ist, und verschiedene Beispiele für ausführbare Testfalldateien und Testkonfiguration JSON-Dateien zu sehen.

Nächster Schritt: Verwenden Sie IDTführen Sie die Beispiel-Testsuite ausdie du geschaffen hast.

Verwenden Sie IDT, um die Beispiel-Testsuite auszuführen

Um die Beispiel-Testsuite auszuführen, führen Sie die folgenden Befehle auf Ihrem Host-Computer aus:

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id IDTSampleSuitePython
```

IDT führt die Beispiel-Testsuite aus und streamt die Ergebnisse zur 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 Ausfüllen des Tutorials zu beheben.

Testfall wird nicht erfolgreich ausgeführt

Wenn der Test nicht erfolgreich ausgeführt wird, streamt IDT die Fehlerprotokolle an die Konsole, die Ihnen bei der Fehlerbehebung beim Testlauf helfen können. Stellen Sie sicher, dass Sie alle treffenVoraussetzungenFür dieses Tutorial.

Es kann keine Verbindung zum zu testenden Gerät hergestellt werden

Überprüfen Sie Folgendes:

- Ihre`device.json`enthält die richtige IP-Adresse, den Port und die richtigen Authentifizierungsinformationen.
- Sie können von Ihrem Host-Computer aus über SSH eine Verbindung zu Ihrem Gerät herstellen.

Tutorial: Entwickeln Sie eine einfache IDT-Testsuite

Eine Testsuite kombiniert Folgendes:

- Testen Sie ausführbare Dateien, die die Testlogik enthalten
- JSON-Konfigurationsdateien, die die Testsuite beschreiben

In diesem Tutorial erfahren Sie, wie IDT für verwendet wirdAWS IoT Greengrassum eine Python-Testsuite zu entwickeln, die einen einzigen Testfall enthält. In diesem Tutorial führen Sie die folgenden Schritte aus:

1. Erstellen Sie ein Test Suite-Verzeichnis

2. Erstellen von JSON-Konfigurationsdateien
3. Erstellen Sie die ausführbare Testfalldatei
4. Führen Sie die Testsuite aus

Voraussetzungen

Zum Durcharbeiten dieses Tutorials ist Folgendes erforderlich:

- -Host-Anforderungen
  - Neueste Version vonAWS IoTDevice Tester
  - Python3.7 oder höher

    Führen Sie den folgenden Befehl aus, um die auf dem Computer installierte Python zu überprüfen:

    ```
    python3 --version
    ```

    Wenn die Verwendung dieses Befehls unter Windows einen Fehler zurückgibt, verwenden Sie`python --version`Stattdessen geschieht Folgendes. Wenn die zurückgegebene Versionsnummer 3.7 oder höher ist, führen Sie den folgenden Befehl in einem Powershell-Terminal aus, um festzulegen`python3`als Alias für`python`befehl.

    ```
    Set-Alias -Name "python3" -Value "python"
    ```

    Wenn keine Versionsinformationen zurückgegeben werden oder die Versionsnummer kleiner als 3.7 ist, befolgen Sie die Anweisungen unterPythonSo installieren Sie Python 3.7+. Weitere Informationen finden Sie im .Python-Dokumentationaus.
  - urllib3

    So überprüfen Sie das`urllib3`Führen Sie den folgenden Befehl aus:

    ```
    python3 -c 'import urllib3'
    ```

    Wenn`urllib3`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 zum selben Netzwerk wie Ihr Host-Computer.

    Wir empfehlen Ihnen, zu verwendenRaspberry Pimit Raspberry Pi OS. Stellen Sie sicher, dass Sie die EinrichtungSSHVerwenden Sie Ihren Raspberry Pi aus, um eine Remoteverbindung zu ermöglichen.

Erstellen Sie ein Test Suite-Verzeichnis

IDT unterteilt Testfälle logisch in Testgruppen innerhalb jeder Testsuite. 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 den folgenden Verzeichnisbaum in diesem Ordner:

```
MyTestSuite_1.0.0
### suite
    ### myTestGroup
        ### myTestCase
```

Erstellen von JSON-Konfigurationsdateien

Ihre Testsuite muss Folgendes enthaltenJSON-Konfigurationsdateien:

Erforderliche JSON-Dateien

`suite.json`

Enthält Informationen zu den Testsuite. Siehe Konfigurieren Sie suite.json.

`group.json`

Enthält Informationen zu einer Testgruppe. Sie müssen eine erstellen`group.json`-Datei für jede Testgruppe in Ihrer Testsuite. Siehe Konfigurieren von group.json.

`test.json`

Enthält Informationen zu einem Testfall. Sie müssen eine erstellen`test.json`-Datei für jeden Testfall in Ihrer Testsuite. Siehe Konfigurieren Sie test.json.

1. In der`MyTestSuite_1.0.0/suite`-Ordner erstellen`suite.json`Datei mit folgender Struktur:

```
{
    "id": "MyTestSuite_1.0.0",
    "title": "My Test Suite",
    "details": "This is my test suite.",
    "userDataRequired": false
}
```

2. In der`MyTestSuite_1.0.0/myTestGroup`-Ordner erstellen`group.json`Datei mit folgender Struktur:

```
{
    "id": "MyTestGroup",
    "title": "My Test Group",
    "details": "This is my test group.",
    "optional": false
}
```

3. In der`MyTestSuite_1.0.0/myTestGroup/myTestCase`-Ordner erstellen`test.json`Datei mit folgender 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 IhreMyTestSuite_1.0.0folder 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

Nutzen Sie dieIDT-Client-SDKum IDT zu ermöglichen, mit dem zu testenden Gerät zu interagieren und Testergebnisse zu melden. Für dieses Tutorial verwenden Sie die Python-Version des SDK.

Aus*<device-tester-extract-location>*/sdks/python/, kopieren Sie denidt_clientOrdner zu IhremMyTestSuite_1.0.0/suite/myTestGroup/myTestCasefolder.

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 zu ausführende Testlogik. Eine Testsuite kann mehrere ausführbare Testfalldateien enthalten. Für dieses Tutorial erstellen Sie nur eine ausführbare Testfalldatei.

1. Erstellen Sie die Test-Suite-Datei.

   In derMyTestSuite_1.0.0/suite/myTestGroup/myTestCase-Ordner erstellenmyTestCase.pyDatei mit folgendem Inhalt:

```
from idt_client import *

def main():
    # Use the client SDK to communicate with IDT
    client = Client()

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

2. Verwenden Sie Client-SDK-Funktionen, um Ihre folgende Testlogik hinzuzufügen`myTestCase.py`file:

   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, damit IDT den Test ausführen kann. Sie müssen die`device.json`Vorlage befindet sich im*<device-tester-extract-location>*/`configs`Ordner mit den folgenden Informationen.

```
[
  {
    "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>"
            }
```

```
          }
        }
      }
    ]
  }
]
```

In der`devices`-Objekt geben Sie die folgenden Informationen an:

`id`

Eine benutzerdefinierte eindeutige Kennung für das 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 bei Ihrem Gerät angemeldet haben.

`connectivity.auth.credentials.privKeyPath`

Der vollständige Pfad zum privaten Schlüssel, der für die Anmeldung bei Ihrem Gerät verwendet wird.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `pki` festgelegt ist.

`devices.connectivity.auth.credentials.password`

Das Passwort für die Anmeldung bei Ihrem Gerät.

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 die folgenden Schritte aus, um die Testsuite mit Ihrem vorhandenen Gerätepool auszuführen.

1. Kopiere dein`MyTestSuite_1.0.0`folder in*<device-tester-extract-location>*/`testsaus`.
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 zur 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 für die Lösung von Problemen beim Ausführen des Tutorials.

Testfall wird nicht erfolgreich ausgeführt

Wenn der Test nicht erfolgreich ausgeführt wird, streamt IDT die Fehlerprotokolle an die Konsole, die Ihnen bei der Fehlerbehebung beim Testlauf helfen können. Überprüfen Sie vor dem Überprüfen der Fehlerprotokolle Folgendes:

- Das IDT-Client-SDK befindet sich im richtigen Ordner wie unterDieser Schrittaus.
- Du triffst alleVoraussetzungenfür dieses Tutorial.

Kann keine Verbindung zu dem zu testenden Gerät hergestellt werden

Überprüfen Sie Folgendes:

- Ihre`device.json`enthält die richtige IP-Adresse, den Port und die richtigen Authentifizierungsinformationen.
- Sie können von Ihrem Host-Computer aus über SSH eine Verbindung zu Ihrem Gerät herstellen.

Erstellen von IDT-Testsuite-Konfigurationsdateien

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 zur Testsuite Siehe Konfigurieren Sie suite.json.

`group.json`

Enthält Informationen zu einer Testgruppe. Sie müssen eine erstellen`group.json`-Datei für jede Testgruppe in Ihrer Testsuite. Siehe Konfigurieren von group.json.

`test.json`

Enthält Informationen zu einem Testfall. Sie müssen eine erstellen`test.json`-Datei für jeden Testfall in Ihrer Testsuite. 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`userdata.json`Ordnerdie Testläufer in ihre Einstellungskonfiguration aufnehmen können. Die`userdata.json`wird für zusätzliche Konfigurationsinformationen verwendet, die zum Ausführen des Tests erforderlich sind, aber nicht im`device.json`file. Siehe Konfigurieren userdata_schema.json.

JSON-Konfigurationsdateien werden in Ihrem*<custom-test-suite-folder>*wie hier dargestellt.

```
<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`file legt Umgebungsvariablen fest und bestimmt, ob Benutzerdaten zum Ausführen der Testsuite erforderlich sind. Konfigurieren Sie Ihre*<custom-test-suite-folder>*/`suite/suite.json`file:

```
{
    "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 die`suite.json`Die Datei befindet sich. Der Name der Suite und die Suite-Version müssen außerdem die folgenden Anforderungen erfüllen:

- *<suite-name>*darf keine Unterstriche enthalten.
- *<suite-version>*ist bezeichnet als*x.x.x*, wobeixist eine Zahl.

Die ID wird in IDT-generierten Testberichten angezeigt.

`title`

Ein benutzerdefinierter Name für das Produkt oder Feature, 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`file. Wenn Sie diesen Wert auf setzen`true`, Sie müssen auch die`userdata_schema.json`Ordnerin Ihrem Test-Suite-Ordner.

`environmentVariables`

Optional. Ein Array an Umgebungsvariablen, die für diese Testsuite festgelegt werden.

`environmentVariables.key`

Der Name der Umgebungsvariable.

`environmentVariables.value`

Der Wert der Umgebungsvariable.

Konfigurieren von group.json

Die`group.json`-Datei legt fest, ob eine Testgruppe erforderlich oder optional ist. Konfigurieren Sie Ihre`<custom-test-suite-folder>`/suite/`<test-group>`/group.jsonfile:

```
{
    "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 die`group.json`Diese Datei befindet sich 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. Legen Sie auf fest`true`um diese Testgruppe als optionale Gruppe anzuzeigen, nachdem IDT die erforderlichen Tests ausgeführt 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 Informationen zum Erstellen von ausführbaren Testfalldateien finden Sie unterErstellen Sie ausführbare IDT-Testfalldateienaus.

Konfigurieren Sie Ihre*<custom-test-suite-folder>*/suite/*<test-group>*/*<test-case>*/`test.json`file:

```
{
    "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 die`test.json`Diese Datei befindet sich und darf keine Unterstriche (_) enthalten. Die ID wird in IDT-generierten Testberichten verwendet.

`title`

Ein aussagekräftiger Name für den Testfall. Der Name wird in der IDT CLI für Testläufer angezeigt.

`details`

Eine kurze Beschreibung des Zwecks des Testfalls.

`requireDUT`

Optional. Legen Sie auf fest`true`wenn ein Gerät zum Ausführen dieses Tests benötigt wird, andernfalls auf`false`aus. Der Standardwert ist `true`. Testläufer konfigurieren die Geräte, mit denen sie den Test in ihrem`device.json`file.

`requiredResources`

Optional. Ein Array, das Informationen über Ressourcengeräte liefert, die zum Ausführen dieses Tests benötigt werden.

`requiredResources.name`

Der eindeutige Name, der dem Ressourcengerät gegeben werden soll, wenn dieser Test ausgeführt wird.

`requiredResources.features`

Ein Array benutzerdefinierter Ressourcengeräte-Features.

`requiredResources.features.name`

Der Name der Funktion. Die Gerätefunktion, für die Sie dieses Gerät verwenden möchten. Dieser Name wird mit dem Feature-Namen abgeglichen, den der Testläufer im`resource.json`file.

`requiredResources.features.version`

Optional. Die Version der Funktion. Dieser Wert wird mit der Feature-Version abgeglichen, die der Testläufer im`resource.json`file. Wenn keine Version bereitgestellt wird, wird die Funktion nicht aktiviert. Lassen Sie dieses Feld leer, wenn keine Versionsnummer für die Funktion erforderlich ist.

`requiredResources.features.jobSlots`

Optional. Die Anzahl der gleichzeitigen Tests, die diese Funktion unterstützen kann. Der Standardwert ist 1. Wenn Sie möchten, dass IDT verschiedene Geräte für einzelne Funktionen verwendet, empfehlen wir Ihnen, diesen Wert auf1aus.

`execution.timeout`

Die Zeitdauer (in Millisekunden), die IDT auf den Abschluss des Tests wartet. Weitere Informationen zum Festlegen dieses Werts finden Sie unterErstellen Sie ausführbare IDT-Testfalldateienaus.

`execution.os`

Die ausführbaren Testfalldateien, die basierend auf dem Betriebssystem des Host-Computers ausgeführt werden sollen, auf dem IDT ausgeführt wird. Unterstützte Werte sind `linux`, `mac` und `win`.

`execution.os.cmd`

Der Pfad zur 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 sollen.

`environmentVariables`

Optional. Ein Array an Umgebungsvariablen, die für diesen Testfall festgelegt wurden.

`environmentVariables.key`

Der Name der Umgebungsvariable.

`environmentVariables.value`

Der Wert der Umgebungsvariable.

> **Note**
>
> Wenn Sie dieselbe Umgebungsvariable im`test.json`-Datei und in der`suite.json`file, der Wert im`test.json`-Datei hat Vorrang.

Konfigurieren Sie state_machine.json

Ein State-Computer ist ein Konstrukt, das den Ausführungsablauf der Testsuite steuert. Es bestimmt den Startstatus einer Testsuite, verwaltet Zustandsübergänge basierend auf benutzerdefinierten Regeln und wechselt weiter durch diese Zustände, bis sie den Endzustand erreicht.

Wenn Ihre Testsuite keinen benutzerdefinierten Zustandscomputer enthält, generiert IDT einen Zustandscomputer für Sie. Der Standardzustandsmaschine führt die folgenden Funktionen aus:

- Bietet Testläufern die Möglichkeit, bestimmte Testgruppen anstelle der gesamten Testsuite auszuwählen und auszuführen.
- Wenn keine bestimmten Testgruppen ausgewählt sind, führt jede Testgruppe in der Testsuite in einer zufälligen Reihenfolge aus.
- Generiert Berichte und druckt eine Konsolenzusammenfassung, die die Testergebnisse für jede Testgruppe und jeden Testfall anzeigt.

Weitere Informationen zur Funktionsweise der IDT-Zustandsautomat finden Sie unterKonfigurieren Sie den IDT-Statuscomputeraus.

Konfigurieren userdata_schema.json

Die`userdata_schema.json`legt das Schema fest, in dem Testläufer Benutzerdaten bereitstellen. Benutzerdaten sind erforderlich, wenn Ihre Testsuite Informationen benötigt, die nicht im`device.json`file. 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 namens`userdata`, dessen Wert ein`userdata.json`Datei, die Benutzer in ihrer*`<device-tester-extract-location>`*`/config`folder. Das Format der`userdata.json`die Datei basiert auf der`userdata_schema.json`-Datei, die Sie in die Testsuite einbinden.

Um anzugeben, dass Testläufer eine`userdata.json`file:

1. In der`suite.json`-Datei, setzen`userDataRequired`zu`true`aus.
2. In Ihrer*`<custom-test-suite-folder>`*, erstelle ein`userdata_schema.json`file.
3. Bearbeiten Sie die`userdata_schema.json`Datei zum Erstellen einer gültigenIETF-Entwurf v4 JSON-Schemaaus.

Wenn IDT Ihre Testsuite ausführt, liest es das Schema automatisch und überprüft damit die`userdata.json`vom Testläufer bereitgestellte Datei. Falls gültig, ist der Inhalt des`userdata.json`die Datei ist sowohl in derIDT-Kontextund imKontext der Zustandsautomatenaus.

Konfigurieren Sie den IDT-Statuscomputer

Ein State-Computer ist ein Konstrukt, das den Ausführungsablauf der Testsuite steuert. Es bestimmt den Startstatus einer Testsuite, verwaltet Zustandsübergänge basierend auf benutzerdefinierten Regeln und wechselt weiter durch diese Zustände, bis sie den Endzustand erreicht.

Wenn Ihre Testsuite keinen benutzerdefinierten Zustandscomputer enthält, generiert IDT einen Zustandscomputer für Sie. Der Standardstatus-Computer führt die folgenden Funktionen aus:

- Bietet Testläufern die Möglichkeit, bestimmte Testgruppen anstelle der gesamten Testsuite auszuwählen und auszuführen.
- Wenn keine bestimmten Testgruppen ausgewählt sind, führt jede Testgruppe in der Testsuite in einer zufälligen Reihenfolge aus.
- Generiert Berichte und druckt eine Konsolenzusammenfassung, die die Testergebnisse für jede Testgruppe und jeden Testfall anzeigt.

Der Zustandsautomat für eine IDT-Testsuite muss die folgenden Kriterien erfüllen:

- Jeder Status entspricht einer Aktion, die IDT ausführen muss, z. B. um eine Testgruppe oder ein Produkt einer Berichtsdatei auszuführen.
- Durch den Übergang zu einem Status wird die mit dem Status verbundene Aktion ausgeführt.
- Jeder Status definiert die Übergangsregel für den nächsten Status.
- Der Endstatus muss entweder`Succeed`oder`Fail`aus.

Format des Zustandsautomaten

Sie können die folgende Vorlage verwenden, um Ihre eigene zu konfigurieren*<custom-test-suite-folder>*`/suite/state_machine.json`file:

```
{
  "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 des Zustandsautomaten.

`StartAt`

Der Name des Zustands, in dem IDT mit der Ausführung der Testsuite beginnt. Der Wert von`StartAt`Der Wert muss auf einen der im`States`-Objekt.

`States`

Ein Objekt, das benutzerdefinierte Statusnamen gültigen IDT-Status zuordnet. Jeder Bundesstaat.*`Name des Zustandsautomaten`*object enthält die Definition eines gültigen Status, der dem*`Name des Zustandsautomaten`*aus.

Die`States`Objekt muss das`Succeed`und`Fail`Status. Weitere Information zu gültigen Zuständen finden Sie unterGültige Status und Statusdefinitionenaus.

Gültige Status und Statusdefinitionen

In diesem Abschnitt werden die Statusdefinitionen aller gültigen Zustände beschrieben, die im IDT-Statuscomputer verwendet werden können. Einige der folgenden Zustände unterstützen Konfigurationen auf Testfallebene. Wir empfehlen jedoch, Statusübergangsregeln auf Testgruppenebene anstelle der Testfallebene zu konfigurieren, sofern dies nicht unbedingt erforderlich ist.

Statusdefinitionen

- RunTask

- Choice
- Parallel
- addProductFeatures
- Bericht
- LogMessage
- SelectGroup
- Fehler
- Succeed

RunTask

DieRunTaskstate führt Testfälle aus einer Testgruppe aus, die in der Testsuite definiert ist.

