Entwicklerhandbuch, Version 2

AWS IoT Greengrass

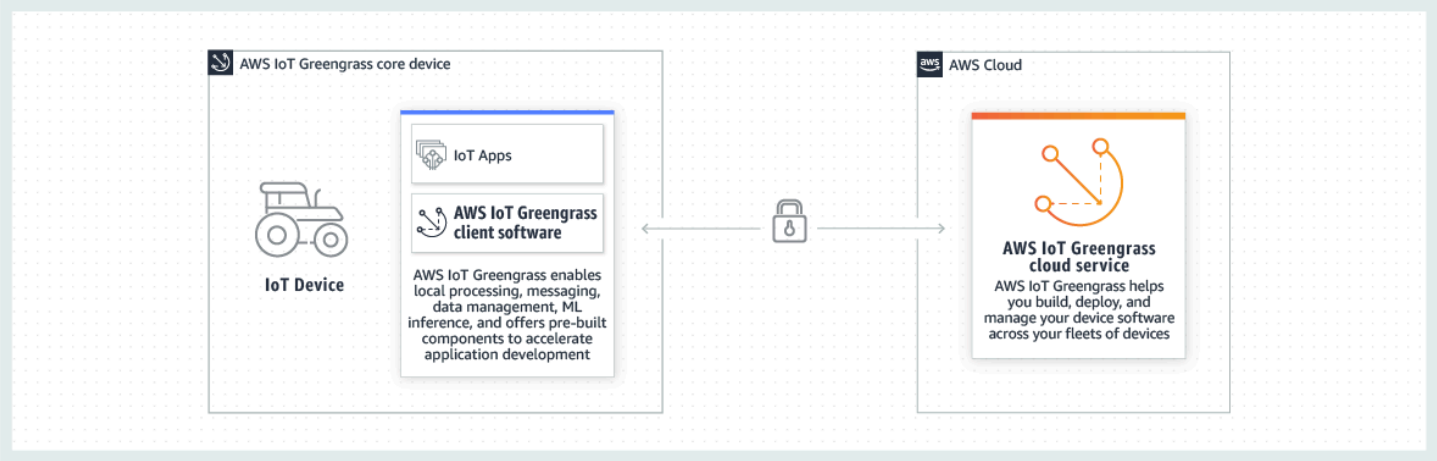
AWS IoT Greengrass: Entwicklerhandbuch, Version 2

Table of Contents

Was ist AWS IoT Greengrass?

AWS IoT Greengrass ist ein Open-Source-Edge-Laufzeit- und Cloud-Service für das Internet der Dinge (IoT), mit dem Sie IoT-Anwendungen auf Ihren Geräten erstellen, bereitstellen und verwalten können. Sie können verwendenAWS IoT Greengrass, um Software zu erstellen, mit der Ihre Geräte lokal auf die von ihnen generierten Daten reagieren können, Vorhersagen auf der Grundlage von Machine-Learning-Modellen ausführen und Gerätedaten filtern und aggregieren können. AWS IoT Greengrass ermöglicht es Ihren Geräten, Daten näher an dem Ort zu sammeln und zu analysieren, an dem diese Daten generiert werden, eigenständig auf lokale Ereignisse zu reagieren und sicher mit anderen Geräten im lokalen Netzwerk zu kommunizieren. Greengrass-Geräte können auch sicher mit kommunizieren AWS IoT Core und IoT-Daten in die exportierenAWS Cloud. Sie können AWS IoT Greengrass nutzen, um Edge-Anwendungen mithilfe von vorgefertigten Softwaremodulen, sogenannten Komponenten, zu erstellen, die Ihre Edge-Geräte mit AWS-Diensten oder Diensten von Drittanbietern verbinden können. Sie können auch verwenden, AWS IoT Greengrass um Ihre Software mithilfe von Lambda-Funktionen, Docker-Containern, nativen Betriebssystemprozessen oder benutzerdefinierten Laufzeiten Ihrer Wahl zu verpacken und auszuführen.

Das folgende Beispiel zeigt, wie ein -AWS IoT GreengrassGerät mit der interagiertAWS Cloud.



Neue Features

AWS IoT Greengrass V2 führt neue Features und Verbesserungen ein. Im Folgenden finden Sie weitere Informationen zu den neuen Funktionen, die in Version 2 angeboten werden.

- Was ist neu in AWS IoT Greengrass Version 2

Für Erstbenutzer von AWS IoT Greengrass

Wenn Sie noch nicht mit vertraut sindAWS IoT Greengrass, empfehlen wir Ihnen, den folgenden Abschnitt zu lesen:

- Funktionsweise von AWS IoT Greengrass

Folgen Sie als Nächstes dem Tutorial „Erste Schritte“, um die grundlegenden Funktionen von auszuprobierenAWS IoT Greengrass. In diesem Tutorial installieren Sie die -AWS IoT GreengrassCore-Software auf einem Gerät, entwickeln eine Hello-World-Komponente und verpacken diese Komponente für die Bereitstellung.

Für bestehende Benutzer von AWS IoT Greengrass

Für aktuelle Benutzer von empfehlen wir die folgenden ThemenAWS IoT Greengrass V1, die Ihnen helfen, die Unterschiede zwischen Greengrass Version 1 und Greengrass Version 2 zu verstehen und zu erfahren, wie Sie von Version 1 zu Version 2 wechseln:

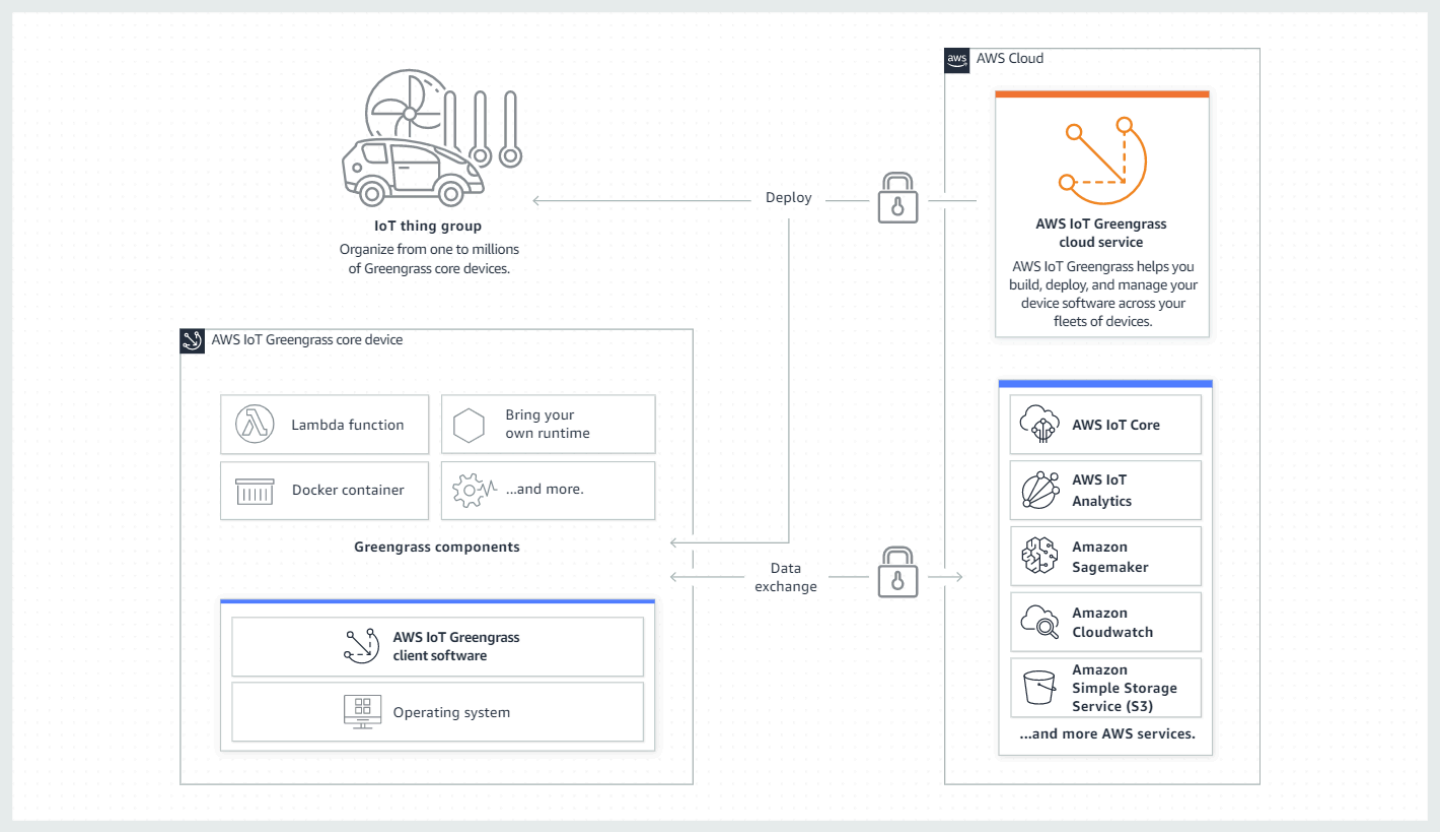
- Migrieren von AWS IoT Greengrass Version 1