```
{
    "Type": "RunTask",
    "Next": "<state-name>",
    "TestGroup": "<group-id>",
    "TestCases": [
        "<test-id>"
    ],
    "ResultVar": "<result-name>"
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`Next`

Der Name des nächsten Zustands, nachdem die Aktionen im aktuellen Zustand ausgeführt werden sollen.

`TestGroup`

Optional. Die ID der auszuführenden Testgruppe. Wenn dieser Wert nicht angegeben wird, führt IDT die Testgruppe aus, die der Testläufer auswählt.

`TestCases`

Optional. Ein Array von Testfall-IDs aus der in`TestGroup`aus. Basierend auf den Werten von`TestGroup`und`TestCases`bestimmt IDT das Verhalten der Testausführung wie folgt:

- Wenn beideTestGroupundTestCasesangegeben sind, führt IDT die angegebenen Testfälle aus der Testgruppe aus.
- WannTestCasessind angegeben aberTestGroupist nicht angegeben, IDT führt die angegebenen Testfälle aus.
- WannTestGroupist angegeben, aberTestCasesWenn nicht angegeben, führt IDT alle Testfälle innerhalb der angegebenen Testgruppe aus.
- Wenn keinesTestGroupoderTestCasesangegeben 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 beide einschließenRunTaskundChoiceBundesstaaten in deinemstatemachine.jsonfile. Ein Beispiel dafür, wie dies funktioniert, finden Sie unterBeispiel für einen Zustandsautomaten: Ausführen von vom Benutzer ausgewählten Testgruppenaus.

  Weitere Informationen zur Aktivierung von IDT CLI-Befehlen für Testläufer finden Sie unterthe section called “Aktivieren von IDT-CLI-Befehlen”aus.

`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 habenTestGroupaus. IDT legt den Wert der Variablen fest, in der Sie definierenResultVarzutrueoderfalsebasierend auf den folgenden Kriterien:

- Wenn der Variablenname vom Formular stammt*text*_*text*_passed, dann wird der Wert darauf eingestellt, ob alle Tests in der ersten Testgruppe bestanden oder übersprungen wurden.
- In allen anderen Fällen wird der Wert darauf festgelegt, ob alle Tests in allen Testgruppen bestanden oder übersprungen wurden.

In Regel verwenden SieRunTaskstatus, um eine Testgruppen-ID anzugeben, ohne einzelne Testfall-IDs anzugeben, damit IDT alle Testfälle in der angegebenen Testgruppe ausführt. Alle Testfälle, die von diesem Status ausgeführt werden, laufen parallel in einer zufälligen Reihenfolge. Wenn jedoch für alle Testfälle ein Gerät ausgeführt werden muss und nur ein einziges Gerät verfügbar ist, werden die Testfälle stattdessen nacheinander ausgeführt.

Fehlerbehandlung

Wenn eine der angegebenen Testgruppen oder Testfall-IDs nicht gültig ist, gibt dieser Status dieRunTaskErrorFehler bei der-Ausführung. Wenn der Status auf einen Ausführungsfehler stößt, wird auch diehasExecutionErrorVariable im Zustandsmaschine-Kontext zutrueaus.

Choice

Die`Choice`state ermöglicht es Ihnen, den nächsten Status dynamisch festzulegen, auf den Sie basierend auf benutzerdefinierten Bedingungen wechseln möchten.

```
{
    "Type": "Choice",
    "Default": "<state-name>",
    "FallthroughOnError": true | false,
    "Choices": [
        {
            "Expression": "<expression>",
            "Next": "<state-name>"
        }
    ]
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`Default`

Der Standardzustand, wenn keiner der in`Choices`kann ausgewertet werden`true`aus.

`FallthroughOnError`

Optional. Gibt das Verhalten an, wenn der Status bei der Auswertung von Ausdrücken auf einen Fehler stößt. Auf einstellen`true`wenn Sie einen Ausdruck überspringen möchten, wenn die Auswertung zu einem Fehler führt. Wenn keine Ausdrücke übereinstimmen, wechselt der Zustandsautomat zum`Default`Status. Wenn das Symbol`FallthroughOnError`Wert ist nicht angegeben, er ist standardmäßig`false`aus.

`Choices`

Ein Array von Ausdrücken und Zuständen, in den festgelegt werden soll, in welchen Status nach Ausführung der Aktionen im aktuellen Status umgestellt werden soll.

`Choices.Expression`

Eine Ausdruckszeichenfolge, die zu einem booleschen Wert ausgewertet wird. Wenn der Ausdruck ausgewertet wird`true`, dann wechselt der Zustandsautomat in den in`Choices.Next`aus. Ausdruckszeichenfolgen rufen Werte aus dem Zustandsmaschinenkontext ab und führen dann Operationen an ihnen aus, um zu einem

booleschen Wert zu gelangen. Informationen über den Zugriff auf den Zustandsmaschinen-Kontext finden Sie unterKontext von Zustandsautomatenaus.

`Choices.Next`

Der Name des nächsten Zustands, wenn der in`Choices.Expression`wertet nach aus`true`aus.

Fehlerbehandlung

Die`Choice`state kann in folgenden Fällen eine Fehlerbehandlung erfordern:

- Einige Variablen in den Auswahlausdrücken existieren im Zustandsmaschinenkontext nicht.
- Das Ergebnis eines Ausdrucks ist kein boolescher Wert.
- Das Ergebnis eines JSON-Lookups ist kein String, keine Zahl oder ein boolescher Wert.

Sie können kein`Catch`Block, um Fehler in diesem Zustand zu behandeln.
Wenn Sie die Ausführung des Statusrechners beenden möchten, wenn ein
Fehler auftritt, müssen Sie`FallthroughOnError`zu`false`aus. Wir empfehlen jedoch,`FallthroughOnError`zu`true`Abhängig von Ihrem Anwendungsfall führen Sie einen der folgenden Schritte aus:

- Wenn erwartet wird, dass eine Variable, auf die Sie zugreifen, in einigen Fällen nicht existiert, verwenden Sie den Wert von`Default`und zusätzlich`Choices`Blöcke, um den nächsten Zustand anzugeben.
- Wenn eine Variable, auf die Sie zugreifen, immer vorhanden sein sollte, setzen Sie die`Default`Status zu`Fail`aus.

Parallel

Die`Parallel`state ermöglicht es Ihnen, neue Zustandsmaschinen parallel zueinander zu definieren und auszuführen.

```
{
    "Type": "Parallel",
    "Next": "<state-name>",
    "Branches": [
        <state-machine-definition>
    ]
```

```
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`Next`

Der Name des nächsten Zustands, nachdem die Aktionen im aktuellen Zustand ausgeführt werden sollen.

`Branches`

Ein Array von auszuführenden Statusmaschinendefinitionen. Jede Definition des Zustandsautomaten muss ihre eigene enthalten`StartAt`,`Succeed`, und`Fail`Status. Die Statusmaschinendefinitionen in diesem Array können keine Zustände außerhalb ihrer eigenen Definition referenzieren.

> **Note**
>
> Da jeder Zweigstatus-Computer denselben Zustands-Maschinen-Kontext hat, kann das Festlegen von Variablen in einem Zweig und das anschließende Lesen dieser Variablen aus einem anderen Zweig zu unerwartetem Verhalten führen.

Die`Parallel`state wechselt erst in den nächsten Status, nachdem alle Zweigzustandsmaschinen ausgeführt wurden. Jeder Status, der ein Gerät benötigt, wartet auf die 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 Testfälle nacheinander ausgeführt. Da Testfälle in einer zufälligen Reihenfolge ausgeführt werden, wenn sie parallel ausgeführt werden, können verschiedene Geräte verwendet werden, um Tests aus derselben Testgruppe auszuführen.

Fehlerbehandlung

Stellen Sie sicher, dass sowohl der Zweigzustandsmaschine als auch der übergeordnete Zustandsmaschine in die`Fail`status, um Ausführungsfehler zu behandeln.

Da Zweigzustandsmaschinen keine Ausführungsfehler an den übergeordneten Zustandscomputer übertragen, können Sie keine`Catch`blockieren, um Ausführungsfehler in Zweigzustandsmaschinen zu behandeln. Verwenden Sie stattdessen den`hasExecutionErrors`Wert im Kontext des freigegebenen Zustands-Rechners. Ein Beispiel dafür, wie dies funktioniert, finden Sie unter[Beispiel für einen Zustandsautomaten: Führen Sie zwei Testgruppen parallel aus](#)aus.

addProductFeatures

Die`AddProductFeatures`state ermöglicht es Ihnen, Produktfunktionen zum`awsiotdevicetester_report.xml`von IDT generierte Datei.

Ein Produktmerkmal sind benutzerdefinierte Informationen zu bestimmten Kriterien, die ein Gerät möglicherweise erfüllen kann. Zum Beispiel, das`MQTT`Die Produktfunktion kann angeben, dass das Gerät MQTT-Nachrichten ordnungsgemäß veröffentlicht. Im Bericht werden Produkteigenschaften als festgelegt`supported`,`not-supported`oder ein benutzerdefinierter Wert, basierend darauf, ob bestimmte Tests bestanden wurden.

> **Note**
>
> Die`AddProductFeatures`state generiert keine Berichte von selbst. Dieser Staat muss auf die`Report`Bundesstaatum Berichte zu generieren.

```
{
    "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 nächsten Zustands, nachdem die Aktionen im aktuellen Zustand ausgeführt werden sollen.

`Features`

Eine Reihe von Produkteigenschaften, die in der`awsiotdevicetester_report.xml`file.

`Feature`

Der Name der Funktion

`FeatureValue`

Optional. Der benutzerdefinierte Wert, der im Bericht anstelle von verwendet werden soll`supported`aus. Wenn dieser Wert nicht angegeben wird, wird der Feature-Wert basierend auf Testergebnissen auf`supported`oder`not-supported`aus.

Wenn Sie einen benutzerdefinierten Wert für verwenden`FeatureValue`können Sie dasselbe Feature mit verschiedenen Bedingungen testen, und IDT verkettet die Feature-Werte für die unterstützten Bedingungen. Der folgende Auszug zeigt beispielsweise`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 bestehen, wird der Feature-Wert auf`first-feature-supported, second-feature-supported`aus.

`Groups`

Optional. Ein Array von Testgruppen-IDs. Alle Tests innerhalb jeder angegebenen Testgruppe müssen bestehen, damit das Feature unterstützt wird.

`OneOfGroups`

Optional. Ein Array von Testgruppen-IDs. Alle Tests innerhalb mindestens einer der angegebenen Testgruppen müssen bestehen, damit das Feature unterstützt wird.

`TestCases`

Optional. Ein Array von Testfall-IDs. Wenn Sie diesen Wert angeben, gilt Folgendes:

- Alle angegebenen Testfälle müssen bestehen, damit das Feature unterstützt wird.
- `Groups`darf nur eine Testgruppen-ID enthalten.
- `OneOfGroups`darf nicht angegeben werden.

`IsRequired`

Optional. Auf einstellen`false`um diese Funktion als optionales Feature im Bericht zu markieren. Der Standardwert ist `true`.

`ExecutionMethods`

Optional. Ein Array von Ausführungsmethoden, die mit`protocol`Der Wert ist im`device.json`file. Wenn dieser Wert angegeben wird, müssen Testläufer ein`protocol`Wert, der mit einem der Werte in diesem Array übereinstimmt, um das Feature in den Bericht aufzunehmen. Wenn dieser Wert nicht angegeben wird, wird das Feature immer in den Bericht aufgenommen.

So verwenden Sie den`AddProductFeatures`state, müssen Sie den Wert von`ResultVar`im`RunTask`Geben Sie einen der folgenden Werte an:

- Wenn Sie einzelne Testfall-IDs angegeben haben, setzen Sie`ResultVar`zu*`group-id_test-id`*`_passed`aus.
- Wenn Sie keine einzelnen Testfall-IDs angegeben haben, setzen Sie`ResultVar`zu*`group-id`*`_passed`aus.

Die`AddProductFeatures`Zustandsprüfungen der Testergebnisse in der folgenden Weise:

- Wenn Sie keine Testfall-IDs angegeben haben, wird das Ergebnis für jede Testgruppe aus dem Wert des*`group-id`*`_passed`variabel im Zustandsmaschinen-Kontext.
- Wenn Sie Testfall-IDs angegeben haben, wird das Ergebnis für jeden der Tests aus dem Wert des*`group-id_test-id`*`_passed`variabel im Zustandsmaschinen-Kontext.

Fehlerbehandlung

Wenn eine in diesem Status angegebene Gruppen-ID keine gültige Gruppen-ID ist, führt dieser Status zu`AddProductFeaturesError`Fehler bei der-Ausführung. Wenn der Status auf einen Ausführungsfehler stößt, wird auch die`hasExecutionErrors`Variable im Zustandsmaschine-Kontext zu`true`aus.

Bericht

Die`Report`Der Zustand generiert den*`suite-name`*`_Report.xml`und`awsiotdevicetester_report.xml`Dateien. Dieser Status streamt den Bericht auch an die Konsole.

```
{
    "Type": "Report",
    "Next": "<state-name>"
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`Next`

Der Name des nächsten Zustands, nachdem die Aktionen im aktuellen Zustand ausgeführt werden sollen.

Du solltest immer auf die`Report`gegen Ende des Testausführungsflusses angeben, damit Testläufer Testergebnisse anzeigen können. In der Regel ist der nächste Status nach diesem Status`Succeed`aus.

Fehlerbehandlung

Wenn dieser Status Probleme beim Erstellen der Berichte aufweist, gibt er die`ReportError`Fehler bei der-Ausführung.

LogMessage

Die`LogMessage`Der Zustand generiert den`test_manager.log`-Datei und streamt die Protokollnachricht zur Konsole.

```
{
```

```
    "Type": "LogMessage",
    "Next": "<state-name>"
    "Level": "info | warn | error"
    "Message": "<message>"
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`Next`

Der Name des nächsten Zustands, nachdem die Aktionen im aktuellen Zustand ausgeführt werden sollen.

`Level`

Die Fehlerstufe, auf der die Protokollmeldung erstellt werden soll. Wenn Sie eine Ebene angeben, die nicht gültig ist, generiert dieser Status eine Fehlermeldung und verwirft diese.

`Message`

Die zu protokolligende Nachricht.

SelectGroup

Die`SelectGroup`status aktualisiert den Zustandsmaschinenkontext, um anzugeben, welche Gruppen ausgewählt sind. Die von diesem Status festgelegten Werte werden von allen nachfolgenden verwendet`Choice`Status.

```
{
    "Type": "SelectGroup",
    "Next": "<state-name>"
    "TestGroups": [
        <group-id>"
    ]
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`Next`

Der Name des nächsten Zustands, nachdem die Aktionen im aktuellen Zustand ausgeführt werden sollen.

`TestGroups`

Ein Array von Testgruppen, die als ausgewählt gekennzeichnet werden. Für jede Testgruppen-ID in diesem Array wird die*`group-id`*`_selected`Variable ist auf`true`im Kontext. Stellen Sie sicher, dass Sie gültige Testgruppen-IDs angeben, da IDT nicht überprüft, ob die angegebenen Gruppen vorhanden sind.

Fehler

Die`Fail`status gibt an, dass der Zustandsmaschine nicht korrekt ausgeführt wurde. Dies ist ein Endstatus für den Statuscomputer, und jede Statusmaschinendefinition muss diesen Status enthalten.

```
{
    "Type": "Fail"
}
```

Succeed

Die`Succeed`status gibt an, dass der Zustandsmaschine korrekt ausgeführt wurde. Dies ist ein Endstatus für den Statuscomputer, und jede Statusmaschinendefinition muss diesen Status enthalten.

```
{
    "Type": "Succeed"
}
```

Kontext von Zustandsautomaten

Der Zustandsmaschinenkontext ist ein schreibgeschütztes JSON-Dokument, das Daten enthält, die dem Statuscomputer während der Ausführung zur Verfügung stehen. Der Zustandsmaschinenkontext ist nur vom Zustandsmaschine aus zugänglich und enthält Informationen, die den Testfluss bestimmen. Beispielsweise können Sie Informationen verwenden, die von Testläufern konfiguriert wurden, im`userdata.json`-Datei, um festzustellen, ob ein bestimmter Test ausgeführt werden muss.

Der Kontext des Zustandsautomaten 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`file.

`userData`

-Informationen im`userdata.json`file.

`config`

-Informationen pinnen Sie den`config.json`file.

`suiteFailed`

Der Wert ist auf`false`wenn der Zustandsautomat gestartet wird. Wenn eine Testgruppe in einer`RunTask`state, dann ist dieser Wert auf`true`Für die verbleibende Dauer der Ausführung des Zustandsautomaten.

`specificTestGroups`

Wenn der Testrunner bestimmte Testgruppen auswählt, die anstelle der gesamten Testsuite ausgeführt werden sollen, wird dieser Schlüssel erstellt und enthält die Liste der spezifischen Testgruppen-IDs.

`specificTestCases`

Wenn der Testläufer bestimmte Testfälle auswählt, die anstelle der gesamten Testsuite ausgeführt werden sollen, wird dieser Schlüssel erstellt und enthält die Liste der spezifischen Testfall-IDs.

`hasExecutionErrors`

Wird nicht beendet, wenn der Zustandsmaschine gestartet wird. Wenn ein Status auf Ausführungsfehler stößt, wird diese Variable erstellt und auf`true`Für die verbleibende Dauer der Ausführung des Zustandsautomaten.

Sie können den Kontext mit der JsonPath-Notation abfragen. Die Syntax für JsonPath-Abfragen in Statusdefinitionen lautet`{{$.query}}`aus. Sie können JsonPath-Abfragen als Platzhalterzeichenfolgen in einigen Zuständen verwenden. IDT ersetzt die Platzhalterzeichenfolgen durch den Wert der ausgewerteten JsonPath-Abfrage aus dem Kontext. Sie können Platzhalter für folgende Werte verwenden:

- Die`TestCases`Wert in`RunTask`Status.
- Die`Expression`Wert`Choice`Status.

Wenn Sie auf Daten aus dem Zustandsautomatenkontext zugreifen, stellen Sie sicher, dass die folgenden Bedingungen erfüllt sind:

- Ihre JSON-Pfade müssen mit beginnen`$.`
- Jeder Wert muss zu einer Zeichenfolge, einer Zahl oder einem booleschen Wert ausgewertet werden.

Weitere Informationen zur Verwendung von JSONPath-Notation für den Zugriff auf Daten aus dem Kontext finden Sie unterVerwenden Sie den IDT-Kontextaus.

Ausführungsfehler

Ausführungsfehler sind Fehler in der Statusmaschinendefinition, auf die der Statuscomputer beim Ausführen eines Status stößt. IDT protokolliert Informationen zu jedem Fehler im`test_manager.log`-Datei und streamt die Protokollnachricht zur Konsole.

Sie können die folgenden Methoden verwenden, um Ausführungsfehler zu behandeln:

- Hinzufügen einer`CatchBlock`in der Statusdefinition.
- Überprüfen Sie den Wert des`hasExecutionErrorsWert`im Zustandsautomatenkontext.