Funktionsweise von AWS IoT Greengrass

Die AWS IoT Greengrass Clientsoftware, auch AWS IoT Greengrass Core-Software genannt, wird auf Windows- und Linux-basierten Distributionen wie Ubuntu oder Raspberry Pi OS für Geräte mit ARM- oder x86-Architekturen ausgeführt. Mit können Sie Geräte so programmierenAWS IoT Greengrass, dass sie lokal auf die Daten reagieren, die sie generieren, Vorhersagen basierend auf Machine-Learning-Modellen ausführen und Gerätedaten filtern und aggregieren. AWS IoT Greengrass ermöglicht die lokale Ausführung von AWS Lambda Funktionen, Docker-Containern, nativen Betriebssystemprozessen oder benutzerdefinierten Laufzeiten Ihrer Wahl.

AWS IoT Greengrass bietet vorgefertigte Softwaremodule, sogenannte Komponenten, mit denen Sie die Funktionalität von Edge-Geräten einfach erweitern können. -AWS IoT GreengrassKomponenten ermöglichen es Ihnen, eine Verbindung zu -AWSServices und Drittanbieteranwendungen am Edge herzustellen. Nachdem Sie Ihre IoT-Anwendungen entwickelt haben, AWS IoT Greengrass ermöglicht es Ihnen, diese Anwendungen auf Ihrer Geräteflotte im -Feld remote bereitzustellen, zu konfigurieren und zu verwalten.

Das folgende Beispiel zeigt, wie ein -AWS IoT GreengrassGerät mit dem AWS IoT Greengrass Cloud-Service und anderen -AWSServices in der interagiertAWS Cloud.



Schlüsselkonzepte für AWS IoT Greengrass

Im Folgenden finden Sie grundlegende Konzepte für das Verständnis und die Verwendung von AWS IoT Greengrass:

AWS IoT -Objekt

Ein -AWS IoTObjekt ist eine Darstellung eines bestimmten Geräts oder einer bestimmten logischen Entität. Informationen zu einem Objekt werden in der AWS IoT Registrierung gespeichert.

Greengrass-Core-Gerät

Ein Gerät, auf dem die AWS IoT Greengrass -Core-Software ausgeführt wird. Ein Greengrass-Core-Gerät ist ein AWS IoT-Objekt. Sie können AWS IoT Objektgruppen mehrere Core-Geräte hinzufügen, um Gruppen von Greengrass-Core-Geräten zu erstellen und zu verwalten. Weitere Informationen finden Sie unter Einrichtung von AWS IoT Greengrass Kerngeräten.

Greengrass-Client-Gerät

Ein Gerät, das über MQTT eine Verbindung zu einem Greengrass-Core-Gerät herstellt und mit diesem kommuniziert. Ein Greengrass-Client-Gerät ist ein -AWS IoTObjekt. Das Core-Gerät kann Daten von Client-Geräten verarbeiten, filtern und aggregieren, die eine Verbindung zu ihm herstellen. Sie können das Core-Gerät so konfigurieren, dass MQTT-Nachrichten zwischen Client-Geräten, dem AWS IoT Core Cloud-Service und Greengrass-Komponenten weitergeleitet werden. Weitere Informationen finden Sie unter Interagieren mit lokalen IoT-Geräten.

Client-Geräte können FreeRTOS ausführen oder die AWS IoT Device SDK oder die Greengrass-Discovery-API verwenden, um Informationen über Core-Geräte abzurufen, mit denen sie eine Verbindung herstellen können.

Greengrass-Komponente

Ein Softwaremodul, das auf einem Greengrass-Kerngerät bereitgestellt wird und darauf ausgeführt wird. Alle Software, die mit entwickelt und bereitgestellt wird, AWS IoT Greengrass wird als Komponente modelliert. AWS IoT Greengrass bietet vorgefertigte öffentliche Komponenten, die Funktionen und Funktionalität bereitstellen, die Sie in Ihren Anwendungen verwenden können. Sie können auch Ihre eigenen benutzerdefinierten Komponenten auf Ihrem lokalen Gerät oder in der Cloud entwickeln. Nachdem Sie eine benutzerdefinierte Komponente entwickelt haben, können Sie den AWS IoT Greengrass Cloud-Service verwenden, um sie auf einzelnen oder mehreren Core-Geräten bereitzustellen. Sie können eine benutzerdefinierte Komponente erstellen und diese Komponente auf einem Core-Gerät bereitstellen. Wenn Sie dies tun, lädt das Core-Gerät die folgenden Ressourcen herunter, um die Komponente auszuführen:

- Rezept : Eine JSON- oder YAML-Datei, die das Softwaremodul beschreibt, indem Komponentendetails, Konfiguration und Parameter definiert werden.
- Artefakt : Der Quellcode, die Binärdateien oder Skripts, die die Software definieren, die auf Ihrem Gerät ausgeführt wird. Sie können Artefakte von Grund auf neu erstellen oder eine Komponente mit einer Lambda-Funktion, einem Docker-Container oder einer benutzerdefinierten Laufzeit erstellen.
- Abhängigkeit: Die Beziehung zwischen Komponenten, mit der Sie automatische Updates oder Neustarts abhängiger Komponenten erzwingen können. Sie können beispielsweise eine sichere Nachrichtenverarbeitungskomponente haben, die von einer Verschlüsselungskomponente abhängt. Dadurch wird sichergestellt, dass alle Aktualisierungen der Verschlüsselungskomponente die Nachrichtenverarbeitungskomponente automatisch aktualisieren und neu starten.

Weitere Informationen finden Sie unter AWSVon bereitgestellte Komponenten und Entwickeln von AWS IoT Greengrass Komponenten.

Bereitstellung

Der Prozess zum Senden von Komponenten und Anwenden der gewünschten Komponentenkonfiguration auf ein Zielgerät, bei dem es sich um ein einzelnes Greengrass-Kerngerät oder eine Gruppe von Greengrass-Kerngeräten handeln kann. Bereitstellungen wenden automatisch alle aktualisierten Komponentenkonfigurationen auf das Ziel an und schließen alle anderen Komponenten ein, die als Abhängigkeiten definiert sind. Sie können auch eine vorhandene Bereitstellung klonen, um eine neue Bereitstellung zu erstellen, die dieselben Komponenten verwendet, aber auf einem anderen Ziel bereitgestellt wird. Bereitstellungen sind kontinuierlich, was bedeutet, dass alle Aktualisierungen, die Sie an den Komponenten oder der Komponentenkonfiguration einer Bereitstellung vornehmen, automatisch an alle Zielziele gesendet werden. Weitere Informationen finden Sie unter Bereitstellen von AWS IoT Greengrass Komponenten auf Geräten.

AWS IoT Greengrass Core-Software

Der Satz aller AWS IoT Greengrass Software, die Sie auf einem Core-Gerät installieren. AWS IoT Greengrass Die Core-Software umfasst Folgendes:

- Bol : Diese erforderliche Komponente bietet die Mindestfunktionalität der -AWS IoT GreengrassCore-Software. Der -Kern verwaltet Bereitstellungen, Orchestrierung und Lebenszyklusmanagement anderer Komponenten. Es erleichtert auch die lokale Kommunikation zwischen AWS IoT Greengrass Komponenten auf einem einzelnen Gerät. Weitere Informationen finden Sie unter Grüngraskern.
- Optionale Komponenten: Diese konfigurierbaren Komponenten werden von bereitgestellt AWS IoT Greengrass und ermöglichen zusätzliche Funktionen auf Ihren Edge-Geräten. Abhängig von Ihren Anforderungen können Sie die optionalen Komponenten auswählen, die Sie auf Ihrem Gerät bereitstellen möchten, z. B. Daten-Streaming, lokale Inferenz für Machine Learning oder eine lokale Befehlszeilenschnittstelle. Weitere Informationen finden Sie unter AWSVon bereitgestellte Komponenten.