Fangen

Um zu verwenden`Catch`, fügen Sie Ihrer Statusdefinition Folgendes hinzu:

```
"Catch": [
    {
        "ErrorEquals": [
            "<error-type>"
        ]
        "Next": "<state-name>"
    }
]
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`Catch.ErrorEquals`

Ein Array der abzusehenden Fehlertypen. Wenn ein Ausführungsfehler mit einem der angegebenen Werte übereinstimmt, wechselt der Zustandsautomat in den in`Catch.Next`aus. Informationen über die Art des Fehlers, den es erzeugt, finden Sie in jeder Statusdefinition.

`Catch.Next`

Der nächste Status, zu dem umgestellt werden soll, wenn der aktuelle Status auf einen Ausführungsfehler stößt, der mit einem der in angegebenen Werte übereinstimmt`Catch.ErrorEquals`aus.

Catch-Blöcke werden nacheinander gehandhabt, bis einer übereinstimmt. Wenn die Keine Fehler mit den in den Catch-Blöcken aufgeführten übereinstimmen, werden die Zustandsmaschinen weiterhin ausgeführt. Da Ausführungsfehler auf falsche Zustandsdefinitionen zurückzuführen sind, empfehlen wir Ihnen, in den Status „Fail“ zu wechseln, wenn ein Status auf einen Ausführungsfehler stößt.

hasExecutionError

Wenn einige Zustände auf Ausführungsfehler stoßen, legen sie zusätzlich zur Fehlerausgabe auch die`hasExecutionError`Wert auf`true`Im Kontext des Zustandsautomaten. Sie können diesen

Wert verwenden, um zu erkennen, wann ein Fehler auftritt, und dann eine`Choice`status, um die Zustandsmaschine auf den`Fail`Status.

Diese Methode hat folgende Merkmale.

- Der Zustandsmaschine beginnt nicht mit einem Wert, der dem zugewiesen wurde`hasExecutionError`, und dieser Wert ist erst verfügbar, wenn ein bestimmter Status ihn festgelegt hat. Dies bedeutet, dass Sie explizit den`FallthroughOnError`zu`false`für`Choice`gibt an, dass auf diesen Wert zugegriffen wird, um zu verhindern, dass der Statuscomputer beendet wird, wenn keine Ausführungsfehler auftreten
- Sobald es auf eingestellt ist`true`,`hasExecutionError`wird niemals auf false festgelegt oder aus dem Kontext entfernt. Dies bedeutet, dass dieser Wert nur nützlich ist, wenn er zum ersten Mal auf`true`und für alle nachfolgenden Staaten bietet es keinen aussagekräftigen Wert.
- Die`hasExecutionError`Wert wird mit allen Zweigzustandsmaschinen im`Parallel`status, was abhängig von der Reihenfolge, in der darauf zugegriffen wird, zu unerwarteten Ergebnissen führen kann.

Aufgrund dieser Eigenschaften empfehlen wir, diese Methode nicht zu verwenden, wenn Sie stattdessen einen Catch-Block verwenden können.

Beispiel für einen Zustandsautomaten

In diesem Abschnitt finden Sie einige Beispiele für Zustandsautomaten-Konfigurationen.

Beispiele

- Beispiel für einen Zustandsautomaten: Führen Sie eine einzelne Testgruppe aus
- Beispiel für einen Zustandsautomaten: Ausführen von vom Benutzer ausgewählten Testgruppen
- Beispiel für einen Zustandsautomaten: Führen Sie eine einzelne Testgruppe mit Produktfunktionen aus
- Beispiel für einen Zustandsautomaten: Führen Sie zwei Testgruppen parallel aus

Beispiel für einen Zustandsautomaten: Führen Sie eine einzelne Testgruppe aus

Dieser Zustandsautomat:

- Führt die Testgruppe mit ID aus`GroupA`, die in der Suite in einem`group.json`file.

- Prüft auf Ausführungsfehler und Übergänge zu`Fail`falls welche gefunden werden.
- Generiert einen Bericht und wechselt zu`Succeed`wenn es keine Fehler gibt, und`Fail`Ansonsten .

```
{
    "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 einen Zustandsautomaten: Ausführen von vom Benutzer ausgewählten Testgruppen

Dieser Zustandsautomat:

- Prüft, ob der Testläufer bestimmte Testgruppen ausgewählt hat. Der Zustandsmaschine prüft nicht nach bestimmten Testfällen, da Testläufer Testfälle nicht auswählen können, ohne auch eine Testgruppe auszuwählen.
- Wenn Testgruppen ausgewählt sind:
  - Führt die Testfälle innerhalb der ausgewählten Testgruppen aus. Zu diesem Zweck gibt der Zustandsmaschine keine Testgruppen oder Testfälle in der`RunTask`Status.
  - Generiert einen Bericht, nachdem alle Tests und Exits ausgeführt wurden.
- Wenn Testgruppen nicht ausgewählt sind:
  - Führt Tests in Testgruppe aus`GroupA`aus.
  - Generiert Berichte und Exits.

```
{
    "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 einen Zustandsautomaten: Führen Sie eine einzelne Testgruppe mit Produktfunktionen aus

Dieser Zustandsautomat:

- Führt die Testgruppe aus`GroupA`aus.
- Prüft auf Ausführungsfehler und Übergänge zu`Fail`falls welche gefunden werden.
- Fügt den`FeatureThatDependsOnGroupA`Funktion zum`awsiotdevicetester_report.xml`file:
  - Wenn`GroupA`Pässe, das Feature ist auf`supported`aus.
  - Die Funktion ist im Bericht nicht als optional gekennzeichnet.
- Generiert einen Bericht und wechselt zu`Succeed`wenn es keine Fehler gibt, und`Fail`ansonsten

```
{
    "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 für einen Zustandsautomaten: Führen Sie zwei Testgruppen parallel aus

Dieser Zustandsautomat:

- Führt den`GroupA`und`GroupB`Testgruppen parallel. Die`ResultVar`Variablen, die im Kontext von der`RunTask`Zustände in den Zweigzustandsmaschinen von sind für die`AddProductFeatures`Status.
- Prüft auf Ausführungsfehler und Übergänge zu`Fail`falls welche gefunden werden. Dieser Zustandsmaschine verwendet kein`Catch`blockieren, weil diese Methode keine Ausführungsfehler in Zweigzustandsmaschinen erkennt.
- Fügt Funktionen zum`awsiotdevicetester_report.xml`Datei basierend auf den Gruppen, die bestehen
  - Wenn`GroupA`Pässe, das Feature ist auf`supported`aus.
  - Die Funktion ist im Bericht nicht als optional gekennzeichnet.
- Generiert einen Bericht und wechselt zu`Succeed`wenn es keine Fehler gibt, und`Fail`ansonsten

Wenn zwei Geräte im Gerätepool konfiguriert sind, sind beide`GroupA`und`GroupB`kann gleichzeitig ausgeführt werden. Wenn entweder entweder`GroupA`oder`GroupB`hat mehrere Tests, dann können

beide Geräte diesen Tests zugeordnet 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"
        }
    }
}
```

Erstellen Sie ausführbare IDT-Testfalldateien

Sie können ausführbare Testfalldateien auf folgende Weise in einem Test-Suite-Ordner erstellen und ablegen:

- Für Test-Suites, die Argumente oder Umgebungsvariablen aus dem`test.json`-Dateien, um festzustellen, welche Tests ausgeführt werden sollen, können Sie eine einzelne ausführbare Testfalldatei für die gesamte Testsuite oder eine ausführbare Testdatei für jede Testgruppe in der Testsuite erstellen.
- Für eine Testsuite, in der Sie bestimmte Tests basierend auf bestimmten Befehlen ausführen möchten, erstellen Sie für jeden Testfall in der Testsuite eine ausführbare Testfalldatei.

Als Testschreiber können Sie bestimmen, welcher Ansatz für Ihren Anwendungsfall geeignet ist, und Ihre ausführbare Testfalldatei entsprechend strukturieren. Stellen Sie sicher, dass Sie in jedem Fall den richtigen ausführbaren Testfallpfad angeben`test.json`-Datei, und dass die angegebene ausführbare Datei korrekt ausgeführt wird.

Wenn alle Geräte bereit sind, einen Testfall auszuführen, liest IDT die folgenden Dateien:

- Die`test.json`bestimmt für den ausgewählten Testfall die zu startenden Prozesse und die festzulegenden Umgebungsvariablen.
- Die`suite.json`für die Testsuite bestimmt die festzulegenden Umgebungsvariablen.

IDT startet den erforderlichen ausführbaren Testprozess basierend auf den Befehlen und Argumenten, die in der`test.json`und übergibt die erforderlichen Umgebungsvariablen an den Prozess.

Verwenden Sie das IDT Client SDK

Mit den IDT Client SDKs können Sie vereinfachen, wie Sie Testlogik in Ihre ausführbare Testdatei schreiben, mit API-Befehlen, die Sie mit IDT und Ihren zu testenden Geräten interagieren können. IDT bietet derzeit die folgenden SDKs:

- IDT-Client-SDK für Python
- IDT-Client-SDK SDK for Go

Diese SDKs befinden sich im`<device-tester-extract-location>`/`sdks`folder. Wenn Sie eine neue ausführbare Testfalldatei erstellen, müssen Sie das zu verwendende SDK in den Ordner kopieren, der Ihre ausführbare Testfalldatei enthält, und auf das SDK in Ihrem Code verweisen. In diesem Abschnitt finden Sie eine kurze Beschreibung der verfügbaren API-Befehle, die Sie in Ihren ausführbaren Testfällen verwenden können.

In diesem Abschnitt

- Interaktion mit Geräten
- IDT-Interaktion
- Host-Interaktion

Interaktion mit Geräten

Die folgenden Befehle ermöglichen es Ihnen, mit dem zu testenden Gerät zu kommunizieren, ohne zusätzliche Funktionen zur Geräteinteraktion und Konnektivitätsverwaltung implementieren zu müssen.

`ExecuteOnDevice`

Ermöglicht Test-Suiten das Ausführen von Shell-Befehlen auf einem Gerät, das SSH- oder Docker-Shell-Verbindungen unterstützt.

`CopyToDevice`

Ermöglicht Test-Suiten das Kopieren einer lokalen Datei von dem Host-Computer, der IDT ausführt, an einen bestimmten Speicherort auf einem Gerät, das SSH- oder Docker-Shell-Verbindungen unterstützt.

`ReadFromDevice`

Ermöglicht es Test-Suiten, 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 die Geräteinteraktion in Ihren ausführbaren Testfalldateien zu verwenden. Wenn diese Befehle jedoch nicht Ihren Testfallanforderungen entsprechen, können Sie Informationen zum Gerätezugriff aus dem IDT-Kontext abrufen und damit aus der Testsuite eine direkte Verbindung zum Gerät herstellen.
> Um eine direkte Verbindung herzustellen, rufen Sie die Informationen im`device.connectivity`und die`resource.devices.connectivity`-Felder für Ihr zu testendes Gerät bzw. für Ressourcengeräte. Weitere Informationen zur Verwendung des IDT-Kontextes finden Sie unterVerwenden Sie den IDT-Kontextaus.

IDT-Interaktion

Die folgenden Befehle ermöglichen es Ihren Test-Suiten, mit IDT zu kommunizieren.

`PollForNotifications`

Ermöglicht Test-Suiten, nach Benachrichtigungen von IDT zu suchen.

`GetContextValue` und `GetContextString`

Ermöglicht Test-Suiten das Abrufen von Werten aus dem IDT-Kontext. Weitere Informationen finden Sie unter Verwenden Sie den IDT-Kontext.

`SendResult`

Ermöglicht Test-Suiten, Testfallergebnisse an IDT zu melden. Dieser Befehl muss am Ende jedes Testfalls in einer Testsuite aufgerufen werden.

Host-Interaktion

Mit dem folgenden Befehl können Ihre Test-Suites mit dem Host-Computer kommunizieren.

`PollForNotifications`

Ermöglicht Test-Suiten, nach Benachrichtigungen von IDT zu suchen.

`GetContextValue` und `GetContextString`

Ermöglicht Test-Suiten das Abrufen von Werten aus dem IDT-Kontext. Weitere Informationen finden Sie unter Verwenden Sie den IDT-Kontext.

`ExecuteOnHost`

Ermöglicht Test-Suites das Ausführen von Befehlen auf dem lokalen Computer und lässt IDT den Lebenszyklus der ausführbaren Testfalldatei verwalten.

Aktivieren von IDT-CLI-Befehlen

Die`run-suite`command IDT CLI bietet mehrere Optionen, mit denen Testläufer die Testausführung anpassen können. 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, einige CLI-Optionen funktionieren jedoch nicht ordnungsgemäß. Um ein ideales Kundenerlebnis zu bieten, empfehlen wir Ihnen, Unterstützung für die folgenden Argumente für die`run-suite`Befehl in der IDT CLI:

`timeout-multiplier`

Gibt einen Wert größer als 1,0 an, der beim Ausführen von Tests auf alle Timeouts 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, IDT berechnet damit den Wert der Umgebungsvariablen IDT_TEST_TIMEOUT und legt die`config.timeoutMultiplier`-Feld im IDT-Kontext. Um dieses Argument zu unterstützen, müssen Sie Folgendes ausführen:

- Anstatt direkt den Timeout-Wert aus dem`test.json`die Umgebungsvariable IDT_TEST_TIMEOUT lesen, um den korrekt berechneten Timeout-Wert zu erhalten.
- Rufen Sie das`config.timeoutMultiplier`Wert aus dem IDT-Kontext und wenden Sie ihn auf lang laufende Timeouts an.

Weitere Informationen zum vorzeitigen Beendigen aufgrund von Timeout-Ereignissen finden Sie unterGeben Sie Exitverhalten anaus.

`stop-on-first-failure`

Gibt an, dass IDT nicht mehr alle Tests ausführen soll, wenn ein Fehler auftritt.

Wenn ein Testläufer dieses Argument in seinem`run-suite`-Befehl, IDT stoppt die Ausführung von Tests, sobald ein Fehler auftritt. Wenn Testfälle jedoch parallel laufen, kann dies zu unerwarteten Ergebnissen führen. Um den Support zu implementieren, stellen Sie sicher, dass Ihre Testlogik alle laufenden Testfälle anweist, temporäre Ressourcen anzuhalten, temporäre Ressourcen zu bereinigen und ein Testergebnis an IDT zu melden. Weitere Informationen zum vorzeitigen Beendigen von Fehlern finden Sie unterGeben Sie Exitverhalten anaus.

`group-id` und `test-id`

Gibt an, dass IDT nur die ausgewählten Testgruppen oder Testfälle ausführen soll.

Testläufer können diese Argumente mit ihren`run-suite`Befehl zur Angabe des folgenden Testausführungsverhaltens:

- Führen Sie alle Tests innerhalb der angegebenen Testgruppen aus.
- Führen Sie eine Auswahl von Tests innerhalb einer bestimmten Testgruppe aus.

Um diese Argumente zu unterstützen, muss der Zustandscomputer für Ihre Testsuite einen bestimmten Satz von enthaltenRunTaskundChoiceZustände in Ihrem Zustandsautomaten.

Wenn Sie keinen benutzerdefinierten Zustandscomputer verwenden, enthält der Standard-IDT-Statuscomputer die für Sie erforderlichen Status, und Sie müssen keine zusätzlichen Maßnahmen ergreifen. Wenn Sie jedoch einen benutzerdefinierten Zustandscomputer verwenden, verwenden SieBeispiel für einen Zustandsautomaten: Ausführen von vom Benutzer ausgewählten Testgruppenals Beispiel, um die erforderlichen Zustände in Ihrem Zustandsmaschine hinzuzufügen.

Weitere Informationen zu IDT-CLI-Befehlen finden Sie unterDebuggen und Ausführen von benutzerdefinierten Test-Suitenaus.

Schreiben Sie Ereignisprotokolle

Während der Test läuft, senden Sie Daten an`stdout`und`stderr`um Ereignisprotokolle und Fehlermeldungen an die Konsole zu schreiben. Weitere Informationen zum Format von Konsolenmeldungen finden Sie unterNachrichtenformat der Konsoleaus.

Wenn der IDT die Testsuite beendet hat, sind diese Informationen auch im`test_manager.log`-Datei befindet sich im Verzeichnis*`<devicetester-extract-location>`*/`results/`*`<execution-id>`*`/logs`folder.

Sie können jeden Testfall so konfigurieren, dass er die Protokolle von seinem Testlauf, einschließlich Logs vom zu testenden Gerät, in die*`<group-id>`*_*`<test-id>`*-Datei befindet sich im Verzeichnis*`<device-tester-extract-location>`*`/results/`*`execution-id`*`/logs`folder. Rufen Sie dazu den Pfad zur Protokolldatei aus dem IDT-Kontext mit dem`testData.logFilePath`abfragen, erstellen Sie eine Datei unter diesem Pfad und schreiben Sie den gewünschten Inhalt. IDT aktualisiert den Pfad automatisch basierend auf dem laufenden Testfall. Wenn Sie die Protokolldatei für einen Testfall nicht erstellen möchten, wird für diesen Testfall keine Datei generiert.

Sie können Ihre ausführbare Textdatei auch so einrichten, dass weitere Protokolldateien nach Bedarf im*`<device-tester-extract-location>`*`/logs`folder. Wir empfehlen Ihnen, eindeutige Präfixe für Logdateinamen 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>`*`/`aus. 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 IDT-Testergebnisse und -protokolle

So füllen Sie den Inhalt der *suite-name*`_report.xml`-Datei verwenden, müssen Sie die`SendResult`Befehl, um 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 keine Ergebnisse über die API melden, sucht IDT im Ordner „Testartefakte“ nach Testergebnissen. Der Pfad zu diesem Ordner wird im Verzeichnis gespeichert`testData.testArtifactsPath`him IDT-Kontext eingereicht. In diesem Ordner verwendet IDT die erste alphabetisch sortierte XML-Datei, die sie als Testergebnis findet.

Wenn Ihre Testlogik JUnit XML-Ergebnisse erzeugt, können Sie die Testergebnisse in eine XML-Datei im Artefaktordner schreiben, um die Ergebnisse direkt an IDT bereitzustellen, anstatt die Ergebnisse zu analysieren und sie dann mit der API an IDT zu übermitteln.

Wenn Sie diese Methode verwenden, stellen Sie sicher, dass Ihre Testlogik die Testergebnisse genau zusammenfasst und Ihre Ergebnisdatei im gleichen Format wie die *suite-name*`_report.xml`file. IDT führt mit den folgenden Ausnahmen keine Validierung der von Ihnen bereitgestellten Daten durch:

- IDT ignoriert alle Eigenschaften des`testsuites`markieren. Stattdessen berechnet es die Tag-Eigenschaften aus anderen gemeldeten Testgruppenergebnissen.
- Mindestens eine`testsuite`Tag muss innerhalb existieren`testsuites`aus.

Da IDT für alle Testfälle denselben Artefaktordner verwendet und keine Ergebnisdateien zwischen Testläufen löscht, kann diese Methode auch zu fehlerhaften Berichten führen, wenn IDT die falsche Datei liest. Wir empfehlen Ihnen, in allen Testfällen denselben Namen für die generierte XML-Ergebnisdatei zu verwenden, um die Ergebnisse für jeden Testfall zu überschreiben und sicherzustellen, dass IDT die richtigen Ergebnisse verfügbar sind. Obwohl Sie einen gemischten Ansatz für die Berichterstellung in Ihrer Testsuite verwenden können, dh für einige Testfälle eine XML-Ergebnisdatei verwenden und Ergebnisse über die API für andere übermitteln, empfehlen wir diesen Ansatz nicht.

Geben Sie Exitverhalten an

Konfigurieren Sie Ihre ausführbare Textdatei so, dass sie immer mit einem Exit-Code von 0 beendet wird, auch wenn ein Testfall einen Fehler oder ein Fehlerergebnis meldet. Verwenden Sie Exit-Codes ungleich Null nur, um anzugeben, dass ein Testfall nicht ausgeführt wurde oder ob die ausführbare Testfalldatei keine Ergebnisse an IDT übermitteln konnte. Wenn IDT einen Exit-Code ungleich Null erhält, wird darauf hingewiesen, dass der Testfall auf einen Fehler gestoßen ist, der die Ausführung verhindert hat.

IDT fordert möglicherweise an oder erwartet, dass ein Testfall nicht mehr ausgeführt wird, bevor er in den folgenden Ereignissen abgeschlossen ist. Verwenden Sie diese Informationen, um Ihre ausführbare Testfalldatei so zu konfigurieren, dass jedes dieser Ereignisse aus dem Testfall erkannt wird:

Timeout (Zeitüberschreitung)

Tritt auf, wenn ein Testfall länger als der in der`test.json`file. Wenn der Testläufer den`timeout-multiplier`-Argument, um einen Timeout-Multiplikator anzugeben, dann 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, legt IDT den Wert der Umgebungsvariablen IDT_TEST_TIMEOUT auf den berechneten Timeout-Wert (in Sekunden) fest und übergibt die Variable an die ausführbare Testfalldatei. Sie können den Variablenwert lesen, um einen geeigneten Timer einzustellen.

Unterbrechen

Tritt auf, wenn der Testläufer IDT unterbricht. Zum Beispiel durch DrückenCtrl+Caus.

Da Terminals Signale an alle untergeordneten Prozesse übertragen, können Sie in Ihren Testfällen einfach einen Signalhandler konfigurieren, um Interruptsignale zu erkennen.

Alternativ können Sie die API regelmäßig abfragen, um den Wert des`CancellationRequested`In der boolesche Werte`PollForNotifications`-API-Antwort. Wenn IDT ein Interruptsignal empfängt, wird der Wert des`CancellationRequested`In boolesche Werte`true`aus.

Stoppen Sie beim ersten Fehler

Tritt auf, wenn ein Testfall, der parallel zum aktuellen Testfall läuft, fehlschlägt und der Testläufer die`stop-on-first-failure`-Argument, um anzugeben, dass IDT angehalten werden soll, wenn ein Fehler auftritt.

Um dieses Ereignis zu erkennen, können Sie die API regelmäßig abfragen, um den Wert des`CancellationRequested`In der boolesche Werte`PollForNotifications`-API-Antwort. Wenn IDT auf einen Fehler stößt und so konfiguriert ist, dass es beim ersten Fehler angehalten wird, wird der Wert des`CancellationRequested`In boolesche Werte`true`aus.

Wenn eines dieser Ereignisse auftritt, wartet IDT 5 Minuten, bis alle derzeit ausgeführten Testfälle abgeschlossen sind. Wenn alle laufenden Testfälle nicht innerhalb von 5 Minuten beendet werden, zwingt IDT jeden ihrer Prozesse zum Stoppen. Wenn IDT vor Ablauf der Prozesse keine Testergebnisse erhalten hat, werden die Testfälle als Zeitüberschreitung gekennzeichnet. Als Best Practice sollten Sie sicherstellen, dass Ihre Testfälle die folgenden Aktionen ausführen, wenn sie auf eines der Ereignisse stoßen:

1. Hör auf, normale Testlogik auszuführen.
2. Bereinigen Sie temporäre Ressourcen wie Testartefakte auf dem getesteten Gerät.
3. Melden Sie ein Testergebnis an IDT, z. B. einen Testfehler oder einen Fehler.
4. Verlassen Sie.

Verwenden Sie den IDT-Kontext

Wenn IDT eine Testsuite ausführt, kann die Testsuite auf eine Reihe von Daten zugreifen, die verwendet werden können, um zu bestimmen, wie jeder Test ausgeführt wird. Diese Daten werden als IDT-Kontext bezeichnet. Zum Beispiel, Benutzerdatenkonfiguration, die von Testläufern in einem`userdata.json`wird Test-Suiten im IDT-Kontext zur Verfügung gestellt.

Der IDT-Kontext kann als schreibgeschütztes JSON-Dokument angesehen werden. Test-Suites können Daten aus dem Kontext abrufen und Daten mit Standard-JSON-Datentypen wie Objekten, Arrays, Zahlen usw. 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 von der`config.json`Ordneraus. Die`config`enthält auch das folgende zusätzliche Feld:

`config.timeoutMultiplier`

Der Multiplikator für den von der Testsuite verwendeten Timeout-Wert. Dieser Wert wird vom Testläufer aus der IDT CLI angegeben. Der Standardwert ist 1.

`device`

Informationen über das für den Testlauf ausgewählte Gerät. Diese Information entsprichtdevicesArray-Element imdevice.jsonOrdnerfür das ausgewählte Gerät.

`devicePool`

Informationen über den Gerätepool, der für den Testlauf ausgewählt wurde. Diese Information entspricht dem Gerätepool-Array-Element der obersten Ebene, das in derdevice.json-Datei für den ausgewählten Gerätepool.

`resource`

Informationen über Ressourcengeräte aus demresource.jsonfile.

`resource.devices`

Diese Information entsprichtdevices-Array definiert imresource.jsonfile. Jederdevices-Element enthält das folgende zusätzliche Feld:

`resource.device.name`

Der Name des Ressourcengeräts. Dieser Wert wird aufrequiredResource.nameWert imtest.jsonfile.

`testData.awsCredentials`

DieAWSAnmeldeinformationen, die vom Test verwendet werden, um eine Verbindung mit demAWS-Wolke. Diese Informationen erhalten Sie von derconfig.jsonfile.

`testData.logFilePath`

Der Pfad zu der Protokolldatei, in die der Testfall Protokollmeldungen schreibt. Die Testsuite erstellt diese Datei, falls sie nicht vorhanden ist.

`userData`

Informationen, die der Testläufer imuserdata.jsonOrneraus.

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 demGetContextValueundGetContextStringAPIs. Die Syntax für JsonPath-Strings für den Zugriff auf den IDT-Kontext variiert wie folgt:

- In `suite.json` und `test.json` verwenden Sie `{{query}}` aus. Das heißt, benutze das Wurzelelement nicht `$.` um deinen Ausdruck zu beginnen.
- In `statemachine.json` verwenden Sie `{{$.query}}` aus.
- In API-Befehlen verwenden Sie *query* oder `{{$.query}}`, je nach Befehl. Weitere Informationen finden Sie in der Inline-Dokumentation in den SDKs.

In der folgenden Tabelle werden die Operatoren in einem typischen JsonPath-Ausdruck beschrieben:

Operator	Description
`$`	The root element. Because the top-level context value for IDT is an object, you will typically use `$.` to start your queries.
`.childName`	Accesses the child element with name `ChildName` from an object. If applied to an array, yields a new array with this operator applied to each element. The element name is case sensitive. For example, the query to access the `awsRegion` value in the `config` object is `$.config.awsRegion`.
`[start:end]`	Filters elements from an array, retrieving items beginning from the `Start` index and going up to the `Ende` index, both inclusive.
`[index1, index2,..., indexN]`	Filters elements from an array, retrieving items from only the specified indices.
`[? (expr)]`	Filters elements from an array using the `expr` expression. This expression must evaluate to a boolean value.

Verwenden Sie die folgende Syntax, um Filterausdrücke zu erstellen:

```
<jsonpath> | <value> operator <jsonpath> | <value>
```

In dieser Syntax gilt:

- `jsonpath`ist ein JsonPath, der die standardmäßige JSON-Syntax verwendet.
- `value`ist ein benutzerdefinierter Wert, der die Standard-JSON-Syntax verwendet.
- `operator`ist einer der folgenden Operatoren:
  - <(kleiner als)
  - <=(kleiner als oder gleich)
  - ==(Gleich)

    Wenn der JsonPath oder Wert in Ihrem Ausdruck ein Array, ein boolescher Wert oder ein Objektwert ist, ist dies der einzige unterstützte binäre Operator, den Sie verwenden können.
  - >=(größer als oder gleich)
  - >(größer als)
  - =~(Reguläre Ausdrücke stimmen überein). Um diesen Operator in einem Filterausdruck verwenden zu können, muss der JsonPath oder Wert auf der linken Seite Ihres Ausdrucks zu einer Zeichenfolge ausgewertet werden, und die rechte Seite muss ein Musterwert sein, der aufRE2-Syntaxaus.

Sie können JsonPath-Abfragen im Formular verwenden {{*abfragen*}} als Platzhalterzeichenfolgen innerhalb der`args`und`environmentVariables`-Felder in`test.json`Dateien und innerhalb der`environmentVariables`-Felder in`suite.json`Dateien. IDT führt eine Kontextsuche durch und füllt die Felder mit dem ausgewerteten Wert der Abfrage aus. Zum Beispiel im`suite.json`können Sie Platzhalterzeichenfolgen verwenden, um Umgebungsvariablenwerte anzugeben, die sich bei 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 berücksichtigen Sie die folgenden Überlegungen für Ihre Anfragen:

- Sie müssen jedes Vorkommen des`devicePool`geben Sie Ihre Anfrage in Kleinbuchstaben ein. Das heißt, benutze`devicepool`Stattdessen.
- Für Arrays können Sie nur Arrays von Strings verwenden. Darüber hinaus verwenden Arrays einen nicht standardmäßigen`item1, item2,...,itemN`. Wenn das Array nur ein Element enthält, wird es als serialisiert`item`, wodurch es von einem String-Feld nicht zu unterscheiden ist.
- Sie können keine Platzhalter verwenden, um Objekte aus dem Kontext abzurufen.

Aufgrund dieser Überlegungen empfehlen wir, dass Sie nach Möglichkeit die API verwenden, um auf den Kontext in Ihrer Testlogik zuzugreifen, anstatt auf Platzhalterzeichenfolgen `intest.jsonundsuite.json`Dateien. In einigen Fällen ist es jedoch möglicherweise bequemer, JsonPath-Platzhalter zu verwenden, um einzelne Strings abzurufen, die als Umgebungsvariablen festgelegt werden sollen.

Konfigurieren von Einstellungen für Testläufer

Um benutzerdefinierte Test-Suiten auszuführen, müssen Testläufer ihre Einstellungen basierend auf der Testsuite konfigurieren, die sie ausführen möchten. Die Einstellungen werden basierend auf JSON-Konfigurationsdateivorlagen angegeben, die sich im*<device-tester-extract-location>*`/configs/`folder. Bei Bedarf müssen Testläufer auch eingerichtet werdenAWSAnmeldeinformationen, die IDT verwendet, um eine Verbindung mit demAWSCloud.

Als Testschreiber müssen Sie diese Dateien so konfigurierendebuggen Sie Ihre Testsuiteaus. Sie müssen Anweisungen geben, um Läufer zu testen, damit sie die folgenden Einstellungen für die Ausführung Ihrer Test-Suites nach Bedarf konfigurieren können.

Konfigurieren von device.json

Die`device.json`-Datei enthält Informationen zu den Geräten, auf denen Tests ausgeführt werden (z. B. IP-Adresse, Anmeldeinformationen, Betriebssystem und CPU-Architektur).

Testläufer können diese Informationen mittels der folgenden Vorlage bereitstellen`device.json`Datei im Verzeichnis*<device-tester-extract-location>*`/configs/`folder.

```
[
    {
        "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 Sie Ihr Board im AWS Partner-Gerätekatalog auflisten möchten, muss die hier angegebene SKU mit der SKU übereinstimmen, die Sie während des Einreichungsprozesses verwenden.

`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 zu den Feature-Namen und Werten zur Verfügung stellen, die in die`device.json`file. 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 namens`MQTT_QOS`aus. Testläufer geben diesen Funktionsnamen an und legen den Funktionswert auf die QOS-Ebenen fest, die von ihrem Gerät unterstützt werden. Sie finden die bereitgestellten Informationen imIDT-Kontextmit dem`devicePool.features`Abfrage oder von derKontext der Zustandsautomatenmit dem`pool.features`abfragen.

`features.name`

Der Name der Funktion.

`features.value`

Die unterstützten Feature-Werte.

`features.configs`

Bei Bedarf Konfigurationseinstellungen für das Feature.

`features.config.name`

Der Name der Konfigurationseinstellung.

`features.config.value`

Die unterstützten Einstellungswerte.

`devices`

Eine Reihe von Geräten im Pool, die getestet werden sollen. Mindestens ein Gerät muss ausgewählt werden.

`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 die Werte unterstützt`ssh`und`uart`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.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. Der serielle Port, an den 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 für den Benutzer im Container. Der Standardwert ist der in der 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 abrufen`pool.Devices[0].Connectivity.Protocol`aus dem Zustandsmaschine-Kontext aus und vergleichen Sie ihn mit dem erwarteten Wert

in a`Choice`Zustand. Wenn ein falsches Protokoll verwendet wird, drucken Sie eine Nachricht mit`LogMessage`Staat und Übergang zum`Fail`Zustand.
Alternativ können Sie den Fehlerbehandlungscode verwenden, um einen Testfehler für falsche Gerätetypen zu melden.

(Optional) Benutzerdata.json konfigurieren

Die`userdata.json`-Datei enthält alle zusätzlichen Informationen, die von einer Testsuite benötigt werden, aber nicht im`device.json`file. Das Format dieser Datei hängt vom`userdata_scheme.json`OrdnerDas ist in der Testsuite definiert. Wenn Sie ein Testautor sind, stellen Sie sicher, dass Sie diese Informationen Benutzern zur Verfügung stellen, die die von Ihnen geschriebenen Test-Suites ausführen.

(Optional) resource.json konfigurieren

Die`resource.json`enthält Informationen über alle Geräte, 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 herstellen kann. Ressourcengeräte sind optional und Sie können so viele Ressourcengeräte benötigen, wie Sie benötigen. Als Testautor benutzt du dastest.json-Dateium die Funktionen des Ressourcengeräts zu definieren, die für einen Test erforderlich sind. Testläufer benutzen 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 Test-Suiten ausführen.

Testläufer können diese Informationen mittels der folgenden Vorlage bereitstellen`resource.json`Datei im Verzeichnis*`<device-tester-extract-location>`*`/configs/`folder.

```
[
    {
        "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 imtest.json-Dateienin der Testsuite und bestimmt, welche Tests ausgeführt werden sollen 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, um anzugeben, wie viele Tests das Gerät gleichzeitig verwenden können. Der Standardwert ist 1.

`devices`

Eine Reihe von Geräten im Pool, die getestet werden sollen. Mindestens ein Gerät muss ausgewählt werden.

`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 die Werte unterstütztsshunduartfür physische Geräte unddockerfür Docker-Container.

`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. Der serielle Port, an den 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 für den Benutzer im Container. Der Standardwert ist der in der Dockerfile angegebene Benutzer.

Der Standardwert ist 22.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `docker` festgelegt ist.

(Optional) Config.json konfigurieren

Die`config.json`enthält Konfigurationsinformationen für IDT. In der Regel müssen Testläufer diese Datei nicht ändern, außer um ihreAWSBenutzeranmeldeinformationen für IDT und optional einAWSRegion. WennAWSAnmeldeinformationen mit erforderlichen Berechtigungen werden bereitgestelltAWS IoTDevice Tester sammelt und übermittelt Nutzungsmetriken anAWSaus. Dies ist eine optionale Funktion und wird verwendet, um IDT-Funktionalität zu verbessern. Weitere Informationen finden Sie unter IDT-Nutzungsmetriken.

Testläufer können ihreAWSAnmeldeinformationen auf eine der folgenden Arten:

- Anmeldeinformationen-Datei

  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. Während einer SSH-Sitzung definierte Variablen sind nach dem Schließen dieser Sitzung nicht verfügbar. IDT kann das`AWS_ACCESS_KEY_ID`und`AWS_SECRET_ACCESS_KEY`Zu speicherende UmgebungsvariablenAWSReferenzen

  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>
```

So konfigurieren SieAWSAnmeldeinformationen für IDT, Testläufer bearbeiten`auth`-Abschnitt im`config.json`Datei im Verzeichnis*<device-tester-extract-location>*`/configs/`folder.

```
{
    "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 zum*<device-tester-extract-location>*aus.

`log.location`

Der Pfad zum Logs-Ordner im*<device-tester-extract-location>*aus.

`configFiles.root`

Der Pfad zu dem Ordner, in dem sich die Konfigurationsdateien befinden.

`configFiles.device`

Der Pfad zum`device.json`file.

`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 Test-Suiten verwenden werden. Wenn nicht festgelegt, verwenden Test-Suites die Standardregion, die in jeder Testsuite angegeben ist.

`auth.method`

Die Methode, die IDT zum Abrufen verwendet`AWS`-Anmeldeinformationen. Unterstützte Werte sind`file`um Anmeldeinformationen aus einer -Anmeldeinformationsdatei abzurufen, und`environment`um Anmeldeinformationen mit Umgebungsvariablen abzurufen.

`auth.credentials.profile`

Das aus der Anmeldeinformationen zu verwendende -Anmeldeinformationsprofil. Diese Eigenschaft gilt nur, wenn `auth.method` auf `file` festgelegt ist.

Debuggen und Ausführen von benutzerdefinierten Test-Suiten

Nach demErforderliche Konfigurationeingestellt ist, kann IDT Ihre Testsuite ausführen. Die Laufzeit der vollständigen Testsuite hängt von der Hardware und der Zusammensetzung der Testsuite ab. Zur Referenz: Es dauert etwa 30 Minuten, die vollständige Ausführung auszuführen`AWS IoT GreengrassQualifikationstestsuite auf einem Raspberry Pi 3B`

Wenn Sie Ihre Testsuite schreiben, können Sie IDT verwenden, um die Testsuite im Debug-Modus auszuführen, um Ihren Code zu überprüfen, bevor Sie ihn ausführen, oder ihn Testläufer bereitstellen.

Führen Sie IDT im Debug-Modus aus

Da Test-Suiten von IDT abhängig sind, um mit Geräten zu interagieren, den Kontext bereitzustellen und Ergebnisse zu erhalten, können Sie Ihre Testsuiten nicht einfach in einer IDE ohne IDT-Interaktion debuggen. Zur Verfügung stellt die IDT CLI`debug-test-suite`-Befehl, mit dem Sie IDT im Debug-Modus ausführen können. Führen Sie den folgenden Befehl aus, um die verfügbaren Optionen anzuzeigen`debug-test-suite`:

```
devicetester_[linux | mac | win_x86-64] debug-test-suite -h
```

Wenn Sie IDT im Debug-Modus ausführen, startet IDT die Testsuite nicht wirklich oder führt den State Machine aus. Stattdessen interagiert es mit Ihrer IDE, um auf Anfragen der Testsuite zu reagieren, die in der IDE ausgeführt wird, und druckt die Protokolle an die Konsole. IDT hat keine Zeitüberschreitung und wartet auf die Beendigung, bis sie manuell unterbrochen wird. Im Debug-Modus führt IDT den Statuscomputer 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 erhält. Stellen Sie sicher, dass Sie die folgenden Informationen angeben:

- Umgebungsvariablen und Argumente für jeden Test. IDT liest diese Informationen nicht aus`test.json`oder`suite.json`aus.
- Argumente zur Auswahl von Ressourcengeräten. IDT liest diese Informationen nicht aus`test.json`aus.

Führen Sie die folgenden Schritte aus, um Ihre Test-Suites zu debuggen:

1. Erstellen Sie die Einstellungskonfigurationsdateien, die zum Ausführen der Testsuite erforderlich sind. Wenn Ihre Testsuite beispielsweise die`device.json`,`resource.json`, und`user data.json`, stellen Sie sicher, dass 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 zum Ausführen des Tests erforderlich sind.

   ```
   devicetester_[linux | mac | win_x86-64] debug-test-suite [options]
   ```

   Nachdem Sie diesen Befehl ausgeführt haben, wartet IDT auf Anfragen aus der Testsuite und antwortet dann darauf. IDT generiert auch die Umgebungsvariablen, die für den Fallprozess für das IDT Client SDK erforderlich sind.

3. Verwenden Sie in Ihrer IDE das`run`oder`debug`Konfiguration für das Ausführen der folgenden Aktionen:

   a. Legen Sie die Werte der IDT-generierten Umgebungsvariablen fest.
   b. Legen Sie den Wert aller Umgebungsvariablen oder Argumente fest, die Sie in Ihrem`test.json`und`suite.json`file.
   c. Legen Sie Haltepunkte nach Bedarf fest.
4. Führen Sie die Testsuite in Ihrer IDE aus.

   Sie können die Testsuite so oft wie nötig debuggen und erneut ausführen. IDT führt im Debugging-Modus nicht aus.
5. Unterbrechen Sie nach dem Debuggen IDT, um den Debug-Modus zu beenden.

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 in diesem Fall die unterstützten Produkte für Ihre IDT-Version aufAWS IoT GreengrassVersionen undAWS IoT GreengrassVersionen der Qualifikationstestsuite sind für die aktuelle IDT-Version verfügbar.

`list-test-cases`

Listet die Testfälle in einer bestimmten Testgruppe auf. Die folgende Option wird unterstützt:

- `group-id`aus. 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 häufig verwendete Optionen:

- `suite-id`aus. Die auszuführende Testsuite-Version. Wenn nicht angegeben, verwendet IDT die neueste Version im `tests`-Ordner.
- `group-id`aus. Die auszuführenden Testgruppen als kommagetrennte Liste. Bei fehlender Angabe führt IDT alle Testgruppen in der Testsuite aus.
- `test-id`aus. Die auszuführenden Testfälle als kommagetrennte Liste. Wenn angegeben, muss `group-id` eine einzelne Gruppe angeben.
- `pool-id`aus. Der zu testende Gerätepool. Testläufer müssen einen Pool angeben, wenn mehrere Gerätepools in Ihrem`device.json`file.
- `timeout-multiplier`aus. Konfiguriert IDT, um das Timeout für die Testausführung zu ändern, das in der`test.json`-Datei für einen Test mit einem benutzerdefinierten Multiplikator.
- `stop-on-first-failure`aus. Konfiguriert IDT so, dass die Ausführung beim ersten Fehler beendet wird. Diese Option sollte mit `group-id` verwendet werden, um die angegebenen Testgruppen zu debuggen.
- `userdata`aus. Legt die Datei fest, die Benutzerdateninformationen enthält, die zum Ausführen der Testsuite erforderlich sind. Dies ist nur erforderlich, wenn`userdataRequired`ist in der`suite.json`Datei für die Testsuite

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 Debugging-Modus aus. Weitere Informationen finden Sie unter Führen Sie IDT im Debug-Modus aus.

Überprüfen Sie IDT-Testergebnisse und -protokolle

In diesem Abschnitt wird das Format beschrieben, in dem IDT Konsolenprotokolle und Testberichte generiert.

Nachrichtenformat der Konsole

AWS IoTDevice Tester verwendet ein Standardformat zum Drucken von Nachrichten an die Konsole, wenn eine Testsuite gestartet wird. Der folgende Auszug zeigt ein Beispiel für eine von IDT generierte Konsolenmeldung.