Sie können Ihre AWS IoT Greengrass -Core-Software aktualisieren, indem Sie neue Versionen Ihrer Komponenten auf Ihrem Gerät bereitstellen.

Features von AWS IoT Greengrass

AWS IoT Greengrass Version 2 besteht aus den folgenden Elementen:

- Softwareverteilungen
  - Die Greengrass-Kernkomponente , die die Mindestinstallation der AWS IoT Greengrass Core-Software ist. Diese Komponente verwaltet Bereitstellungen, Orchestrierung und Lebenszyklusmanagement von Greengrass-Komponenten.
  - Zusätzliche optionale, von AWSbereitgestellte Komponenten, die in -Services, -Protokolle und -Software integriert werden können.
  - Greengrass-Entwicklungstools, mit denen Sie benutzerdefinierte Greengrass-Komponenten erstellen, testen, erstellen, veröffentlichen und bereitstellen können.
  - Die AWS IoT Device SDK, die die Interprocess Communication (IPC)-Bibliothek für benutzerdefinierte Greengrass-Komponenten und die Greengrass-Discovery-Bibliothek für Client-Geräte enthält.
  - Das Stream Manager SDK, mit dem Sie Datenströme auf Core-Geräten verwalten können.
- Cloud-Service
  - AWS IoT Greengrass V2-API
  - AWS IoT Greengrass V2-Konsole

AWS IoT Greengrass Core-Software

Sie können die -AWS IoT GreengrassCore-Software, die auf Ihren Edge-Geräten ausgeführt wird, verwenden, um Folgendes zu tun:

- Verarbeiten Sie Datenströme auf dem lokalen Gerät mit automatischen Exporten in die AWS Cloud. Weitere Informationen finden Sie unter Verwalten von Datenströmen auf Greengrass-Core-Geräten.
- Unterstützung von MQTT-Nachrichten zwischen - AWS IoT und -Komponenten. Weitere Informationen finden Sie unter MQTT-Nachrichten veröffentlichen/abonnieren AWS IoT Core.
- Interagieren Sie mit lokalen Geräten, die eine Verbindung herstellen und über MQTT kommunizieren. Weitere Informationen finden Sie unter Interagieren mit lokalen IoT-Geräten.
- Unterstützung für lokales Veröffentlichen und Abonnieren von Messaging zwischen Komponenten. Weitere Informationen finden Sie unter Lokale Nachrichten veröffentlichen/abonnieren.
- Stellen Sie Komponenten und Lambda-Funktionen bereit und rufen Sie sie auf. Weitere Informationen finden Sie unter Bereitstellen von AWS IoT Greengrass Komponenten auf Geräten.

- Verwalten Sie Komponentenlebenszyklen, z. B. mit Unterstützung für die Installation und Ausführung von Skripts. Weitere Informationen finden Sie unter AWS IoT Greengrass Referenz zum Komponenten-Rezept.
- Führen Sie sichere Softwareupdates over-the-air (OTA) der -AWS IoT GreengrassCore-Software und benutzerdefinierter Komponenten durch. Weitere Informationen finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA) und Bereitstellen von AWS IoT Greengrass Komponenten auf Geräten.
- Ermöglichen Sie eine sichere, verschlüsselte Speicherung lokaler Secrets und einen kontrollierten Zugriff durch Komponenten. Weitere Informationen finden Sie unter Geheimer Manager.
- Sichere Verbindungen zwischen Geräten und der AWS Cloud mit Geräteauthentifizierung und -autorisierung. Weitere Informationen finden Sie unter Geräteauthentifizierung und -autorisierung für AWS IoT Greengrass.

Sie konfigurieren und verwalten Greengrass-Core-Geräte über AWS IoT Greengrass APIs, in denen Sie kontinuierliche Softwarebereitstellungen erstellen. Weitere Informationen finden Sie unter Bereitstellen von AWS IoT Greengrass Komponenten auf Geräten.

Einige Funktionen werden nur auf bestimmten Plattformen unterstützt. Weitere Informationen finden Sie unter Greengrass-Funktionskompatibilität nach Betriebssystem.

Weitere Informationen zu unterstützten Plattformen, Anforderungen und Downloads finden Sie unter Einrichtung von AWS IoT Greengrass Kerngeräten.

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

Greengrass-Funktionskompatibilität nach Betriebssystem

AWS IoT Greengrass unterstützt Geräte, auf denen verschiedene Betriebssysteme ausgeführt werden. Einige Funktionen werden nur auf bestimmten Betriebssystemen unterstützt. In den folgenden Tabellen erfahren Sie, welche Funktionen für jedes unterstützte Betriebssystem verfügbar sind. Weitere Informationen zu unterstützten Betriebssystemen, Anforderungen und zum Einrichten von Greengrass-Core-Geräten finden Sie unter Einrichtung von AWS IoT Greengrass Kerngeräten.

Messaging

Funktion	Linux	Windows
Austauschen von MQTT-Nachrichten zwischen - AWS IoT und -Komponenten	 Ja	 Ja
Lokale Veröffentlichungs-/Abonnementnachrichten zwischen Komponenten austauschen	 Ja	 Ja
Interagieren mit lokalen IoT-Geräten über MQTT	 Ja	 Ja
Interagieren mit lokalen Bol-RTU-Geräten mithilfe der Bol-RTU-Komponente	 Ja	 Nein

Sicherheit

Funktion	Linux	Windows
Sichere Verbindungen mit Geräteauthentifizierung und -autorisierung	 Ja	 Ja
Bereitstellen und Zugriff auf sichere, verschlüsselte Secrets von AWS Secrets Manager	 Ja	 Ja

Funktion	Linux	Windows
Verwenden eines Hardware-Sicherheitsmoduls (HSM), um den privaten Schlüssel und das Zertifikat des Geräts sicher zu speichern	 Ja	 Nein
Prüfen von -Core-Geräten mit AWS IoT Device Defender	 Ja	 Ja
Verwenden von AWS Anmeldeinformationen für die Interaktion mit - AWS Services	 Ja	 Ja

Installation

Funktion	Linux	Windows
Installation AWS IoT Greengrass mit automatischer Bereitstellung	 Ja	 Ja
Installation AWS IoT Greengrass mit manueller Bereitstellung	 Ja	 Ja
Installation AWS IoT Greengrass mit AWS IoT Flottenbereitstellung	 Ja	 Ja

Funktion	Linux	Windows
Installieren AWS IoT Greengrass mit benutzerd efinierten Bereitstellungs-Pl ugins	 Ja	 Ja
Ausführen AWS IoT Greengrass in einem Docker-Container mit einem vorgefert igten Docker-Image	 Ja	 Nein

Remote-Wartung und Updates

Funktion	Linux	Windows
Ausführen sicherer Softwareu pdates over-the-air (OTA)	 Ja	 Ja
Verwalten von -Core-Geräten mit AWS Systems Manager	 Ja	 Nein
Herstellen einer Verbindung zu -Core-Geräten mit AWS IoT Secure Tunneling	 Ja	 Nein

Machine Learning

Funktion	Linux	Windows
Durchführen von Machine-Learning-Inferenzen mit Amazon SageMaker Edge Manager	 Ja	 Ja
Durchführen von Machine-Learning-Inferenzen mit Amazon Lookout for Vision	 Ja	 Nein
Durchführen von Machine-Learning-Inferenzen mit DLR	 Ja	 Ja
Durchführen von Machine Learning-Inferenzen mit TensorFlow	 Ja	 Ja

Komponenten-Features

Funktion	Linux	Windows
Bereitstellen und Aufrufen von Lambda-Funktionen	 Ja	 Nein
Ausführen von Docker-Containern in Komponenten	 Ja	 Ja

Funktion	Linux	Windows
Verarbeiten und Exportieren von Datenströmen mit hohem Volumen mithilfe von Stream Manager	 Ja	 Ja
Verwalten von Komponentenlebenszyklen mit Lebenszyklusskripten	 Ja	 Ja
Interagieren mit Gerätesch atten	 Ja	 Ja
Hochladen von Protokollen in Amazon CloudWatch Logs	 Ja	 Ja
Hochladen von Daten in Amazon- CloudWatch Metriken mithilfe der CloudWatch Metrikkomponente	 Ja	 Ja
Veröffentlichen von Nachrichten in Amazon Simple Notification Service mithilfe der Amazon SNS-Komponente	 Ja	 Nein
Veröffentlichen von Daten in Amazon-Data-Firehose-Bereitstellungsdatenströmen mithilfe des Stream-Managers	 Ja	 Ja

Funktion	Linux	Windows
Veröffentlichen von Daten in Amazon-Data-Firehose-Bereitstellungsdatenströmen mithilfe der Firehose-Komponente	 Ja	 Nein
Erfassen und handeln Sie auf Echtzeit-Systemtelemetrieme triken	 Ja	 Ja
Konfigurieren von Systemressourcenlimits für Komponentenprozesse	 Ja	 Nein
Anhalten und Fortsetzen von Komponentenprozessen	 Ja	 Nein
Integrieren mit AWS IoT SiteWise mithilfe der AWS IoT SiteWise Komponenten	 Ja	 Ja
Veröffentlichen von Videostreams in Amazon Kinesis Video Streams mithilfe des Edge-Konnektors für die Komponente Kinesis Video Streams	 Ja	 Nein

Komponentenentwicklung

Funktion	Linux	Windows
Lokales Entwickeln von Komponenten auf -Core-Ger äten	 Ja	 Ja
Interagieren mit einem Core-Gerät mithilfe der AWS IoT Greengrass CLI	 Ja	 Ja
Interagieren mit einem Core-Gerät mithilfe der lokalen Debug-Konsole	 Ja	 Ja
Verwenden der AWS IoT Device SDK für Python in benutzerdefinierten Komponenten	 Ja	 Ja
Verwenden der AWS IoT Device SDK für C+ + in benutzerdefinierten Komponenten	 Ja	 Ja
Verwenden der AWS IoT Device SDK für Java in benutzerdefinierten Komponenten	 Ja	 Ja

Gerätezertifizierung

Funktion	Linux	Windows
Verwenden von AWS IoT Device Tester für AWS IoT Greengrass V2 die Validierung von IoT-Geräten	 Ja	 Ja

Was ist neu in AWS IoT Greengrass Version 2

AWS IoT Greengrass Version 2 ist eine Hauptversion von AWS IoT Greengrass , die die folgenden Funktionen einführt:

- Von Publisher unterstützte Komponenten — bietet AWS IoT Greengrass jetzt auch vom Publisher unterstützte Komponenten. Diese Komponenten werden von Drittanbietern entwickelt, angeboten und gewartet. Weitere Informationen finden Sie unter Vom Publisher unterstützte Komponenten.
- Betrieb eines Greengrass-Geräts in VPC — Der Betrieb eines Greengrass-Core-Geräts in VPC ist jetzt verfügbar. Auf diese Weise können Sie Bereitstellungen in VPC ohne öffentlichen Internetzugang durchführen. Weitere Informationen finden Sie unter Betreiben eines -AWS IoT GreengrassCore-Geräts in der VPC.
- Greengrass Testing Framework (GTF) — GTF for AWS IoT Greengrass Version 2 ist jetzt verfügbar. GTF ist eine Sammlung von Bausteinen zur Unterstützung der Automatisierung. end-to-end Es ermöglicht AWS IoT Greengrass Version 2 internen Kunden, dasselbe Test-Framework zu verwenden, das das Serviceteam zur Qualifizierung von Softwareänderungen, zur automatisierten Abnahme und zur Qualitätssicherung verwendet. Weitere Informationen finden Sie unter Greengrass Testing Framework auf Github.
- PSA-zertifiziert — Die AWS IoT Greengrass Nucleus-Versionen 2.7.0 und höher sind jetzt für die Platform Security Architecture (PSA) zertifiziert. Weitere Informationen finden Sie unter Ist PSA-zertifiziert.AWS IoT Greengrass

AWS IoT Greengrass In den Versionshinweisen finden Sie Einzelheiten zu AWS IoT Greengrass Versionen, d. h. zu neuen Funktionen, Updates und Verbesserungen sowie allgemeinen Problembehebungen. AWS IoT Greengrass hat die folgenden Arten von Versionen:

- Neue Feature-Releases für AWS IoT Greengrass
- AWS IoT Greengrass Kern-Softwareupdates

Dieser Abschnitt enthält alle AWS IoT Greengrass V2 Versionshinweise, die neuesten zuerst, und enthält wichtige Funktionsänderungen und wichtige Fehlerkorrekturen. Informationen zu weiteren kleineren Korrekturen finden Sie bei der Organisation aws-greengrass unter. GitHub

Versionshinweise

- Veröffentlichung: AWS IoT Greengrass Core v2.12.6-Softwareupdate am 24. Mai 2024

- Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.12.5 am 25. April 2024
- Veröffentlichung: AWS IoT Greengrass Core v2.12.4-Softwareupdate am 02. April 2024
- Veröffentlichung: AWS IoT Greengrass Core v2.12.3-Softwareupdate am 27. März 2024
- Veröffentlichung: AWS IoT Greengrass Core v2.12.2-Softwareupdate am 15. Februar 2024
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.12.1 am 8. Dezember 2023
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.12.0 am 7. November 2023
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.11.3 am 18. Oktober 2023
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.11.2 am 9. August 2023
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.11.1 am 21. Juli 2023
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.11.0 am 28. Juni 2023
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.10.3 am 21. Juni 2023
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.10.2 am 5. Juni 2023
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.10.1 am 11. Mai 2023
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.10.0 am 9. Mai 2023
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.9.6 am 20. April 2023
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.9.5 am 30. März 2023
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.9.4 am 24. Februar 2023
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.9.3 am 01. Februar 2023
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.9.2 am 22. Dezember 2022
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.9.1 am 18. November 2022
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.9.0 am 15. November 2022
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.8.1 am 13. Oktober 2022
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.8.0 am 7. Oktober 2022
- Veröffentlichung: Update der Software AWS IoT Greengrass Core v2.7.0 am 28. Juli 2022
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.6.0 am 27. Juni 2022
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.5.6 am 31. Mai 2022
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.5.5 am 6. April 2022
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.5.4 am 23. März 2022

- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.5.3 am 6. Januar 2022
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.5.2 am 3. Dezember 2021
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.5.1 am 23. November 2021
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.5.0 am 12. November 2021
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.4.0 am 3. August 2021
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.3.0 am 29. Juni 2021
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.2.0 am 18. Juni 2021
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.1.0 am 26. April 2021
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.0.5 am 09. März 2021
- Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.0.4 am 04. Februar 2021

Veröffentlichung: AWS IoT Greengrass Core v2.12.6-Softwareupdate am 24. Mai 2024

Diese Version enthält Version 2.12.6 der Greengrass Nucleus-Komponente und Updates der bereitgestellten Komponenten. AWS

Veröffentlichungsdatum: 24. Mai 2024

Einzelheiten zur Veröffentlichung

- Öffentliche Komponenten-Updates

Öffentliche Komponenten-Updates

In der folgenden Tabelle sind die von bereitgestellten Komponenten aufgeführt AWS , die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, werden die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente AWS IoT Greengrass installiert. Aus diesem Grund werden neue Patch-Versionen von AWS bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren Kerngeräten bereitgestellt, wenn Sie einer Dinggruppe neue Geräte hinzufügen oder wenn Sie die Bereitstellung aktualisieren, die für diese Geräte

vorgesehen ist. Einige automatische Updates, wie z. B. ein Nucleus-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir, dass Sie Ihre bevorzugte Version dieser Komponente direkt angeben, wenn Sie eine Bereitstellung erstellen. Weitere Informationen zum Aktualisierungsverhalten der AWS IoT Greengrass Core-Software finden Sie unterAktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.12.6 von Greengrass Nucleus ist verfügbar. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, das bei bestimmten ARMv8-Prozessoren, einschlie ßlich dem Jetson Nano, beim Start zu einem Absturz führt.
Greengrass CLI	Version 2.12.6 der Greengrass CLI ist verfügbar. Fehlerkorrekturen und Verbesserungen • Die Version wurde für die Version 2.12.6 von Greengrass Nucleus aktualisiert.
Geheimer Manager	Version 2.1.8 des Secret Managers ist verfügbar. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem der Secret Manager einen teilweisen ARN nicht akzeptiert.

Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.12.5 am 25. April 2024

Diese Version enthält Version 2.12.5 der Greengrass Nucleus-Komponente und Updates der bereitgestellten Komponenten. AWS

Veröffentlichungsdatum: 25. April 2024

Einzelheiten zur Veröffentlichung

- Öffentliche Komponenten-Updates

Öffentliche Komponenten-Updates

In der folgenden Tabelle sind die von bereitgestellten Komponenten aufgeführt AWS , die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, werden die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente AWS IoT Greengrass installiert. Aus diesem Grund werden neue Patch-Versionen von AWS bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren Kerngeräten bereitgestellt, wenn Sie einer Dinggruppe neue Geräte hinzufügen oder wenn Sie die Bereitstellung aktualisieren, die für diese Geräte vorgesehen ist. Einige automatische Updates, wie z. B. ein Nucleus-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
> Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir, dass Sie Ihre bevorzugte Version dieser Komponente direkt angeben, wenn Sie eine Bereitstellung erstellen. Weitere Informationen zum Aktualisierungsverhalten der AWS IoT Greengrass Core-Software finden Sie unterAktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.12.5 von Greengrass Nucleus ist verfügbar. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem das Rollback der Bereitstellung gelegentl ich hängen bleibt, wenn eine zuvor defekte Komponente mit festen Abhängigkeiten zurückgesetzt wird. • Behebt ein Problem, bei dem der Nucleus nach der Flottenbereitstellung keine Status-Updates veröffentlicht. • Fügt Wiederholungsversuche für die `GetDeploymentConfi guration` API hinzu, nachdem 404-Fehler aufgetreten sind.

Veröffentlichung: AWS IoT Greengrass Core v2.12.4-Softwareupdate am 02. April 2024

Diese Version enthält Version 2.12.4 der Greengrass Nucleus-Komponente und Updates der bereitgestellten Komponenten. AWS

Veröffentlichungsdatum: 02. April 2024

Einzelheiten zur Veröffentlichung

- Öffentliche Komponenten-Updates

Öffentliche Komponenten-Updates

In der folgenden Tabelle sind die von bereitgestellten Komponenten aufgeführt AWS , die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, werden die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente AWS IoT Greengrass installiert. Aus diesem Grund werden neue Patch-Versionen von AWS bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren Kerngeräten bereitgestellt, wenn Sie einer Dinggruppe neue Geräte hinzufügen oder wenn Sie die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Nucleus-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
> Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir, dass Sie Ihre bevorzugte Version dieser Komponente direkt angeben, wenn Sie eine Bereitstellung erstellen. Weitere Informationen zum Aktualisierungsverhalten der AWS IoT Greengrass Core-Software finden Sie unterAktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.12.4 von Greengrass Nucleus ist verfügbar.

Komponente	Details
	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem der Nucleus beim Start auf einigen Linux-Geräten in einen Deadlock-Zustand gerät.

Veröffentlichung: AWS IoT Greengrass Core v2.12.3-Softwareupdate am 27. März 2024

Diese Version enthält Version 2.12.3 der Greengrass Nucleus-Komponente und Updates der bereitgestellten Komponenten. AWS

Veröffentlichungsdatum: 27. März 2024

Einzelheiten zur Veröffentlichung

- Öffentliche Komponenten-Updates

Öffentliche Komponenten-Updates

In der folgenden Tabelle sind die von bereitgestellten Komponenten aufgeführt AWS , die neue und aktualisierte Funktionen enthalten.

Important

Wenn Sie eine Komponente bereitstellen, werden die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente AWS IoT Greengrass installiert. Aus diesem Grund werden neue Patch-Versionen von AWS bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren Kerngeräten bereitgestellt, wenn Sie einer Dinggruppe neue Geräte hinzufügen oder wenn Sie die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Nucleus-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir, dass Sie Ihre bevorzugte Version dieser Komponente direkt angeben, wenn Sie eine Bereitstellung erstellen. Weitere Informationen zum Aktualisierungsverhalten der AWS IoT Greengrass Core-Software finden Sie unterAktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.12.3 von Greengrass Nucleus ist verfügbar. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem der Nucleus nach dem Neustart des Nucleus und während der Wiederherstellung der Komponenten nicht den korrekten Komponentenstatus meldet. • Allgemeine Fehlerbehebungen und Verbesserungen.
Schattenmanager	Version 2.3.7 der Shadow Manager-Komponente ist verfügbar. Fehlerkorrekturen und Verbesserungen Behebt ein Problem, bei dem Shadow Manager während einer Shadow Manager-Synchronisierung regelmäßig einen `NullPointerException` Fehler protokolliert.
Bereitstellung von Flotten	Version 1.2.1 des AWS IoT Fleet Provisioning-Plugins ist verfügbar. Fehlerkorrekturen und Verbesserungen Behebt ein Problem, bei dem das Fleet Provisioning Plugin während eines Greengrass Nucleus-Starts offline ist. Das Fleet Provisioning-Plugin versucht nun auf unbestimmte Zeit, MQTT Connect-Aufrufe zu wiederholen.
IP-Detektor	Version 2.1.9 der Disk Spooler-Komponente ist verfügbar. Fehlerkorrekturen und Verbesserungen Passt den Schritt zur Erfassung der IP-Adresse so an, dass nur Protokolle auf Debug-Protokollebene gesendet werden.
Moquette MQTT 3.1.1 Broker-Komponente	Version 2.3.6 der Moquette MQTT 3.1.1 Broker-Komponente ist verfügbar. Fehlerkorrekturen und Verbesserungen Allgemeine Fehlerbehebungen und Verbesserungen.

Komponente	Details
Lambda-Manager	Version 2.3.3 der Lambda-Manager-Komponente ist verfügbar. Fehlerkorrekturen und Verbesserungen Allgemeine Fehlerbehebungen und Verbesserungen.
Lokale Debug-Konsole	Version 2.4.2 der lokalen Debug-Konsolenkomponente ist verfügbar. Fehlerkorrekturen und Verbesserungen Allgemeine Fehlerbehebungen und Verbesserungen.

Veröffentlichung: AWS IoT Greengrass Core v2.12.2-Softwareupdate am 15. Februar 2024

Diese Version enthält Version 2.12.2 der Greengrass-Kernkomponente und Updates für von AWSbereitgestellte Komponenten.

Veröffentlichungsdatum: 15. Februar 2024

Versionsdetails

- Aktualisierungen öffentlicher Komponenten

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind die von bereitgestellten Komponenten aufgeführt AWS , die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren AWS-Core-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige

automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente anzugeben. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.12.2 des Greengrass-Kerns ist verfügbar. Fehlerbehebungen und Verbesserungen • Behebt ein Problem, bei dem alte Protokolle nicht ordnungsgemäß bereinigt wurden. • Allgemeine Fehlerbehebungen und Verbesserungen.
Schattenmanager	Version 2.3.6 der Shadow Manager-Komponente ist verfügbar. Fehlerbehebungen und Verbesserungen Behebt ein Problem, bei dem Schatteneigenschaften, die durch AWS Cloud Updates gelöscht werden, während das Gerät offline ist, auch nach dem Wiedererlangen der Konnektivität im lokalen Schatten bestehen.
Lambda-Launcher	Version 2.0.13 der Lambda-Launcher-Komponente ist verfügbar. Fehlerbehebungen und Verbesserungen Allgemeine Fehlerbehebungen und Verbesserungen.
Festplatten-Spooler	Version 1.0.3 der Datenträger-Spooler-Komponente ist verfügbar. Fehlerbehebungen und Verbesserungen Verbessert die Leistung durch die Wiederverwendung von Datenbankverbindungen.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.12.1 am 8. Dezember 2023

Diese Version enthält Version 2.12.1 der Greengrass-Kernkomponente und Updates für von AWSbereitgestellte Komponenten.

Veröffentlichungsdatum: 8. Dezember 2023

Versionsdetails

- Aktualisierungen öffentlicher Komponenten

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind die von bereitgestellten Komponenten aufgeführtAWS, die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren AWS-Core-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
> Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente aufzunehmen. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.12.1 des Greengrass-Kernus ist verfügbar.