```
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 Konsolenmeldungen 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 Nachrichten eine von`info`,`warn`, oder`error`aus. IDT gibt eine`fatal`oder`panic`Meldung, wenn es auf ein erwartetes Ereignis stößt, 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.

Konsolenmeldungen, die aus einer Testsuite generiert wurden, liefern 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 Test-Suite-spezifische Teil der Konsolennachricht enthält die folgenden Felder:

`suiteId`

Name der derzeit laufenden Testsuite

`groupId`

Die ID der derzeit laufenden Testgruppe.

`testCaseId`

ID des laufenden Testfalls.

`deviceId`

Eine ID des zu testenden Geräts, das der aktuelle Testfall verwendet.

Um eine Testzusammenfassung auf die Konsole zu drucken, wenn ein IDT einen Test ausgeführt hat, müssen Sie eine`ReportBundesstaat`In Ihrem Zustandsautomaten. Die Testzusammenfassung enthält Informationen über die Testsuite, die Testergebnisse für jede ausgeführte Gruppe und die Speicherorte der generierten Protokolle und Berichtsdateien. Das folgende Beispiel zeigt eine Testzusammenfassungsmeldung.

```
========== 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 IoTDevice Tester-Berichtsschema

`awsiotdevicetester_report.xml`Es handelt sich um einen signierten Bericht, der die folgenden Informationen enthält:

- Die IDT-Version.
- Die Testsuite-Version.
- Die Unterschrift und der Schlüssel des Berichts, mit denen der Bericht signiert wurde.
- Device SKU und Gerätename, die im`device.json`file.
- Die Produktversion und die getesteten Gerätefunktionen.
- Die aggregierte Zusammenfassung der Testergebnisse. Diese Informationen sind die gleichen wie die in der*suite-name*`_report.xml`file.

```
<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 Funktionen markiert als`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>
```

Funktionen markiert als`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>
```

Testsuite-Berichtsschema

Der Bericht *`suite-name`*`_Result.xml` wird im JUnit-XML-Format erstellt. Sie können ihn in Continuous Integration and Deployment-Plattformen wie Jenkins, Bamboo usw. integrieren. Der -Bericht enthält eine aggregierte 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>
```

Der Berichtsabschnitt in beiden`awsiotdevicetester_report.xml`oder*`suite-name`*`_report.xml`Führt die Tests und die Ergebnisse auf, die ausgeführt wurden.

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`

Zeit (in Sekunden), die zur Ausführung der Testsuite 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.

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 Ordner *< device-tester-extract-location >*
- 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.jso` n-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 fortthe 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/signup.
2. Folgen Sie den Online-Anweisungen.

   Bei der Anmeldung müssen Sie auch einen Telefonanruf entgegennehmen und einen Verifizierungscode über die Telefontasten eingeben.

   Wenn Sie sich für eine anmelden AWS-Konto, Root-Benutzer des AWS-Kontoswird eine erstellt. Der Root-Benutzer hat Zugriff auf alle AWS -Services und Ressourcen des Kontos. Aus Sicherheitsgründen sollten Sie einem Benutzer Administratorzugriff zuweisen und nur den Root-Benutzer verwenden, um Aufgaben auszuführen, für die Root-Benutzerzugriff erforderlich ist.

AWS sendet Ihnen nach Abschluss des Anmeldevorgangs eine Bestätigungs-E-Mail. Sie können jederzeit Ihre aktuelle Kontoaktivität anzeigen und Ihr Konto verwalten. Rufen Sie dazu https://aws.amazon.com/ auf und klicken Sie auf Mein Konto.

Erstellen Sie einen Benutzer 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 Management Consoleals 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 Sie einen Benutzer 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 Benutzer in IAM Identity Center Administratorzugriff.

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

Melden Sie sich als Benutzer mit Administratorzugriff an

- 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 Zugriffsportal.

Weisen Sie weiteren Benutzern Zugriff zu

1. Erstellen Sie in IAM Identity Center einen Berechtigungssatz, der der bewährten Methode zur Anwendung von Berechtigungen mit den geringsten Rechten folgt.

Anweisungen finden Sie im Benutzerhandbuch unter Einen Berechtigungssatz erstellen.AWS IAM Identity Center

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

   Anweisungen finden Sie im AWS IAM Identity Center Benutzerhandbuch unter Gruppen hinzufügen.

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 Management Console 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 Policies (Richtlinien) und dann Create policy (Richtlinie erstellen).

   b. Ersetzen Sie auf der Registerkarte JSON den Platzhalterinhalt durch die folgende Richtlinie.

```
{
    "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 IDT UsageMetrics IamPermissions-Richtlinie, die Sie im vorherigen Schritt erstellt haben. Markieren Sie das Kontrollkästchen.

   c. Wählen Sie Weiter: Markierungen.

   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 Show (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 Ihres Geräts 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. Ersetzen *`Sie den Benutzernamen`* 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) Config.json konfigurieren.

Fehlerbehebung in IDT für AWS IoT Greengrass

IDT für AWS IoT Greengrass schreibt diese Fehler je nach Art der Fehler an verschiedene Stellen. 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 *<device-tester-extract-location>* `/results`. Wenn Sie das Problem nicht beheben können, wenden Sie sich bitte

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
			an AWS Developer Support aus.
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). • Ein langsames Netzwerk beeinträchtigt die Kommunikation zwischen Gerät und Cloud. • Das Feld `timeout` in Testkonfigurationsdateien (`test.json`) wurde versehentlich geändert.	• Überprüfen Sie die Netzwerkverbindung und -geschwindigkeit. • Stellen Sie sicher, dass Sie keine Datei im Verzeichnis `/test` geändert haben. • Versuchen Sie, die fehlerhafte Testgruppe manuell mit dem Flag `"--group-id"` 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	PlatformNotSupportError	Eine falsche Kombination aus Betriebssystem und Architektur ist in `device.json` 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 verwendeten Version von IDT nicht unterstützt.	Verwenden Sie den Befehl device_tester_bin version, um die unterstützte Version der AWS IoT Greengrass Core-Software zu suchen. Wenn Sie beispielsweise macOS nutzen, verwenden Sie: ./devicetester_mac_x86_64 version. So suchen Sie die Version der AWS IoT Greengrass Core-Software, die Sie verwenden: • Wenn Sie Tests mit vorinstallierten ausführenAWS IoT Greengrass sVerwenden Sie SSH, um eine Verbindung zu IhremAWS IoT Greengrass sCore-Gerät und Run***<path-to-preinstalled-greengrass-loc***

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
			ation> /greengrass/ggc/core/greengrassd --version • Wenn Sie Tests mit einer anderen Version der AWS IoT Greengrass Core-Software ausführen, navigieren Sie zum Verzeichnis `devicetester_greengrass_` ***<os>***`/products/greengrass/gcc`. Die AWS IoT Greengrass Core-Softwareversion ist im ZIP-Dateinamen enthalten. 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 nur für AWS IoT Greengrass-Bibliotheken und SDKs.	Stellen Sie Folgendes sicher: • Beim SDK-Paket unter `devicetester_greengrass_ <os>/products/greengrass/ggsdk` handelt es sich um das Python SDK. • Der Inhalt des Ordners `bin` unter `devicetester_greengrass_ <os> /tests/GGQ_1.0.0/suite/resources/runtimefarm/bin` wurde nicht geändert.

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
106	ValidationError	Manche Felder in `device.json` oder `config.json` sind ungültig.	Überprüfen Sie die Fehlermeldung auf der rechten Seite des Fehlercodes im Bericht. • Ungültiger Auth Typ 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 privater Schlüsselpfad: Geben Sie den richtigen Pfad zu Ihrem privaten Schlüssel an. Weitere Informationen finden Sie unter Konfigurieren von device.json. • UngültigAWS-Region: Geben

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
			Sie eine gültigeAWS-RegionIn Ihrem`config.json` file. Weitere Informationen finden Sie unter AWS-Service-Endpunkte. • AWSAnmeldedaten: Gültig einstellenAWS-Anmeldeinformationen auf Ihrem Testgerät (mithilfe von Umgebungsvariablen oder dem`credentials` -Datei). Überprüfen Sie, ob das Feld `auth` korrekt konfiguriert ist. Weitere Informationen finden Sie unter the section called “Erstellen und konfigurieren Sie ein AWS-Konto”. • Ungültige HSM-Eingabe: Überprüfen Sie Ihrep11`Provid`

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
			er ,privateKeyLabel ,slotLabel ,slotUserPin , undopenSSLEngine -Felder indevice.js on aus.
107	SSHConnectionFailed	Das Testgerät kann keine Verbindung mit dem konfigurierten Gerät herstellen.	Überprüfen Sie, ob die folgenden Felder in Ihrer Datei device.json korrekt sind: • ip • user • privKeyPath • password 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 `device.json` 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 zu der AWS IoT Greengrass Core-Software ist falsch.	Überprüfen Sie, ob der Pfad in der Fehlermeldung gültig ist. Bearbeiten Sie keine Dateien im Verzeichnis `devicetester_greengrass_` *`<os>`*.
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. • `limits.config` 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	FileNotFoundError	Eine erforderliche Datei wurde nicht gefunden.	Überprüfen Sie Folgendes: • Es gibt eine komprimierte Greengrass-Datei unter `devicetester_greengrass_ <os>/products/greengrass/ggc`. Sie können denAWS IoT GreengrassCore-Tar-Datei aus demAWS IoT GreengrassCore-SoftwareSeite Downloads. • Das SDK-Paket ist unter `devicetester_greengrass_ <os>/products/greengrass/ggsdk` vorhanden. • Die Dateien unter `devicetester_greengrass_ <os>/`

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. • `limits.config` 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 vorhanden ist und dass Sie Schreibberechtigung haben.

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 `suite.json` gültig ist.

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
120	InvalidCredentialError	UngültigAWS-Anmeldeinformationen.	• Überprüfen Sie IhreAWS-Anmeldeinformationen. Weitere Informationen finden Sie unter the section called “Konfigurieren Ihrer AWS-Anmeldeinformationen”. • Überprüfen Sie Ihre Netzwerkverbindung und führen Sie die Testgruppe erneut aus. Netzwerkprobleme können diesen Fehler ebenfalls hervorrufen.
121	AWSSessionError	Fehler beim Erstellen einer AWS-Sitzung.	Dieser Fehler kann auftreten, wennAWSDie Anmeldeinformationen sind ungültig oder die Internetverbindung ist instabil. Versuchen Sie, mithilfe der 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 dasAWS IoT Greengrass-Konsole zum Überprüfen der Konnektivitätsinformationen für denAWS IoT GreengrassKernsache, die vom 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 Anfügen einer Richtlinie an die AWS IoT Greengrass-Servicerolle ist ein Fehler aufgetreten. • Die mit der Servicerolle verknüpfte Richtlinie ist ungültig. • Beim Verknüpfen einer Rolle mit einemAWS-Kontoaus.	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: *<device-tester-extract-location>* /results/ *<execution-id>*/logs/*<test-case-name.log>*
127	InvalidHSMConfiguration	Die bereitgestellte HSM/PKCS-Konfiguration ist nicht korrekt.	Geben Sie in Ihrer Datei `device.json` die erforderliche Konfiguration an, um mithilfe von PKCS#11 mit dem HSM zu interagieren.

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
128	OTAJobNotSuccededError	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 *`<device-tester-extract-location>`* `/results/` *`<execution-id>`*`/logs/` 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 bitte an AWS Developer Support aus.

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 ausführenAWS IoT Greengrassverfügt nicht über die Berechtigung zum Erstellen desAWSzum Ausführen von IDT erforderliche Ressourcen.	Siehe .Berechtigungsrichtlinienvorlage Für die Richtlinienvorlage, die die erforderlichen Berechtigungen zum Ausführen von IDT für bereitstelltAWS IoT Greengrassaus.

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
131	LeftoverAgentExistError	Ihr Gerät führt AWS IoT Greengrass-Prozesse aus, wenn Sie versuchen, IDT für AWS IoT Greengrass zu starten.	Stellen Sie sicher, dass kein Greengrass-Daemon auf Ihrem Gerät ausgeführt wird. • Sie können den folgenden Befehl verwenden, um einen Daemon zu beenden: sudo ./*<absolute-path-to-greengrass-daemon>* / greengrassd stop. • Sie können den Greengrass-Daemon auch über PID beenden. **Note** Wenn Sie eine vorhandene Installation von AWS IoT Greengrass verwenden, die für einen automatischen Start

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
			nach einem Neustart konfiguriert ist, 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	InvalidMLConfiguration	Die angegebene ML-Konfiguration ist falsch.	Geben Sie in Ihrer Datei `device.json` 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”.

Beseitigen von Fehlern in IDT für AWS IoT Greengrass

Wenn Sie IDT verwenden, müssen Sie die richtigen Konfigurationsdateien bereitstellen, bevor Sie IDT für AWS IoT Greengrass ausführen. 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?

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
`test_manager.log`	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.
<test-group-id> `__`*<test-name>* `.log`	Detaillierte Protokolle für den jeweiligen Test.
<test-name> _ggc_logs.tar.gz	Eine komprimierte Sammlung aller Protokoll eAWS IoT GreengrassWährend des Tests generierter Core-Daemon. Weitere Informati onen finden Sie unter Fehlerbehebung in AWS IoT Greengrass.
<test-name> `_ota_logs.tar.gz`	Eine komprimierte Sammlung von Protokoll en, die vom AWS IoT Greengrass-OTA-Age nten während des Tests erstellt wurden. Nur für OTA-Tests.
<test-name> `_basic_assertion_p ublisher_ggad_logs.tar.gz`	Eine komprimierte Sammlung von Protokollen, die vom AWS IoT-Herausgebergerät während des Tests erstellt wurden.
<test-name> `_basic_assertion_s ubscriber_ggad_logs.tar.gz`	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 für die Fehlerbehebung ausreichen. Sollten Sie den Fehler weiterhin nicht finden, können Sie eine Validierung manuell in Ihrer IDE, in einem Text-Editor 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-Einrichtungen verwenden die PuTTy Terminal-Anwendung, um eine Verbindung mit 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 zum Ausführen von Tests auf AWS IoT Greengrass-Geräten eine ältere Version der OpenSSL-Bibliothek (libssl1.0.0). 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 wird nicht ausgeführt. Um IDT auszuführen, müssen Sie eine Sicherheitsausnahme für die`devicetester_mac_x86-64`ausführbar.

So gewähren Sie eine Sicherheitsausnahme für die ausführbare IDT-Datei

1. startenSystemeinstellungenaus dem Apple-Menü.

2. Klicken Sie aufSicherheit & Datenschutz, dann auf derAllgemeinesKlicken Sie auf das Sperrsymbol, um Änderungen an den Sicherheitseinstellungen vorzunehmen.
3. Sehen Sie unter der Nachricht nach`"devicetester_mac_x86-64" was blocked from use because it is not from an identified developer.`und wähleJedenfalls zulassenaus.
4. Akzeptieren der Sicherheitswarnung.

Bei Fragen zur IDT-Support-Richtlinie wenden Sie sich bitte anAWSKundensupportaus.

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

AWS IoTDevice Tester (IDT) fürAWS IoT Greengrassist ein herunterladbares Test-Framework, mit dem Sie validieren undqualifizierendeinAWS IoT GreengrassGeräte zur Einbeziehung in dieAWS PartnerGeräte-Katalogaus. Es wird empfohlen, die neueste Version von zu verwendenAWS IoT Greengrassund IDT, um Ihre Geräte zu testen oder zu qualifizieren. Weitere Informationen finden Sie unterUnterstützte Versionen von IDT fürAWS IoT Greengrass V2imAWS IoT Greengrass Version 2Entwicklerhandbuchaus.

Sie können auch eine der unterstützten Versionen von verwendenAWS IoT Greengrassund IDT, um Ihre Geräte zu testen oder zu qualifizieren. Obwohl Sie weiterhin verwenden könnenNicht unterstützte Versionen von IDTerhalten diese Versionen keine Fehlerbehebungen oder Updates.

> **Important**
>
> Ab dem 4. April 2022AWS IoTDevice Tester (IDT) fürAWS IoT Greengrass V1generiert keine signierten Qualifikationsberichte mehr. Sie können sich nicht mehr als neu qualifizierenAWS IoT Greengrass V1Geräte zum Auflisten imAWS PartnerGeräte-KatalogDurch dasAWSGerätequalifizierungsprogrammaus. Obwohl Sie Greengrass V1-Geräte nicht qualifizieren können, können Sie IDT weiterhin fürAWS IoT Greengrass V1um Ihre Greengrass V1-Geräte zu testen. Es wird empfohlen, zu verwendenIDT fürAWS IoT Greengrass V2um Greengrass-Geräte zu qualifizieren und aufzulistenAWS PartnerGeräte-Katalogaus.

Wenn Sie Fragen zu den Supportrichtlinien haben, wenden Sie sich an den AWS Kundenservice.

Fehlerbehebung für AWS IoT Greengrass

Dieser Abschnitt enthält Informationen zur Fehlerbehebung und zu möglichen Lösungen für Probleme mit AWS IoT Greengrass.

Weitere Informationen zu AWS IoT Greengrass-Kontingenten (Einschränkungen) finden Sie unter Servicekontingente in der Allgemeine Amazon Web Services-Referenz.