Komponente	Details
	Fehlerbehebungen und Verbesserungen • Behebt ein Problem, bei dem der Kern MQTT-Abonnements für Bereitstellungsthemen duplizieren kann, was zu zusätzlicher Protokoll ierung und MQTT-Veröffentlichungen führt.
Client-Geräte-Authentifizierung	Version 2.4.5 der Clientgerät-Authentifizierungskomponente ist verfügbar. Neue Features Fügt Unterstützung für Platzhalterpräfixe für die Auswahl von Objektnamen mit dem `selectionRule` Parameter hinzu. Fehlerbehebungen und Verbesserungen Behebt ein Problem, bei dem Zertifikate in bestimmten Fällen nicht mit neuen Konnektivitätsinformationen aktualisiert werden.
Festplatten-Spooler	Version 1.0.2 der Datenträger-Spooler-Komponente ist verfügbar. Fehlerbehebungen und Verbesserungen Behebt ein Problem, bei dem das MQTT-Nachrichtenformatfeld in bestimmten Fällen nicht beibehalten wird.
MQTT-Brücke	Version 2.3.1 der Datenträger-Spooler-Komponente ist verfügbar. Fehlerbehebungen und Verbesserungen Behebt ein Problem, bei dem der lokale MQTT-Client in eine Verbindun gstrennungsschleife gelangt.
Stream-Manager	Version 2.1.12 der Stream-Manager-Komponente ist verfügbar. Fehlerbehebungen und Verbesserungen Aktualisiert die Reihenfolge, in der Anmeldeinformationen verwendet werden, sodass Greengrass-Anmeldeinformationen für AWS Servicean fragen bevorzugt werden.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.12.0 am 7. November 2023

Diese Version enthält Version 2.12.0 der Greengrass-Kernkomponente und Updates für von AWSbereitgestellte Komponenten.

Veröffentlichungsdatum: 7. November 2023

Veröffentlichungs-Merkmale

- Bootstrap beim Rollback – bietet AWS IoT Greengrass jetzt einen Greengrass-Kernkonfigurationsparameter namens `BootstrapOnRollback`. Mit dieser Funktion können Sie die Bootstrap-Lebenszyklusschritte als Teil einer Rollback-Bereitstellung ausführen.

Versionsdetails

- Aktualisierungen öffentlicher Komponenten

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind die von bereitgestellten Komponenten aufgeführtAWS, die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren CoreAWS-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
> Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente anzugeben. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.12.0 des Greengrass-Kerns ist verfügbar. Neue Features • Ermöglicht das Ausführen der Bootstrap-Lebenszyklusschritte als Teil einer Rollback-Bereitstellung.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.11.3 am 18. Oktober 2023

Diese Version enthält Version 2.11.3 der Greengrass-Kernkomponente.

Veröffentlichungsdatum: 18. Oktober 2023

Versionsdetails

- Aktualisierungen öffentlicher Komponenten

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind die von bereitgestellten Komponenten aufgeführtAWS, die neue und aktualisierte Funktionen enthalten.

Important

Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren AWS-Core-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente anzugeben. Weitere Informationen zum

Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.11.3 des Greengrass-Kerns ist verfügbar. Fehlerbehebungen und Verbesserungen • Behebt ein Problem im Kern, bei dem eine Komponente möglicher weise nicht ordnungsgemäß gestartet wird, wenn ihre Abhängigkeiten fehlschlagen. Neue Features • Fügt einen konfigurierbaren S3-Endpunkttyp hinzu.
Lambda-Ma nager	Version 2.3.1 der Lambda-Manager-Komponente ist verfügbar. Fehlerbehebungen und Verbesserungen • Passt die Protokollstufen für bestimmte Fehler an.
Lokale Deubg-Konsole	Version 2.4.0 der Lambda-Manager-Komponente ist verfügbar. Neue Features • Fügt die Stream-Manager-Debugging-Konsole hinzu.
Protokoll manager	Version 2.3.6 der Log Manager-Komponente ist verfügbar. Fehlerbehebungen und Verbesserungen • Passt die Protokollstufen für bestimmte Fehler an.
Schattenm anager	Version 2.3.4 der Shadow Manager-Komponente ist verfügbar. Fehlerbehebungen und Verbesserungen • Fügt Unterstützung für Dokumente mit null und leerem Schattenstatus hinzu.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.11.2 am 9. August 2023

Diese Version enthält Version 2.11.2 der Greengrass-Kernkomponente.

Veröffentlichungsdatum: 9. August 2023

Versionsdetails

- Aktualisierungen öffentlicher Komponenten

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind die von bereitgestellten Komponenten aufgeführtAWS, die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren AWS-Core-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
> Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente anzugeben. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.11.2 des Greengrass-Kernus ist verfügbar.

Komponente	Details
	Fehlerbehebungen und Verbesserungen • Behebt ein Problem im -Kern-MQTT-5-Client, bei dem er offline erscheinen kann, wenn eine große Anzahl (> 50) von Abonnements verwendet wird. • Fügt einen Wiederholungsversuch für den TCP-Fehler des Docker-Dials hinzu.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.11.1 am 21. Juli 2023

Diese Version enthält Version 2.11.1 der Greengrass-Kernkomponente.

Veröffentlichungsdatum: 21. Juli 2023

Versionsdetails

- Aktualisierungen öffentlicher Komponenten

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind die von bereitgestellten Komponenten aufgeführtAWS, die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren CoreAWS-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
> Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente einzubeziehen. Weitere Informationen zum

Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.11.1 des Greengrass-Kernus ist verfügbar. Fehlerbehebungen und Verbesserungen • Behebt ein Problem, bei dem der Kern nicht startet, wenn eine Bootstrap -Aufgabe fehlschlägt und die Bereitstellungsmetadatendatei beschädigt ist. • Behebt ein Problem, bei dem On-Demand-Lambda-Komponenten nicht in Bereitstellungsstatusaktualisierungen gemeldet werden. • Fügt Unterstützung für doppelte Autorisierungsrichtlinien-IDs hinzu.
Lambda-Manager	Version 2.2.11 des Lambda-Managers ist verfügbar. Fehlerbehebungen und Verbesserungen • Behebt ein Problem, bei dem die LegacySubscriptionRouter Konfiguration nicht aktualisiert wird, wenn sich die Lambda-Konfiguration ändert.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.11.0 am 28. Juni 2023

Diese Version enthält Version 2.11.0 der Greengrass-Kernkomponente.

Veröffentlichungsdatum: 28. Juni 2023

Veröffentlichungs-Merkmale

- Persistenter Festplatten-Spooler – bietet AWS IoT Greengrass jetzt eine persistente Spooler-Implementierung für Nachrichten, die von Greengrass-Core-Geräten nach gespoolt werdenAWS IoT Core. Diese Komponente speichert diese ausgehenden Nachrichten auf der Festplatte. Weitere Informationen finden Sie unter Festplatten-Spooler.

- Verbesserungen der lokalen Bereitstellung – Sie können jetzt lokale Bereitstellungen abbrechen, Richtlinien für die fehlgeschlagene Bereitstellung festlegen und einen detaillierten Bereitstellungsstatus abrufen.
- Verbesserungen der Protokollierungsgeschwindigkeit – Die Protokoll-Upload-Geschwindigkeiten für die Log Manager-Komponente wurden verbessert.

Versionsdetails

- Aktualisierungen öffentlicher Komponenten

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind die von bereitgestellten Komponenten aufgeführtAWS, die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren CoreAWS-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
> Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente aufzunehmen. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.11.0 des Greengrass-Kerns ist verfügbar. Neue Features • Ermöglicht das Abbrechen einer lokalen Bereitstellung.

Komponente	Details
	• Ermöglicht Ihnen die Konfiguration einer Richtlinie zur Fehlerbehandlung für eine lokale Bereitstellung. • Fügt Unterstützung für ein Disk-Spooler-Plugin hinzu.
Greengrass-CLI	Version 2.11.0 der Greengrass-CLI ist verfügbar. Neue Features • Ermöglicht das Abbrechen einer lokalen Bereitstellung. • Ermöglicht Ihnen die Konfiguration einer Richtlinie zur Fehlerbehandlung für eine lokale Bereitstellung. • Verbessert die detaillierte Berichterstellung zum Bereitstellungsstatus.
Festplatten-Spooler	Version 1.0.0 der Datenträger-Spooler-Komponente ist verfügbar. • Die Datenträger-Spooler-Komponente bietet persistente Speicherung von Nachrichten, die von Greengrass-Core-Geräten an gesendet werdenAWS IoT Core.
Protokoll manager	Version 2.3.5 der Log Manager-Komponente ist verfügbar. Verbesserungen Verbessert die Geschwindigkeit des Protokoll-Uploads.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.10.3 am 21. Juni 2023

Diese Version enthält Version 2.10.3 der Greengrass-Kernkomponente.

Veröffentlichungsdatum: 21. Juni 2023

Versionsdetails

- Aktualisierungen öffentlicher Komponenten

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind die von bereitgestellten Komponenten aufgeführtAWS, die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren CoreAWS-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
> Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente einzubeziehen. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.10.3 des Greengrass-Kerns ist verfügbar. Fehlerbehebungen und Verbesserungen • Behebt ein Problem, bei dem Greengrass keine Bereitstellungsben achrichtigungen abonniert, wenn der PKCS#11-Anbieter verwendet wird.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.10.2 am 5. Juni 2023

Diese Version enthält Version 2.10.2 der Greengrass-Kernkomponente.

Veröffentlichungsdatum: 5. Juni 2023

Versionsdetails

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind die von bereitgestellten Komponenten aufgeführtAWS, die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren CoreAWS-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
> Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente einzubeziehen. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.10.2 des Greengrass-Kerns ist verfügbar. Fehlerbehebungen und Verbesserungen • Ermöglicht das Parsen von Komponentenlebenszyklen ohne Berücksic htigung der Groß- und Kleinschreibung. • Behebt ein Problem, bei dem die Umgebungs-PATH-Variable nicht korrekt neu erstellt wurde. • Behebt die Proxy-URI-Kodierung für Komponenten, einschließlich Stream-Manager für Benutzernamen mit Sonderzeichen.
Client-Geräte-Authentifizierung	Version 2.4.2 der Clientgerät-Authentifizierungskomponente ist verfügbar.

Komponente	Details
	Neue Features Fügt eine neue `startupTimeoutSeconds` Konfigurationsoption hinzu.
Lambda-Manager	Version 2.2.9 der Lambda-Manager-Komponente ist verfügbar. Fehlerbehebungen und Verbesserungen Behebt ein Problem, bei dem die Portnummer aufgrund einer verzerrten Uhr beschädigt wurde.
Protokollmanager	Version 2.3.4 der Log Manager-Komponente ist verfügbar. Fehlerbehebungen und Verbesserungen • Fügt Unterstützung für die Einstellung des `periodicUploadIntervalSec` Parameters auf Bruchwerte hinzu. Das Minimum beträgt 1 Mikrosekunde. • Behebt ein Problem, bei dem Log Manager die CloudWatch `putLogEvents` Limits nicht einhält.
MQTT-3.1-Broker (Moquette)	Version 2.3.3 der MQTT 3.1 Broker (Moquette)-Komponente ist verfügbar. Neue Features Fügt eine neue `startupTimeoutSeconds` Konfigurationsoption hinzu.
MQTT-Brücke	Version 2.2.6 der MQTT-Bridge-Komponente ist verfügbar. Neue Features Fügt eine neue `startupTimeoutSeconds` Konfigurationsoption hinzu.
Stream-Manager	Version 2.1.7 der Stream-Manager-Komponente ist verfügbar. Fehlerbehebungen und Verbesserungen Behebt ein Problem, bei dem Stream Manager die Proxy-Konfiguration nicht korrekt liest.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.10.1 am 11. Mai 2023

Diese Version enthält Version 2.10.1 der Greengrass-Kernkomponente.

Veröffentlichungsdatum: 11. Mai 2023

Versionsdetails

- Aktualisierungen öffentlicher Komponenten

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind die von bereitgestellten Komponenten aufgeführtAWS, die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren AWS-Core-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
> Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente aufzunehmen. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.10.1 des Greengrass-Kerns ist verfügbar.

Komponente	Details
	Fehlerbehebungen und Verbesserungen • Behebt ein Problem, das beim Start auf bestimmten ARMv8-Pro zessoren, einschließlich Jetson Nano, zu einem Absturz führen konnte. • Greengrass schließt den Standard einer Komponente nicht mehr, wodurch das Verhalten auf das Verhalten vor 2.10.0 zurückgesetzt wird
Stream-Manager	Version 2.1.6 des neuen Stream-Managers ist verfügbar. Fehlerbehebungen und Verbesserungen Behebt ein Problem, das beim Start auf bestimmten ARMv8-Prozessoren, einschließlich Jetson Nano, zu einem Absturz führen konnte.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.10.0 am 9. Mai 2023

Diese Version enthält Version 2.10.0 der Greengrass-Kernkomponente und Updates für von AWSbereitgestellte Komponenten.

Veröffentlichungsdatum: 9. Mai 2023

Veröffentlichungs-Merkmale

- MQTT5-Unterstützung – unterstützt AWS IoT Greengrass jetzt das Senden und Empfangen von Nachrichten von AWS IoT Core mithilfe von MQTT5. Weitere Informationen finden Sie unter Veröffentlichen von AWS IoT Core MQTT-Nachrichten.

Versionsdetails

- Aktualisierungen öffentlicher Komponenten

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind die von bereitgestellten Komponenten aufgeführtAWS, die neue und aktualisierte Funktionen enthalten.

Important

Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren AWS-Core-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente anzugeben. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.10.0 des Greengrass-Kerns ist verfügbar. Neue Features • Fügt `interpolateComponentConfiguration` Unterstützung für den leeren regulären Ausdruck hinzu. Greengrass interpoliert jetzt vom Root-Konfigurationsobjekt. • Fügt Unterstützung für MQTT5 hinzu. • Fügt einen Mechanismus zum schnellen Laden von Plugin-Komponenten ohne Scannen hinzu. • Ermöglicht Greengrass, Speicherplatz zu sparen, indem nicht verwendete Docker-Images gelöscht werden. Fehlerbehebungen und Verbesserungen • Behebt ein Problem, bei dem das Rollback bestimmte Konfigurationswerte aus einer Bereitstellung beibehalten hat. • Behebt ein Problem, bei dem der Greengrass-Kern auf eine AWS Domainsequenz in benutzerdefinierten Nicht--AWSAnmeldeinformationen und Datenendpunkten validiert.

Komponente	Details
	• Aktualisiert die Auflösung von Multi-Gruppen-Abhängigkeiten, um alle Gruppenabhängigkeiten per AWS Cloud Aushandlung erneut aufzulöse n, anstatt die aktive Version zu sperren. Durch dieses Update wird auch der Bereitstellungsfehlercode entfernt`INSTALLED_COMPONEN T_NOT_FOUND` . • Aktualisiert den Greengrass-Kern, um das Herunterladen von Docker-Im ages zu überspringen, wenn sie bereits lokal vorhanden sind. • Aktualisiert den Greengrass-Kern, um einen Schritt zur Komponent eninstallation neu zu starten, bevor das Timeout abläuft. • Zusätzliche kleinere Korrekturen und Verbesserungen.
Schattenm anager	Version 2.3.2 des neuen Schattenmanagers ist verfügbar. Fehlerbehebungen und Verbesserungen Behebt ein Problem, bei dem der Schattenmanager in den -`BROKEN`Status übergeht, wenn die lokale Schattendatenbank beschädigt ist.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.9.6 am 20. April 2023

Diese Version enthält Version 2.9.6 der Greengrass-Kernkomponente.

Veröffentlichungsdatum: 20. April 2023

Versionsdetails

- Aktualisierungen öffentlicher Komponenten

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind die von bereitgestellten Komponenten aufgeführtAWS, die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren AWS-Core-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
> Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente anzugeben. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.9.6 des Greengrass-Kernus ist verfügbar. Fehlerbehebungen und Verbesserungen • Behebt ein Problem, bei dem eine Greengrass-Bereitstellung mit dem Fehler LAUNCH_DIRECTORY_CORRUPTED fehlschlägt und ein nachfolgender Geräteneustart Greengrass nicht starten kann. Dieser Fehler kann auftreten, wenn Sie das Greengrass-Gerät zwischen mehreren Objektgruppen mit Bereitstellungen verschieben, für die Greengrass neu gestartet werden muss.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.9.5 am 30. März 2023

Diese Version enthält Version 2.9.5 der Greengrass-Kernkomponente.

Veröffentlichungsdatum: 30. März 2023

Versionsdetails

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind die von bereitgestellten Komponenten aufgeführtAWS, die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren CoreAWS-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
> Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente einzubeziehen. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.9.5 des Greengrass-Kerns ist verfügbar. Neue Features • Fügt Unterstützung für die Überprüfung der Signatur der Greengrass-Kernsoftware hinzu. Fehlerbehebungen und Verbesserungen • Behebt ein Problem, bei dem eine Bereitstellung fehlschlägt, wenn die Metadatenregion des lokalen Rezepts nicht mit der Startregion des Greengrass-Kerns übereinstimmt. Der Greengrass-Kern verhandelt jetzt mit der Cloud neu, wenn dies geschieht.