Probleme mit AWS IoT Greengrass Core

Wenn die AWS IoT Greengrass Core-Software nicht gestartet wird, führen Sie die folgenden allgemeinen Schritte zur Fehlerbehebung aus:

- 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”.

Durchsuchen Sie die folgenden Faktoren und Fehler, um Informationen zur Behebung von Problemen mit einem -AWS IoT GreengrassKern zu finden.

Problembereiche

- Fehler: In der Konfigurationsdatei fehlt 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: ErrUnknownURIScheme
- Fehler: Laufzeit konnte nicht gestartet werden: Es konnten keine Worker gestartet werden: Zeitüberschreitung für den Container-Test.
- Fehler: Auf lokaler Cloudwatch konnte logGroup nicht aufgerufen PutLogEvents werden: /GreengrassSystem/connection_manager, Fehler: RequestError: Anforderung fehlgeschlagen gesendet durch: Post http://<path>/cloudwatch/logs/: dial tcp <address>: getsockopt: Verbindung verweigert, Antwort: { }.

- Fehler: Server kann nicht erstellt werden, da: chmod /<greengrass-root>/ggc/deployment/lambda/arn:aws:lambda:<region>:<account-id>:function:<function-name>:<version>/<file-name>: keine solche Datei oder kein solches Verzeichnis vorhanden ist.
- Die AWS IoT Greengrass Core-Software wird nicht gestartet, 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)
- Fehler: container_linux.go:344: Start des Container-Prozesses verursachte „process_linux.go:424: container init caused \"rootfs_linux.go:64: mounting \\\"/greengrass/ggc/socket/greengrass_ipc.sock\\\" to rootfs \\\"/greengrass/ggc/packages/<version>/rootfs/merged\\\" at \\\"/greengrass_ipc.sock\\\" caused \\\"stat /greengrass/ggc/socket/greengrass_ipc.sock: permission denied\\\"\"“.
- 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 von öffentlichen Schlüsseldaten: ErrPrincipalNotConfigured: privater Schlüssel für MqttCertificate ist nicht festgelegt.
- Fehler: Berechtigung verweigert, wenn Sie versuchen, die Rolle arn:aws:iam::<account-id>:role/<role-name> zu verwenden, um auf s3 url https://<region>-greengrass-updates.s3.<region>.amazonaws.com/core/<architecture>/greengrass-core-<distribution-version>.tar.gz zuzugreifen.
- Der -AWS IoT GreengrassKern 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 \\\"
- [ERROR]-Laufzeitausführungsfehler: Lambda-Container konnte nicht gestartet werden. {"errorString ": "Container-Mounts konnten nicht initialisiert werden: Greengrass-Root konnte in der Overlay-Oberseite nicht maskiert werden: Maskgerät konnte nicht im Verzeichnis <ggc-path>: Datei existiert"}
- [ERROR]-Deployment fehlgeschlagen. {"deploymentId ": "<deployment-id>", "errorString": "Container-Testprozess mit pid <pid> fehlgeschlagen: Container-Prozessstatus: Beendigungsstatus 1"}
- Fehler: [FEHLER]-Laufzeitausführungsfehler: Lambda-Container konnte nicht gestartet werden. {"errorString": "failed to initialize container mounts: failed to create overlay fs for container: mounting overlay at /greengrass/ggc/packages/<ggc-version>/rootfs/merged failed: failed to mount with args source=\"no_source\" dest=\"/greengrass/ggc/packages/<ggc-version>/rootfs/merged\" fstype=\"overlay\" flags=\"0\" data=\"lowerdir=/greengrass/ggc/packages/<ggc-version>/dns:/,upperdir=/greengrass/ggc/packages/<ggc-version>/rootfs/upper,workdir=/greengrass/ggc/packages/<ggc-version>/rootfs/work\": too many levels of symbolic links"}
- 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: ds-Server konnte Socket nicht abhören: Listen unix <ggc-path>/ggc/socket/greengrass_ipc.sock: Bind: Ungültiges Argument
- [INFO] (Copier) aws.greengrass.StreamManager: stdout. Ursache: com.fasterxml.jackson.databind.JsonMappingException: Instant überschreitet den minimalen oder maximalen Instant
- GPG error: https://dnw9lb6lzp2d8.cloudfront.net stable InRelease: The following signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass Master Key

Fehler: In der Konfigurationsdatei fehlt CaPath, CertPath oder KeyPath. Der Greengrass-Daemon-Prozess mit [pid = <pid>] ist abgestürzt.

Lösung: Möglicherweise wird Ihnen diese Fehlermeldung in `crash.log` angezeigt, 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. Sie sollten stets die neueste Version der AWS IoT Greengrass Core-Software ausführen. 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 Core-Konfigurationsdatei”.

> **Note**
>
> Um festzustellen, welche Version der AWS IoT Greengrass Core-Software auf dem Core-Gerät installiert ist, führen Sie in Ihrem Geräteterminal die folgenden Befehle 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: ErrUnknownURIScheme

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. Beispielsweise:

```
...
"runtime" : {
  "cgroup" : {
    "useSystemd" : "yes"
  },
  "postStartHealthCheckTimeout" : 40000
},
...
```

Fehler: Auf lokaler Cloudwatch konnte logGroup nicht aufgerufen PutLogEvents werden: /GreengrassSystem/connection_manager, Fehler: RequestError: Anforderung fehlgeschlagen gesendet durch: Post http://<path>/cloudwatch/logs/: dial tcp <address>: getsockopt: Verbindung verweigert, Antwort: { }.

Lösung: Möglicherweise wird Ihnen diese Fehlermeldung angezeigt, wenn die AWS IoT Greengrass Core-Software nicht gestartet wird. Dies kann der Fall sein, wenn Sie AWS IoT Greengrass auf einem Raspberry Pi ausführen und die erforderliche Speichereinrichtung nicht abgeschlossen wurde. Weitere Informationen finden Sie in diesem Schritt.

Fehler: Server kann nicht erstellt werden, da: chmod /<greengrass-root>/ggc/deployment/lambda/arn:aws:lambda:<region>:<account-id>:function:<function-name>:<version>/<file-name>: keine solche Datei oder kein solches Verzeichnis vorhanden ist.

Lösung: Möglicherweise wird Ihnen diese Fehlermeldung angezeigt, wenn die AWS IoT Greengrass Core-Software nicht gestartet wird. Wenn Sie eine Lambda-ausführbare Datei im Kern 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 wird nicht gestartet, 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 1.10.2 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. Beispielsweise:

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

Der Standardwert ist `5`, und der Mindestwert ist `5`.

Fehler: container_linux.go:344: Start des Container-Prozesses verursachte „process_linux.go:424: container init caused \"rootfs_linux.go:64: mounting \\\"/greengrass/ggc/socket/greengrass_ipc.sock\\\" to rootfs \\\"/greengrass/ggc/packages/<version>/rootfs/merged\\\" at \\\"/greengrass_ipc.sock\\\" caused \\\"stat /greengrass/ggc/socket/greengrass_ipc.sock: permission denied\\\"\"“.

Lösung: Möglicherweise wird Ihnen diese Fehlermeldung in `runtime.log` angezeigt, wenn die AWS IoT Greengrass 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 AWS IoT Greengrass Berechtigungen für die Aktionen `iot:UpdateThingShadow` und `iot:GetThingShadow` in der Greengrass-Servicerolle besitzen. 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 auftreten, wenn das Dateideskriptor-Limit für die AWS IoT Greengrass Core-Software den Schwellenwert erreicht hat und erhöht werden muss.

Verwenden Sie den folgenden Befehl und starten Sie anschließend 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 AWS IoT Greengrass direkt im Stammverzeichnis oder stellen Sie sicher, dass das Verzeichnis, in dem die AWS IoT Greengrass Core-Software und ihre übergeordneten Verzeichnisse installiert sind, allen Benutzern `execute`-Berechtigungen erteilen.

Warnung: [WARN]-[5]GK Remote: Fehler beim Abrufen von öffentlichen Schlüsseldaten: ErrPrincipalNotConfigured: privater Schlüssel für MqttCertificate ist nicht festgelegt.

Lösung: AWS IoT Greengrass verwendet einen gemeinsamen Handler zum Validieren der Eigenschaften aller Sicherheitsprinzipale. 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”.

Fehler: Berechtigung verweigert, wenn Sie versuchen, die Rolle arn:aws:iam::<account-id>:role/<role-name> zu verwenden, um auf s3 url https://<region>-greengrass-updates.s3.<region>.amazonaws.com/core/<architecture>/greengrass-core-<distribution-version>.tar.gz zuzugreifen.

Lösung: Dieser Fehler wird möglicherweise angezeigt, wenn ein over-the-air (OTA)-Update fehlschlägt. Fügen Sie in der Richtlinie der Unterzeichnerrolle das Ziel AWS-Region als hinzu`Resource`. Diese Signer-Rolle wird zum Vorsignieren der S3-URL zur Aktualisierung der AWS IoT Greengrass-Software verwendet. Weitere Informationen finden Sie unter S3-URL-Signer-Rolle.

Der -AWS IoT GreengrassKern 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 Erstellen von Verbindungen verwendet werden, erhalten Sie möglicherweise auch Verbindungs-Timeout-Fehler. Stellen Sie sicher, dass Ihre Lambda-Funktionen die entsprechende Proxy-Konfiguration verwendenhttp_proxy, um eine Verbindung über den Netzwerk-Proxy herzustellen. AWS IoT Greengrass übergibt die Proxy-Konfiguration an benutzerdefinierte Lambda-Funktionen über die no_proxy Umgebungsvariablen https_proxy, und . 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). Für diese Variablen unterstützt AWS IoT Greengrass beides.

> **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 eindeutige Client-IDs verwenden. 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 stoppen:

   ```
   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. Er muss im aktuellen AWS-Region für das eindeutig seinAWS-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 in angezeigt, `runtime.log` wenn AWS IoT Greengrass versucht, 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.

[ERROR]-Laufzeitausführungsfehler: Lambda-Container konnte nicht gestartet werden. {"errorString ": "Container-Mounts konnten nicht initialisiert werden: Greengrass-Root konnte in der Overlay-Oberseite nicht maskiert werden: Maskgerät konnte nicht im Verzeichnis <ggc-path>: Datei existiert"}

Lösung: Dieser Fehler wird möglicherweise 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
- Sie müssen mit der Lambda-Funktion verbunden sein und schreibgeschützten Zugriff zulassen.

[ERROR]-Deployment fehlgeschlagen. {"deploymentId ": "<deployment-id>", "errorString": "Container-Testprozess mit pid <pid> fehlgeschlagen: Container-Prozessstatus: Beendigungsstatus 1"}

Lösung: Dieser Fehler wird möglicherweise 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 bestätigen, dass dies der Fall ist, indem Sie `GGCanary.log` nach zusätzlichen Fehlern suchen. Wenn die Lambda-Funktion nicht auf das `/usr` Verzeichnis zugreifen kann, `GGCanary.log` enthält den folgenden Fehler:

```
[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
- Sie müssen mit der Lambda-Funktion verbunden sein und schreibgeschützten Zugriff zulassen.

Fehler: [FEHLER]-Laufzeitausführungsfehler: Lambda-Container konnte nicht gestartet werden. {"errorString": "failed to initialize container mounts: failed to create overlay fs for container: mounting overlay at /greengrass/ggc/packages/<ggc-version>/rootfs/merged failed: failed to mount with args source=\"no_source\" dest=\"/greengrass/ggc/packages/<ggc-version>/rootfs/merged\" fstype=\"overlay\" flags=\"0\" data=\"lowerdir=/greengrass/ggc/packages/<ggc-version>/dns:/,upperdir=/greengrass/ggc/packages/<ggc-version>/rootfs/upper,workdir=/greengrass/ggc/packages/<ggc-version>/rootfs/work\": too many levels of symbolic links"}

Lösung: Dieser Fehler wird möglicherweise in der `runtime.log` Datei angezeigt, wenn die AWS IoT Greengrass Core-Software nicht gestartet wird. 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 dieses Problem automatisch behoben werden.
2. Wenn Sie nach dem Upgrade der AWS IoT Greengrass Core-Software immer noch diesen Fehler erhalten, 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: Dieser Fehler wird möglicherweise in Ihrer Lambda-Funktionsprotokolldatei angezeigt, wenn die Funktion `StreamManagerClient` im Funktionshandler instanziiert. Es wird empfohlen, den Client außerhalb des Handlers zu erstellen. Weitere Informationen finden Sie unter the section called “Verwenden von StreamManagerClient um mit Streams zu arbeiten”.

Fehler: ds-Server konnte Socket nicht abhören: 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 ist. Installieren Sie die AWS IoT Greengrass Core-Software in einem Ordner mit einem Dateipfad von weniger als 79 Byte, wenn Sie kein Schreibverzeichnis verwenden, oder 83 Byte, wenn Sie ein Schreibverzeichnis verwenden.

[INFO] (Copier) aws.greengrass.StreamManager: stdout. Ursache: com.fasterxml.jackson.databind.JsonMappingException: Instant überschreitet den minimalen oder maximalen Instant

Wenn Sie die AWS IoT Greengrass Kernsoftware auf v1.11.3 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 Core-Software vor v1.11.3 verwenden und 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 ausführen, auf dem Sie die AWS IoT Greengrass Core-Software 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 nicht mehr die Möglichkeit bietet, die AWS IoT Greengrass Core-Software aus dem APT-Repository zu installieren oder zu aktualisieren. Um erfolgreich auszuführen`apt update`, entfernen Sie das AWS IoT GreengrassRepository 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 derFehler: 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: Java- oder Java8-ausführbare Dateien konnten nicht gefunden werden, oder der Fehler: Bereitstellung <deployment-id> vom Typ NewDeployment für Gruppe <group-id> fehlgeschlagener Fehler: Worker mit <worker-id> konnte nicht initialisiert werden, da die Installierte Java-Version größer oder gleich 8 sein muss
- 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"}"

- Fehler: Bereitstellung <deployment-id> vom Typ NewDeployment für Gruppe <group-id> fehlgeschlagen Fehler: Fehler bei der Verarbeitung. Gruppenkonfiguration ist ungültig: 112 oder [119 0] hat keine rw-Berechtigung für die Datei: <path>.
- Fehler: <list-of-function-arns> sind für die Ausführung als Root konfiguriert, aber Greengrass ist nicht für die Ausführung von Lambda-Funktionen mit Root-Berechtigungen konfiguriert.
- Fehler: Bereitstellung <deployment-id> vom Typ NewDeployment für Gruppe <group-id> fehlgeschlagener Fehler: Greengrass-Bereitstellungsfehler: Download-Schritt bei der Bereitstellung nicht ausführen. Fehler während der Verarbeitung: Die heruntergeladene Gruppendatei konnte nicht geladen werden: konnte die UID basierend auf Benutzername nicht finden 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> vom Typ NewDeployment für Gruppe <group-id> fehlgeschlagener Fehler: Prozessstart fehlgeschlagen: container_linux.go:259: Starten des Containerprozesses verursachte „process_linux.go:250: Ausführen des exec setns-Prozesses für init verursachte \"wait: no child processes\"“.
- Fehler: [WARN]-MQTT[client] dial tcp: lookup <host-prefix>-ats.iot.<region>.amazonaws.com: kein solcher Host ... [ERROR]-Greengrass-Bereitstellungsfehler: Bereitstellungsstatus konnte nicht zurück an die Cloud gemeldet werden ... net/http: Anforderung abgebrochen, während auf Verbindung gewartet wurde (Client.Timeout überschritten, während Header gewartet wurden)

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 frühere 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 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 wählen Sie Erneut bereitstellen aus.

So stellen Sie eine Bereitstellung erneut bereit (CLI)

1. Verwenden Sie ListDeployments, um die ID der Bereitstellung zu finden, die Sie erneut bereitstellen möchten. Beispielsweise:

```
aws greengrass list-deployments --group-id 74d0b623-c2f2-4cad-9acc-ef92f61fcaf7
```

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-ef92f61fcaf7/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 Bereitstellungs-ID. Wenn Sie die Befehle ausführen, müssen Sie die Beispielwerte ersetzen.

2. Verwenden Sie CreateDeployment, um die Zielbereitstellung erneut bereitzustellen. Legen Sie den Bereitstellungstyp auf `Redeployment` fest. Beispielsweise:

```
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. Verwenden Sie 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 `"greengrass:*"` als zulässige Aktion enthält.

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 derFehler: Fehlgeschlagen: TES-Servicerolle ist nicht mit diesem Konto verknüpft.

Lösung: Möglicherweise wird Ihnen dieser Fehler angezeigt, wenn die Bereitstellung fehlschlägt. Überprüfen Sie, ob eine Greengrass-Servicerolle mit Ihrem AWS-Konto in der aktuellen verknüpft istAWS-Region. Weitere Informationen finden Sie unter 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 Gruppendefinitionsdatei:... 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 Zeitangabe haben (innerhalb von 15 Minuten), bevor sie versuchen, eine Verbindung zu AWS IoT Greengrass oder anderen TLS-Services herzustellen, die Serverzertifikate verwenden. Weitere Informationen finden Sie unter Verwenden von Gerätezeit zur Validierung von AWS IoT Serverzertifikaten im Internet der Dinge im AWS offiziellen Blog .

Die Bereitstellung wird nicht abgeschlossen.

Lösung: Führen Sie die folgenden Schritte aus:

- Stellen Sie sicher, dass der AWS IoT Greengrass-Daemon auf Ihrem Code-Gerät ausgeführt wird. Führen Sie in Ihrem Core-Geräte-Terminal die folgenden Befehle aus, um zu überprüfen, ob der Daemon ausgeführt wird, 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 in dem Pfad hängt von der AWS IoT Greengrass-Core-Softwareversion ab, die auf Ihrem Core-Gerät installiert ist.

  2. So starten Sie den Daemon:

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

- Stellen Sie sicher, dass Ihr Core-Gerät verbunden ist und die Core-Verbindungsendpunkte ordnungsgemäß konfiguriert sind.

Fehler: Java- oder Java8-ausführbare Dateien konnten nicht gefunden werden, oder der Fehler: Bereitstellung <deployment-id> vom Typ NewDeployment für Gruppe <group-id> fehlgeschlagener Fehler: Worker mit <worker-id> konnte nicht initialisiert werden, da die Installierte Java-Version größer oder gleich 8 sein muss

Lösung: Wenn der Stream-Manager für den AWS IoT Greengrass Core aktiviert ist, müssen Sie die Java-8-Laufzeit 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 IoTKonsole 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 folgende Befehle in Ihrem Core-Geräteterminal 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 Zertifikatkette 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/5YVRYtRQOxXUTEpHuEmApzGCSqGSIb3DQEK
\nCwUAhuL9MQswCQwJVUzEPMAVUzEYMBYGA1UECgwP1hem9uLmNvbSBJbmMuMRww
... content of proxy CA certificate ...
+vHIRlt0e5JAm5\noTIZGoFbK82A0/nO7f/t5PSIDAim9V3Gc3pSXxCCAQoFYnui
GaPUlGk1gCE84a0X\n7Rp/lND/PuMZ/s8YjlkY2NmYmNjMCAXDTE5MTEyN2cM216
gJMIADggEPADf2/m45hzEXAMPLE=
-----END CERTIFICATE-----

# Amazon Root CA 1
-----BEGIN CERTIFICATE-----
MIIDQTCCAimgF6AwIBAgITBmyfz/5mjAo54vB4ikPmljZKyjANJmApzyMZFo6qBg
ADA5MQswCQYDVQQGEwJVUzEPMA0tMVT8QtPHRh8jrdkGA1UEChMGDV3QQDExBBKW
... content of root CA certificate ...
o/ufQJQWUCyziar1hem9uMRkwFwYVPSHCb2XV4cdFyQzR1KldZwgJcIQ6XUDgHaa
5MsI+yMRQ+hDaXJiobldXgjUka642M4UwtBV8oK2xJNDd2ZhwLnoQdeXeGADKkpy
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, unter dem die AWS IoT Greengrass Core-Software auf Ihrem Gerät installiert ist. Normalerweise ist dies das Verzeichnis `/greengrass`.

Fehler: Bereitstellung <deployment-id> vom Typ NewDeployment für Gruppe <group-id> fehlgeschlagen Fehler: Fehler bei der Verarbeitung. Gruppenkonfiguration ist ungültig: 112 oder [119 0] hat keine rw-Berechtigung für die Datei: <path>.

Lösung: Stellen Sie sicher, dass die Besitzergruppe des Verzeichnisses *<path>* über Lese- und Schreibberechtigungen für das Verzeichnis verfügt.

Fehler: <list-of-function-arns> sind für die Ausführung als Root konfiguriert, aber Greengrass ist nicht für die Ausführung von Lambda-Funktionen mit Root-Berechtigungen konfiguriert.

Lösung: Möglicherweise wird Ihnen diese Fehlermeldung in `runtime.log` angezeigt, wenn die Bereitstellung fehlschlägt. Stellen Sie sicher, dass Sie so konfiguriert habenAWS IoT Greengrass, dass Lambda-Funktionen mit Root-Berechtigungen ausgeführt werden können. Ändern Sie entweder den Wert von `allowFunctionsToRunAsRoot` in `greengrass_root/config/config.json` `yes` oder ändern Sie die Lambda-Funktion so, dass sie als ein anderer Benutzer/eine andere Gruppe ausgeführt wird. Weitere Informationen finden Sie unter the section called “Ausführen einer Lambda-Funktion als Root”.

Fehler: Bereitstellung <deployment-id> vom Typ NewDeployment für Gruppe <group-id> fehlgeschlagener Fehler: Greengrass-Bereitstellungsfehler: Download-Schritt bei der Bereitstellung nicht ausführen. Fehler während der Verarbeitung: Die heruntergeladene Gruppendatei konnte nicht geladen werden: konnte die UID basierend auf Benutzername nicht finden 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 die `ggc_group` 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 Kerns 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
- Sie müssen mit der Lambda-Funktion verbunden sein und schreibgeschützten Zugriff zulassen.

Fehler: Bereitstellung <deployment-id> vom Typ NewDeployment für Gruppe <group-id> fehlgeschlagener Fehler: Prozessstart fehlgeschlagen: container_linux.go:259: Starten des Containerprozesses verursachte „process_linux.go:250: Ausführen des exec setns-Prozesses für init verursachte \"wait: no child processes\"“.

Lösung: Möglicherweise wird Ihnen dieser Fehler angezeigt, wenn die Bereitstellung fehlschlägt. Wiederholen Sie die Bereitstellung.

Fehler: [WARN]-MQTT[client] dial tcp: lookup <host-prefix>-ats.iot.<region>.amazonaws.com: kein solcher Host ... [ERROR]-Greengrass-Bereitstellungsfehler: Bereitstellungsstatus konnte nicht zurück an die Cloud gemeldet werden ... net/http: Anforderung abgebrochen, während auf Verbindung gewartet wurde (Client.Timeout überschritten, während Header gewartet wurden)

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. Verbindungsversuche mit dem AWS IoT Greengrass-Endpunkt können den Host nicht finden, sodass Ihre Bereitstellungen weiter den Status `In Progress` haben.

Sie können die folgenden Befehle und Ausgaben verwenden, um auf dieses Problem zu testen. Ersetzen Sie den Platzhalter für die *`Region`* in den Endpunkten durch Ihr 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 GreengrassGruppe oder Greengrass-Lambda-Funktion zu beheben.

Problembereiche

- Fehler: Ihre IsolationMode'-Konfiguration für die Gruppe ist ungültig.
- Fehler: Ihre 'IsolationMode-Konfiguration für die Funktion mit arn <function-arn> ist ungültig.
- Fehler: MemorySize Die Konfiguration für die Funktion mit arn <function-arn> ist in IsolationMode= nicht zulässigNoContainer.
- Fehler: Zugriff auf die Sysfs-Konfiguration für Funktion mit arn <function-arn> ist in IsolationMode= nicht zulässigNoContainer.
- Fehler: MemorySize Konfiguration für Funktion mit arn <function-arn> ist in IsolationMode= erforderlichGreengrassContainer.
- Fehler: Funktion <function-arn> bezieht sich auf eine Ressource vom Typ <resource-type>, die in IsolationMode= nicht zulässig istNoContainer.
- Fehler: Die Ausführungskonfiguration für die Funktion mit dem ARN <function-arn> ist nicht zulässig.

Fehler: Ihre IsolationMode'-Konfiguration 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 'IsolationMode-Konfiguration 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: MemorySize Die Konfiguration für die Funktion mit arn <function-arn> ist in IsolationMode= nicht zulässigNoContainer.

Lösung: Dieser Fehler tritt auf, wenn Sie einen `MemorySize` Wert angeben und ohne Containerisierung ausführen möchten. Lambda-Funktionen, die ohne Containerisierung ausgeführt werden, können keine Speicherlimits haben. Sie können das Limit entweder entfernen oder die Lambda-Funktion so ändern, dass sie in einem -AWS IoT GreengrassContainer ausgeführt wird.

Fehler: Zugriff auf die Sysfs-Konfiguration für Funktion mit arn <function-arn> ist in IsolationMode= nicht zulässigNoContainer.

Lösung: Dieser Fehler tritt auf, wenn Sie `true` für angeben `AccessSysfs` und ohne Containerisierung ausführen möchten. Lambda-Funktionen, die ohne Containerisierung ausgeführt werden, müssen ihren Code aktualisiert haben, um direkt auf das Dateisystem zuzugreifen, und können nicht verwenden`AccessSysfs`. Sie können entweder den Wert `false` für angeben `AccessSysfs` oder die Lambda-Funktion so ändern, dass sie in einem -AWS IoT GreengrassContainer ausgeführt wird.

Fehler: MemorySize Konfiguration für Funktion mit arn <function-arn> ist in IsolationMode= erforderlichGreengrassContainer.

Lösung: Dieser Fehler tritt auf, weil Sie kein `MemorySize` Limit für eine Lambda-Funktion angegeben haben, die Sie in einem -AWS IoT GreengrassContainer ausführen. Geben Sie einen `MemorySize`-Wert an, um den Fehler zu beheben.

Fehler: Funktion <function-arn> bezieht sich auf eine Ressource vom Typ <resource-type>, die in IsolationMode= nicht zulässig istNoContainer.

Lösung: Sie können nicht auf die `S3_Object.Generic_Archive` Ressourcentypen `Local.Device`, `Local.Volume`, `ML_Model.SageMaker.Job`, oder zugreifen`ML_Model.S3_Object`, wenn Sie eine Lambda-Funktion ohne Containerisierung ausführen. Wenn Sie diese Ressourcentypen benötigen, müssen Sie die Funktion in einem AWS IoT Greengrass-Container ausführen. Sie können auch direkt auf lokale Geräte zugreifen, wenn Sie ohne Containerisierung ausführen, 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 System-Lambda-Funktion mit `GGIPDetector` oder erstellen `GGCloudSpooler` und die - `IsolationMode` oder -`RunAs`Konfiguration angegeben haben. Sie müssen die `Execution` Parameter für diese System-Lambda-Funktion weglassen.

Erkennungsprobleme

Verwenden Sie die folgenden Informationen zur Behandlung von Problemen mit den AWS IoT Greengrass-Konnektoren.

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

- InvalidMLModelOwner GroupOwnerSetting –wird in der ML-Modellressource bereitgestellt, aber GroupOwner oder GroupPermission ist nicht vorhanden
- NoContainer Die -Funktion kann die Berechtigung nicht konfigurieren, wenn Machine Learning-Ressourcen angefügt werden. <function-arn> bezieht sich auf die Machine-Learning-Ressource <resource-id> mit der Berechtigung <ro/rw> in der Ressourcenzugriffsrichtlinie.
- Funktion <function-arn> bezieht sich auf die Machine Learning-Ressource <resource-id> mit fehlender Berechtigung sowohl in als auch in der ResourceAccessPolicy Ressource OwnerSetting.
- Funktion <function-arn> bezieht sich auf die Machine Learning-Ressource <resource-id> mit der Berechtigung \"rw\", während die Einstellung GroupPermission des Ressourcenbesitzers nur \"ro\" zulässt.
- NoContainer 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

InvalidMLModelOwner GroupOwnerSetting –wird in der ML-Modellressource bereitgestellt, aber GroupOwner oder GroupPermission ist nicht vorhanden

Lösung: Sie erhalten diesen Fehler, wenn eine Machine-Learning-Ressource das ResourceDownloadOwnerSetting Objekt enthält, die erforderliche - `GroupOwner` oder -`GroupPermission`Eigenschaft jedoch nicht definiert ist. Um dieses Problem zu beheben, definieren Sie die fehlende Eigenschaft.

NoContainer Die -Funktion kann die Berechtigung nicht konfigurieren, wenn Machine Learning-Ressourcen angefügt werden. <function-arn> bezieht sich auf die Machine-Learning-Ressource <resource-id> mit der Berechtigung <ro/rw> in der Ressourcenzugriffsrichtlinie.

Lösung: Sie erhalten diesen Fehler, wenn eine nicht containerisierte Lambda-Funktion Berechtigungen auf Funktionsebene für eine Machine-Learning-Ressource angibt. Nicht containerisierte Funktionen müssen Berechtigungen von den Ressourcenbesitzer-Berechtigungen erben, die für die Machine Learning-Ressource definiert sind. Um dieses Problem zu beheben, wählen Sie , um Berechtigungen des Ressourcenbesitzers zu erben (Konsole) oder entfernen Sie die Berechtigungen aus der Ressourcenzugriffsrichtlinie (API) der Lambda-Funktion.

Funktion <function-arn> bezieht sich auf die Machine Learning-Ressource <resource-id> mit fehlender Berechtigung sowohl in als auch in der ResourceAccessPolicy Ressource OwnerSetting.

Lösung: Sie erhalten diesen Fehler, wenn Berechtigungen für die Machine-Learning-Ressource nicht für die angehängte Lambda-Funktion oder die Ressource konfiguriert sind. Um dieses Problem zu beheben, konfigurieren Sie Berechtigungen in der -ResourceAccessPolicyEigenschaft für die Lambda-Funktion oder der -OwnerSettingEigenschaft für die -Ressource.

Funktion <function-arn> bezieht sich auf die Machine Learning-Ressource <resource-id> mit der Berechtigung \"rw\", während die Einstellung GroupPermission des Ressourcenbesitzers nur \"ro\" zulässt.

Lösung: Sie erhalten diesen Fehler, wenn die für die angefügte Lambda-Funktion definierten Zugriffsberechtigungen die für die Machine-Learning-Ressource 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 Funktion <function-arn> bezieht sich auf Ressourcen des verschachtelten Zielpfads.

Lösung: Sie erhalten diesen Fehler, wenn mehrere Machine-Learning-Ressourcen, die einer nicht containerisierten Lambda-Funktion zugeordnet 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 in , `runtime.log` wenn dieselbe Betriebssystemgruppe als „Als Identität ausführen“ 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 erteilt 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 fügen Sie die Machine-Learning-Ressource an die Lambda-Funktion an.

AWS IoT Greengrass Core-Probleme in Docker

Verwenden Sie die folgenden Informationen, um Probleme beim Ausführen eines AWS IoT Greengrass-Cores 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 (AccessDeniedException) aufgetreten: Benutzer: arn:aws:iam::<account-id>:user/<user-name> ist nicht berechtigt: ecr:GetAuthorizationToken on Ressource: *
- 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-Namespace 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. Stellen Sie sicher, dass die neueste AWS CLI-Version (z. B. run: `pip install awscli --upgrade --user`) installiert ist. Wenn Sie Windows verwenden und die CLI mit dem MSI-Installationsprogramm installiert haben, müssen Sie den Installationsvorgang wiederholen. Weitere Informationen finden Sie unter Installieren der AWS Command Line Interface unter Microsoft Windows im AWS Command Line Interface -Benutzerhandbuch.

Warnung: IPv4 ist deaktiviert. Das Netzwerk wird nicht funktionieren.

Lösung: Sie können diese Warnung oder eine ähnliche Nachricht erhalten, wenn Sie AWS IoT Greengrass auf einem Linux-Computer ausführen. Aktivieren Sie die IPv4-Netzwerkweiterleitung wie in diesem Schritt beschrieben. Die AWS IoT Greengrass-Cloud Bereitstellung und MQTT-Kommunikation sind nicht funktionsfähig, 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 (AccessDeniedException) aufgetreten: Benutzer: arn:aws:iam::<account-id>:user/<user-name> ist nicht berechtigt: ecr:GetAuthorizationToken on Ressource: *

Dieser Fehler wird möglicherweise angezeigt, 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 in `runtime.log` kann auftreten, wenn Sie versuchen, eine `GreengrassContainer` Lambda-Funktion auf einem AWS IoT Greengrass-Core 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, dürfen Sie das vorhandene `deployment` Verzeichnis nicht auf dem Docker-AWS IoT GreengrassCore-Container binden. Erstellen Sie stattdessen ein leeres `deployment`-Verzeichnis an seiner Stelle und fügen Sie es über Bind-Mount im Docker-Container ein. Auf diese Weise 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 “Ausführen von AWS IoT Greengrass in einem Docker-Container”.

Fehlerbehebung mit Protokollen

Sie können Protokollierungseinstellungen für eine Greengrass-Gruppe konfigurieren, z. B. ob Protokolle an CloudWatch Protokolle gesendet, Protokolle auf dem lokalen Dateisystem gespeichert 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 der Protokollierung für AWS IoT Greengrass”.

Auf dem lokalen Dateisystem speichert AWS IoT Greengrass Protokolle an den folgenden Speicherorten. Das Lesen der Protokolle auf dem Dateisystem erfordert Root-Berechtigungen.

greengrass-root`/ggc/var/log/crash.log`

Zeigt Nachrichten an, die bei einem Absturz eines AWS IoT Greengrass-Cores generiert werden.

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 zum Beispiel vom Certificate Manager und vom Connection Manager. Mit den Meldungen in `ggc/var/log/system/` und `ggc/var/log/system/runtime.log`, sollten Sie ermitteln können, 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 CloudWatch Protokolle übernimmt. Wenn Sie Greengrass-Protokolle in nicht anzeigen können CloudWatch, können Sie diese Protokolle zur Fehlerbehebung verwenden.

greengrass-root/ggc/var/log/user/

Enthält alle Protokolle von benutzerdefinierten Lambda-Funktionen. Überprüfen Sie diesen Ordner, um Fehlermeldungen von Ihren lokalen Lambda-Funktionen zu finden.

> **Note**
>
> Standardmäßig ist *greengrass-root* das /greengrass-Verzeichnis. 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 -Protokolle, um Protokollmeldungen anzuzeigen. crash.log befindet sich nur in Dateisystemprotokollen auf dem AWS IoT Greengrass Core-Gerät.

Wenn so konfiguriert AWS IoT ist, dass Protokolle in geschrieben werden CloudWatch, überprüfen Sie diese Protokolle, wenn Verbindungsfehler auftreten, wenn Systemkomponenten versuchen, eine Verbindung zu herzustellenAWS IoT.

Weitere Informationen zur AWS IoT Greengrass-Protokollierung finden Sie unter the section called “Überwachen mit AWS IoT Greengrass-Protokollen”.

> **Note**
>
> Protokolle für AWS IoT Greengrass Core-Software v1.0 werden im Verzeichnis *greengrass-root*/var/log gespeichert.

Fehlerbehebung bei Speicherproblemen

Wenn der lokale Dateispeicher voll ist, schlagen einige Komponenten möglicherweise fehl:

- Es werden keine Updates von lokalen Schatten ausgeführt.
- Neue MQTT-AWS IoT GreengrassKernserverzertifikate können nicht lokal heruntergeladen werden.
- Die Bereitstellung schlägt fehl.

Sie sollten immer wissen, wie viel lokaler Speicherplatz verfügbar ist. Sie können freien Speicherplatz basierend auf 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 in AWS IoT Greengrass werden mit QoS 0 gesendet. Standardmäßig speichert AWS IoT Greengrass Nachrichten in einer speicherinternen Warteschlange. Daher gehen nicht verarbeitete Nachrichten verloren, wenn der Greengrass-Core neu gestartet wird (z. B. nach einer Gruppenbereitstellung oder einem Geräteneustart). Sie können jedoch AWS IoT Greengrass (v1.6 oder höher) so konfigurieren, dass Nachrichten im Dateisystem zwischengespeichert werden, sodass sie über Kernneustarts hinweg 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 AWS IoT Greengrass möglicherweise nicht ordnungsgemäß 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 Kern Nachrichten empfangen, die vom gesendet werden, AWS Cloud während der Kern 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. Beispielsweise:

```
{
  "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 AWS IoT Greengrass Core Software bis Version 1.6 beträgt der `shadowSyncTimeout`-Standardwert 1 Sekunde.

AWS re:Post prüfen

Wenn Sie Ihr Problem nicht mithilfe der Informationen zur Fehlerbehebung in diesem Thema beheben können, können Sie das AWS IoT Greengrass-Tag auf AWS re:Post nach verwandten Problemen durchsuchen Fehlerbehebung oder eine neue Frage stellen. Mitglieder des AWS IoT Greengrass Teams überwachen AWS re:Post aktiv.

Dokumentverlauf für AWS IoT Greengrass

In der folgenden Tabelle werden wichtige Änderungen am - AWS IoT Greengrass Entwicklerhandbuch nach Juni 2018 beschrieben. Um Benachrichtigungen über Aktualisierungen dieser Dokumentation zu erhalten, können Sie einen RSS-Feed abonnieren.

Änderung	Beschreibung	Datum
Aktualisierung auf das Ende der Unterstützung für v1.11.x Snap	Das Ende der Support-Informationen für AWS IoT Greengrass Core v 1.11.x Snap auf snapcraft.io wurde aktualisiert.	22. September 2023
Ende der Unterstützung für v1.11.x Snap	Ende der Support-Informationen für AWS IoT Greengrass Core v 1.11.x Snap auf snapcraft.io hinzugefügt.	19. September 2023
Docker-Images für AWS IoT Greengrass v1.11.6	Die Docker-Images für AWS IoT Greengrass Core-Software v1.11.6 sind in Amazon Elastic Container Registry (Amazon ECR) und Docker Hub verfügbar. Wir empfehlen Ihnen, immer die neueste Version auszuführen.	12. April 2022
AWS IoT Device Tester (IDT) zur AWS IoT Greengrass V1 Veralterung	IDT für AWS IoT Greengrass V1 generiert keine signierten Qualifizierungsberichte 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 Core-Soft	24. März 2022

	wareversion v1.11.6 für die Gerätequalifizierung.	
AWS IoT Greengrass Version 1.11.6 veröffentlicht	Version 1.11.6 der - AWS IoT Greengrass Core-Software ist verfügbar. Diese Version enthält Leistungsverbesser ungen und Fehlerbehebungen. Wir empfehlen Ihnen, immer die neueste Version auszuführ en.	24. März 2022
IoT SiteWise -Konnektor-Version 12 veröffentlicht	Version 12 des IoT SiteWise -Konnektors ist verfügbar . Diese Version enthält Fehlerbehebungen.	23. Dezember 2021
Docker-Images für AWS IoT Greengrass v1.11.5 und v1.10.5	Die Docker-Images für AWS IoT Greengrass Core-Soft ware v1.11.5 und v1.10.5 sind in Amazon Elastic Container Registry (Amazon ECR) und Docker Hub verfügbar. Wir empfehlen Ihnen, immer die neueste Version auszuführen.	22. Dezember 2021
AWS IoT Greengrass V1 - Wartungsrichtlinie	Die AWS IoT Greengrass V1 Wartungsrichtlinie definiert die verschiedenen Wartungs- und Aktualisierungsstufen für den AWS IoT Greengras s 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 Qualifizierungssuite (CCPQ) v1.3.1 und unterstützt die Verwendung der AWS IoT Greengrass Core-Softwareversionen 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 AWS IoT Greengrass -Core-Software sind verfügbar. Diese Versionen enthalten Leistungsverbesserungen und Fehlerbehebungen. Wir empfehlen Ihnen, immer die neueste Version auszuführen.	12. Dezember 2021

Erneut veröffentlichte AWS IoT Greengrass Docker-Images v1.11.4 und v1.10.4	Docker-Images für die AWS IoT Greengrass Core-Softwareversionen 1.11.4 und 1.10.4 wurden in Amazon Elastic Container Registry (Amazon ECR) und Docker Hub erneut veröffentlicht, um Fehlerbehebungen mit zu beheben BusyBox. Um die neuesten Docker-Images zu verwenden, verwenden Sie die Tags `1.11.4-1` oder `1.10.4-1` . Weitere Informationen zu verfügbaren Tags finden Sie unter amazon/aws-iot-greengrass in Docker Hub .	8. Dezember 2021
CloudWatch Der Metrik-Konnektor 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 `aws:SourceAccount` globalen Bedingungskontextschlüssel `aws:SourceArn` und in IAM-Ressourcenrichtlinien, um das Confused-Deputy-Problem zu vermeiden .	1. November 2021

Docker-Images für AWS IoT Greengrass v1.11.4	Die Docker-Images für AWS IoT Greengrass Core-Software v1.11.4 sind in Amazon Elastic Container Registry (Amazon ECR) und Docker Hub verfügbar. Wir empfehlen Ihnen, immer die neueste Version auszuführen.	24. August 2021
Veröffentlicht AWS IoT Greengrass v1.11.4 Snap	Version 1.11.4 des AWS IoT Greengrass Snap ist verfügbar. Wir empfehlen Ihnen, immer die neueste Version auszuführen.	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 Core-Softwareversion 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 Stream Manager, das Upgrades auf v1.11.3 von einer früheren Version der AWS IoT Greengrass Core-Software verhinderte. Wir empfehlen Ihnen, immer die neueste Version auszuführen.	17. August 2021

VPC-Endpunkte (AWS PrivateLink)	AWS IoT Greengrass unterstützt jetzt Schnittstellen-VPC-Endpunkte (AWS PrivateLink) für die AWS IoT Greengrass Steuerebene. Sie können eine private Verbindung zwischen Ihrer VPC und der AWS IoT Greengrass Steuerebene 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 Core-Softwareversionen 1.11.3 und 1.10.4 für die Gerätequalifizierung.	23. Juni 2021
Veröffentlicht AWS IoT Greengrass v1.11.3 Snap	Version 1.11.3 von AWS IoT Greengrass Snap enthält Leistungsverbesserungen und Fehlerbehebungen. Als bewährte Methode empfehlen wir, immer die neueste Version auszuführen.	15. Juni 2021

Docker-Images für AWS IoT Greengrass v1.11.3 und v1.10.4 veröffentlicht	Die Docker-Images für 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 Fehlerbehebungen. Wir empfehlen Ihnen, immer die neueste Version auszuführen.	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 Ihnen, immer die neueste Version auszuführen.	14. Juni 2021
AWS IoT Greengrass Version 1.10.4 veröffentlicht	Version 1.10.4 der - AWS IoT Greengrass Core-Software ist verfügbar. Diese Version enthält Leistungsverbesserungen und Fehlerbehebungen. Wir empfehlen Ihnen, immer die neueste Version auszuführen.	14. Juni 2021

Bol-TCP Protocol Adapter Version 2 veröffentlicht	Version 2 des microSD-TCP Protocol Adapter-Konnektors ist verfügbar. Diese Version hat Unterstützung für ASCII-, UTF8- und ISO8859-kodierte Quellzeichenfolgen hinzugefügt.	24. Mai 2021
Docker-Anwendungsbereitstellungs-Connector Version 7 veröffentlicht	Version 7 des Greengrass-Docker-Anwendungsbereitstellungs-Connectors ist verfügbar.	05. April 2021
AWS IoT Greengrass Version 1.11.1 veröffentlicht	Version 1.11.1 der - AWS IoT Greengrass Core-Software ist verfügbar. Diese Version enthält Leistungsverbesserungen und Fehlerbehebungen. Wir empfehlen Ihnen, immer die neueste Version auszuführen.	29. März 2021
AWS IoT Device Tester Version 4.0.2 veröffentlicht	Version 4.0.2 von AWS IoT Device Tester für AWS IoT Greengrass ist verfügbar. Diese Version ersetzt IDT v4.0.0 und fügt Unterstützung für Version 1.11.1 der - AWS IoT Greengrass Core-Software hinzu. Dies behebt auch ein Problem, das dazu führte, dass IDT Hardware Security Integration (HSI)-Fehler maskierte.	29. März 2021

IoT SiteWise -Konnektor-Version 11 veröffentlicht	Version 11 des IoT SiteWise -Konnektors ist verfügbar. Dadurch wird die Unterstützung für Zeichenfolgen gestartet, die versteckte oder nicht darstellbare Zeichen enthalten. Diese Version enthält auch allgemeine Leistungsverbesserungen und Fehlerbehebungen.	24. März 2021
Erneut veröffentlicht AWS IoT Greengrass v1.11.0 Snap	AWS IoT Greengrass Snap-Version 1.11.0 wurde auf Snapcraft erneut veröffentlicht, um Fehlerbehebungen und einen möglichen Anwendungsabsturz bei Verwendung des Python-Interpreters zu beheben. AWS IoT Greengrass stellt keine Snaps für die Softwareversionen 1.10 und 1.9 bereit.	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 Core-Softwareversion v1.10.3 für die Gerätequalifizierung.	18. März 2021

Erneut veröffentlicht AWS IoT Greengrass v1.8.4 Snap	AWS IoT Greengrass Snap-Version 1.8.4 wurde auf Snapcraft erneut veröffentlicht, um Fehlerbehebungen und einen möglichen Anwendungsabsturz bei Verwendung des Python-Interpreters zu beheben.	15. März 2021
Erneut veröffentlichtes AWS IoT Greengrass v1.11.0 Docker-Image für Armv7l	Das Docker-Image für 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 Fehlerbehebungen und einen möglichen Anwendungsabsturz bei Verwendung des Python-Interpreters zu beheben.	8. März 2021
AWS IoT Greengrass v1.10.3 Docker-Images veröffentlicht	Docker-Images für AWS IoT Greengrass Core-Softwareversion 1.10.3 sind jetzt auf Amazon Elastic Container Registry (Amazon ECR) und Docker Hub verfügbar.	8. März 2021

Erneut veröffentlichte AWS IoT Greengrass -v1.11.0- und -v1.9.4-Docker-Images	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 Fehlerbehebungen und einen möglichen Anwendungsabsturz bei 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 Ihnen, immer die neueste Version auszuführen.	24. Februar 2021

IoT SiteWise -Konnektor-Version 10 veröffentlicht	Version 10 des IoT SiteWise -Konnektors ist verfügbar . Diese Version behebt Stabilitätsprobleme mit dem StreamManager Client, wenn die Verbindung unterbroc hen wird, und verbessert die Handhabung von OPC-UA-Werten, wenn ein `SourceTim estamp` fehlt. Verwenden Sie den IoT SiteWise -Konnekto r, um lokale Geräte- und Gerätedaten an Komponent eneigenschaften in IoT SiteWise zu senden.	22. Januar 2021
IoT SiteWise -Konnektor Version 9 veröffentlicht	Version 9 des IoT SiteWise -Konnektors ist verfügbar. Dadurch wird die Unterstüt zung für benutzerdefinierte Greengrass-`StreamMan ager` Stream-Ziele, OPC-UA-Deadbanding, benutzerd efinierten Scanmodus und benutzerdefinierte Scanrate gestartet. Dies beinhaltet auch eine verbesserte Leistung bei Konfigurationsaktualisierun gen, die vom IoT SiteWise -Gateway durchgeführt wurden. Verwenden Sie den IoT SiteWise -Konnektor, um lokale Geräte- und Gerätedat en an Komponenteneigensc haften in IoT SiteWise zu senden.	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 von AWS IoT Greengrass Snap unterstützt nicht containerisierte Lambda-Funktionen. Als bewährte Methode empfehlen wir, immer die neueste Version auszuführen.	6. Dezember 2020
IoT SiteWise -Konnektor Version 8 veröffentlicht	Version 8 des IoT SiteWise -Konnektors ist verfügbar. Diese Version verbessert die Stabilität, wenn der Konnektor zeitweilige Netzwerkkonnektivität aufweist. Verwenden Sie den IoT SiteWise -Konnektor, um lokale Geräte- und Gerätedaten an Komponenteneigenschaften in IoT SiteWise zu senden.	19. November 2020
Der Kinesis-Firehose-Konnektor unterstützt den Modus Kein Container	Sie können den `IsolationMode` Parameter verwenden, um den Containerisierungsmodus für den Konnektor zu konfigurieren.	19. Oktober 2020

Docker-Anwendungsbereitstellungs-Connector Version 6 veröffentlicht	Version 6 des Greengrass Docker-Anwendungsbereitstellungs-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 Funktion Systemzustandtelemetrie und eine lokale Zustandsprüfungs-API hinzu. Stream Manager kann jetzt Daten nach Amazon Simple Storage Service (Amazon S3) und IoT SiteWise exportieren. Diese Version enthält auch Leistungsverbesserungen und Fehlerbehebungen. Wir empfehlen Ihnen, immer die neueste Version auszuführen.	16. September 2020
IoT SiteWise -Konnektor Version 7 veröffentlicht	Version 7 des IoT SiteWise -Konnektors ist verfügbar . Diese Version behebt ein Problem mit Gateway-Metriken. Verwenden Sie den IoT SiteWise -Konnektor, um lokale Geräte- und Gerätedaten an Komponenteneigenschaften in IoT SiteWise zu senden.	14. August 2020
ServiceNow MetricBase Konnektoren für Integration, Splunk-Integration und Twilio Notifications unterstützen den Modus Kein Container	Sie können den `Isolation Mode` Parameter verwenden , um den Containerisierungsmodus für den Konnektor zu konfigurieren.	30. Juli 2020

Der SNS-Konnektor 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 Metrik-Konnektor 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-Konfigurationseigenschaft hinzu und enthält Leistungsverbesserungen und Fehlerbehebungen. Wir empfehlen Ihnen, immer die neueste Version auszuführen.	08. Juni 2020
Tensorflow-Installationsprogramme für Machine Learning sind veraltet	AWS IoT Greengrass Tensorflow vorgefertigte Machine Learning-Installationsprogramme sind veraltet. Machine-Learning-Beispiele wurden auf Python 3.7 aktualisiert.	29. Mai 2020

Unterstützung für Chainer-Frameworks und Greengrass-Installationsprogramme für Machine Learning sind veraltet	AWS IoT Greengrass Vorgefertigte Installationsprogrammen für Machine Learning und Downloads für MXNet und DLR sind veraltet. Unterstützung für Chainer-Framework und zugehörige Downloads ist veraltet.	4. Mai 2020
IoT SiteWise -Konnektor Version 6 veröffentlicht	Version 6 des IoT SiteWise -Konnektors 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 Konnektors erfordert Stream Manager und AWS IoT Greengrass Core-Software v1.10.0 oder höher. Verwenden Sie den IoT SiteWise -Konnektor, um lokale Geräte- und Gerätedaten an Komponenteneigenschaften in IoT SiteWise zu senden.	29. April 2020
Konnektoren, die auf Python 3.7 aktualisiert wurden	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

Die Einrichtung des Greengrass-Geräts kann im Hintergrund ausgeführt werden	Sie können Greengrass Device Setup im Hintergrundmodus ausführen, damit das Skript Sie nicht zur Eingabe von Werten auffordert.	27. April 2020
Neue Docker-Basis-Images	Sie können AWS IoT Greengrass Docker-Images herunterladen, die auf Alpine Linux (x86_64, Armv7l oderAArch64)-Basis-Images basieren.	23. April 2020
AWS IoT Greengrass Version 1.10.1 veröffentlicht	Version 1.10.1 der - AWS IoT Greengrass Core-Software ist verfügbar. Diese Version enthält Leistungsverbesserungen und Fehlerbehebungen. Wir empfehlen Ihnen, immer die neueste Version auszuführen.	16. April 2020
Neues Kapitel bezüglich der Sicherheit	AWS IoT Greengrass - Sicherheitsinhalte wurden neu organisiert und neue Informationen hinzugefügt.	30. März 2020
Verwenden von APT-Paketmanager zur Installation von AWS IoT Greengrass	Auf unterstützten Debian-basierten Linux-Distributionen können Sie verwenden, `apt` um die - AWS IoT Greengrass Core-Software auf Ihren Geräten zu installieren.	26. Februar 2020

IoT SiteWise -Konnektor Version 5 veröffentlicht	Version 5 des IoT SiteWise -Konnektors ist verfügbar . Diese Version behebt ein Kompatibilitätsproblem mit AWS IoT Greengras s Core-Software v1.9.4. Verwenden Sie den IoT SiteWise -Konnektor, um lokale Geräte- und Gerätedat en an Komponenteneigensc haften in IoT SiteWise zu senden.	12. Februar 2020
Neues Skript zur schnellen Einrichtung eines Core-Geräts	Mit Greengrass Device Setup lässt sich Ihr Core-Gerät in wenigen Minuten konfiguri eren. Außerdem unterstützt AWS IoT Greengrass jetzt die Lambda-Funktionen 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 Features in dieser Version gehören Stream Manager, Container-Unterstützung mit dem Docker-Anwendungsbereitstellungs-Konnektor, nicht containerisierte Lambda-Funktionen, die auf Machine-Learning-Ressourcen zugreifen können, Unterstützung für persistente MQTT-Sitzungen mit AWS IoTund Unterstützung für lokalen MQTT-Datenverkehr über einen bestimmten Port.	25. November 2019
Konsolenunterstützung für Bereitstellungsbenachrichtigungen	Verwenden Sie die Amazon-EventBridge Konsole, um Ereignisregeln zu erstellen, die ausgelöst werden, wenn sich Ihre Greengrass-Gruppenbereitstellungen den Status ändern.	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	Sie können Tags für Ihre Greengrass-Gruppen in der AWS IoT Konsole erstellen, anzeigen und verwalten.	23. September 2019
Neue Konnektoren für Machine Learning	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. Mit dieser Version können Sie die - AWS IoT Greengrass Core-Software auf Raspbian-Verteilungen auf Armv6l-Architekturen installieren, OTA-Updates auf Port 443 mit ALPN unterstützen und einen Bugfix für binäre Nutzlasten enthalten, die von Python-2.7-Lambda-Funktionen an andere Lambda-Funktionen gesendet werden.	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 Leistungsverbesser ungen 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 mit Unterstützung für veröffentlicht 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 Verteilungen 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-Soft ware 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 ein Platzhalterzeichen im Thema 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: Unterstützung 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-Verfügbar auf Snapmaker	Verwenden Sie die AWS IoT Greengrass Snap Store-App , um Software mit schnell auf Linux-Geräten zu entwerfen, zu testen und bereitzustellen AWS IoT Greengrass.	1. April 2019
Unterstützung 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 steuern.	29. März 2019
IoT Analytics Connector veröffentlicht	Verwenden Sie den IoT Analytics-Konnektor, um lokale Gerätedaten an Kanäle zu AWS IoT Analytics senden.	15. März 2019
Batch-Unterstützung im Kinesis-Firehose-Konnektor	Der Kinesis-Firehose-Konnektor unterstützt das Senden von Datensätzen im Stapel an Amazon Data Firehose in einem bestimmten Intervall.	15. März 2019

AWS CloudFormation - Unterstützung für - AWS IoT Greengrass Ressourcen	Verwenden Sie - AWS 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: Konfiguri erbare Standardzugriffsid entität für Lambda-Funktionen, Unterstützung für HTTPS-Dat enverkehr über Port 443 und vorhersehbar benannte 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 Neo-Deep-Learning-Laufzeit	Die SageMaker Neo-Deep-Learning-Laufzeit unterstützt Machine-Learning-Modelle, die vom SageMaker 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, lokaler Secrets-Manager, Isolations- und Berechtigungseinstellungen für Lambda-Funktionen, Hardware-Root-of-Trust-Sicherheit, Verbindung mit ALPN oder Netzwerkproxy und Raspbian-Travel-Unterstützung.	26. November 2018
AWS IoT Greengrass - Software-Downloads	Die Pakete AWS IoT Greengrass Core-Software, AWS IoT Greengrass Core SDK und AWS IoT Greengrass Machine Learning SDK sind für dowload über Amazon verfügbar CloudFront.	26. November 2018
AWS IoT Device Tester für AWS IoT Greengrass	Verwenden Sie AWS IoT Device Tester für , AWS IoT Greengrass um zu überprüfen, ob Ihre CPU-Architektur, Kernel-Konfiguration und Treiber mit funktionieren AWS IoT Greengrass.	26. November 2018

AWS CloudTrail -Protokollierung für AWS IoT Greengrass API-Aufrufe	AWS IoT Greengrass ist in integriert, einem Service AWS CloudTrail, der die Aktionen eines Benutzers, einer Rolle oder eines - AWS Services in aufzeichnet AWS IoT Greengrass.	29. Oktober 2018
Unterstützung für TensorFlow v1.10.1 auf NVIDIA Jetson TX2	Die TensorFlow vorkompilierte Bibliothek für NVIDIA Jetson TX2, die AWS IoT Greengrass bereitstellt, verwendet jetzt TensorFlow v1.10.1. Dadurch werden Jetpack 3.3 und CUDA Toolkit 9.0 unterstützt.	18. Oktober 2018
Unterstützung für MXNet v1.2.1 Machine Learning-Ressourcen	AWS IoT Greengrass unterstützt Machine-Learning-Modelle, die mit MXNet v1.2.1 trainiert wurden.	29. August 2018
AWS IoT Greengrass Version 1.6.0 veröffentlicht	Neue Features: Lambda-ausführbare Dateien, konfigurierbare Nachrichtenwarteschlange, konfigurierbares Wiederholungsintervall für Wiederverbindung, Volume-Ressourcen unter /proc und konfigurierbares Schreibverzeichnis.	26. Juli 2018

Frühere Aktualisierungen

In der folgenden Tabelle werden wichtige Änderungen am - AWS IoT Greengrass Entwicklerhandbuch vor Juli 2018 beschrieben.

Änderung	Beschreibung	Datum
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. • Greengrass-Lambda-Funktionen unterstützen zusätzlich zu JSON 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: • O ver-the-air (OTA) Update Agent, der in der Lage ist, von der Cloud bereitgestellte Greengrass-Aktualisierungsaufträge zu verarbeiten. Weitere Informationen finden Sie unter OTA-Updates der AWS IoT Greengrass Core-Software. • Greifen Sie über Greengrass-Lambda-Funktionen auf lokale Telefonie 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 bereitgestellte AWS IoT Greengrass Gruppen zurück. Weitere Informationen finden Sie unter Zurücksetzen von Bereitstellungen. • Unterstützung für Node.js 6.10- und Java 8-Lambda-Laufzeiten, zusätzlich zu Python 2.7.	20. September 2017
AWS IoT Greengrass Version 1.0.0 veröffentlicht	AWS IoT Greengrass ist allgemein verfügbar.	7. Juni 2017