Komponente	Details
	• Behebt ein Problem, bei dem der MQTT-Nachrichten-Spooler füllt und keine Nachrichten entfernt. • Zusätzliche kleinere Korrekturen und Verbesserungen.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.9.4 am 24. Februar 2023

Diese Version enthält Version 2.9.4 der Greengrass-Kernkomponente.

Veröffentlichungsdatum: 24. Februar 2023

Versionsdetails

- Aktualisierungen öffentlicher Komponenten

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind die von bereitgestellten Komponenten aufgeführtAWS, die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren AWS-Core-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
> Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente aufzunehmen. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.9.4 des Greengrass-Kerns ist verfügbar. Fehlerbehebungen und Verbesserungen • Prüft auf eine Nullnachricht, bevor QOS-0-Nachrichten gelöscht werden. • Kürzt die Detailwerte des Auftragsstatus, wenn sie das Limit von 1024 Zeichen überschreiten. • Aktualisiert das Bootstrap-Skript für Windows, um den Greengrass-Stammpfad korrekt zu lesen, wenn dieser Pfad Leerzeichen enthält. • Aktualisiert das Abonnieren von , AWS IoT Core sodass Clientnac hrichten gelöscht werden, wenn die Abonnementantwort nicht gesendet wurde. • Stellt sicher, dass der Kern seine Konfiguration aus Sicherungsdateien lädt, wenn die Hauptkonfigurationsdatei beschädigt ist oder fehlt.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.9.3 am 01. Februar 2023

Diese Version enthält Version 2.9.3 der Greengrass-Kernkomponente.

Veröffentlichungsdatum: 01. Februar 2023

Versionsdetails

- Aktualisierungen öffentlicher Komponenten

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind die von bereitgestellten Komponenten aufgeführtAWS, die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden

neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren AWS-Core-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente aufzunehmen. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.9.3 des Greengrass-Kerns ist verfügbar. Fehlerbehebungen und Verbesserungen • Stellt sicher, dass MQTT-Client-IDs nicht dupliziert werden. • Fügt robusteres Lesen und Schreiben von Dateien hinzu, um Beschädigungen zu vermeiden und zu beheben. • Wiederholt das Abrufen von Docker-Images bei bestimmten netzwerkbezogenen Fehlern. • Fügt die `noProxyAddresses` Option für die MQTT-Verbindung hinzu.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.9.2 am 22. Dezember 2022

Diese Version enthält Version 2.9.2 der Greengrass-Kernkomponente.

Veröffentlichungsdatum: 22. Dezember 2022

Versionsdetails

- Aktualisierungen öffentlicher Komponenten

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind AWSvon bereitgestellte Komponenten aufgeführt, die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren CoreAWS-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
> Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente einzubeziehen. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.9.2 des Greengrass-Kernus ist verfügbar. Fehlerbehebungen und Verbesserungen • Behebt ein Problem, bei dem die Konfiguration von `interpolateComponentConfiguration` nicht für eine fortlaufende Bereitstellung gilt. • Verwendet OSHI, um alle untergeordneten Prozesse aufzulisten.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.9.1 am 18. November 2022

Diese Version enthält Version 2.9.1 der Greengrass-Kernkomponente und Updates für von AWSbereitgestellte Komponenten.

Veröffentlichungsdatum: 18. November 2022

Veröffentlichungs-Merkmale

- Log Manager – Log Manager verarbeitet jetzt aktive Protokolldateien und lädt sie direkt hoch, anstatt darauf zu warten, dass neue Dateien rotiert werden. Diese Verbesserung reduziert Protokollverzögerungen erheblich. Weitere Informationen finden Sie unter Log Manager.

Versionsdetails

- Aktualisierungen öffentlicher Komponenten

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind AWSvon bereitgestellte Komponenten aufgeführt, die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren AWS-Core-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
> Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente anzugeben. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.9.1 des Greengrass-Kerns ist verfügbar.

Komponente	Details
	Fehlerbehebungen und Verbesserungen • Fügt Fehlerbehebung hinzu, bei der Greengrass neu gestartet wird, wenn eine Bereitstellung eine Plugin-Komponente entfernt.
Protokoll manager	Version 2.3.0 des neuen Protokollmanagers ist verfügbar. **Note** Wir empfehlen, dass Sie ein Upgrade auf Greengrass-Kern 2.9.1 durchführen, wenn Sie ein Upgrade auf Log Manager 2.3.0 durchführ en. Neue Features • Reduziert Protokollverzögerungen, indem aktive Protokolldateien verarbeitet und direkt hochgeladen werden, anstatt darauf zu warten, dass neue Dateien rotiert werden. Fehlerbehebungen und Verbesserungen • Verbessert die Unterstützung der Protokollrotation beim Rotieren von Dateien mit einem eindeutigen Namen. • Zusätzliche kleinere Korrekturen und Verbesserungen.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.9.0 am 15. November 2022

Diese Version enthält Version 2.9.0 der Greengrass-Kernkomponente und Updates für von AWSbereitgestellte Komponenten.

Veröffentlichungsdatum: 15. November 2022

Veröffentlichungs-Merkmale

- Offline-Authentifizierung – unterstützt AWS IoT Greengrass jetzt die Offline-Authentifizierung. Sie können Ihr AWS IoT Greengrass Core-Gerät so konfigurieren, dass Client-Geräte eine Verbindung

zu einem Core-Gerät herstellen können, auch wenn das Core-Gerät nicht mit der Cloud verbunden ist. Weitere Informationen finden Sie unter Offline-Authentifizierung.

- Unterbereitstellungen – Sie können jetzt Unterbereitstellungen erstellen. Sie können eine Unterbereitstellung verwenden, um erfolglose Bereitstellungen aufzulösen. Jede Unterbereitstellung kann eine andere Konfiguration einer erfolglosen Bereitstellung auf einer kleineren Teilmenge von Geräten testen. Weitere Informationen finden Sie unter Erstellen von Unterbereitstellungen.

Versionsdetails

- Aktualisierungen öffentlicher Komponenten

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind AWSvon bereitgestellte Komponenten aufgeführt, die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren CoreAWS-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
> Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente aufzunehmen. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.9.0 des Greengrass-Kerns ist verfügbar.

Komponente	Details
	Neue Features • Fügt die Möglichkeit hinzu, Unterbereitstellungen zu erstellen, die Bereitstellungen mit einer kleineren Teilmenge von Geräten wiederhol en. Diese Funktion bietet eine effizientere Möglichkeit, erfolglose Bereitstellungen zu testen und zu beheben. Fehlerbehebungen und Verbesserungen • Verbessert die Unterstützung für Systeme`useradd`, die nicht über `groupadd`, und verfügen`usermod`. • Zusätzliche kleinere Korrekturen und Verbesserungen.
Client-Geräte-Authentifizierung	Version 2.3.0 der Client-Geräte-Authentifizierungskomponente ist verfügbar. Neue Features • Fügt Unterstützung für die Offline-Authentifizierung von Clientger äten hinzu. Mit dieser Funktion können Client-Geräte weiterhin eine Verbindung zum Core-Gerät herstellen, wenn das Core-Gerät nicht mit dem Internet verbunden ist. • Fügt Unterstützung für vom Kunden bereitgestellte Zertifizierungsstellen (CA) hinzu. Ihr Core-Gerät verwendet eine vom Kunden bereitgestellte Zertifizierungsstelle als Stammzertifikat, um MQTT-Brokerzertifikate zu generieren.
MQTT-5-Broker (EMQX)	Version 1.2.0 der MQTT 5-Broker (EMAX) Komponente ist verfügbar. Neue Features Fügt Unterstützung für Zertifikatsketten hinzu.
Moquette-MQTT-Broker	Version 2.3.0 der neuen Moquette MQTT-Brokerkomponente ist verfügbar. Neue Features Fügt Unterstützung für Zertifikatsketten hinzu.

Komponente	Details
Secret Manager	Version 2.1.4 des neuen Secret-Managers ist verfügbar. Fehlerbehebungen und Verbesserungen Behebt ein Problem, bei dem zwischengespeicherte Secrets entfernt wurden, wenn Secret Manager bereitgestellt wurde und der Greengrass-Kern neu gestartet wurde.
Stream-Manager	Version 2.1.2 des neuen Stream-Managers ist verfügbar. Fehlerbehebungen und Verbesserungen Behebt ein Problem unter Windows OS, das eine nicht englische Sprache verwendet.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.8.1 am 13. Oktober 2022

Diese Version enthält Version 2.8.1 der Greengrass-Kernkomponente.

Veröffentlichungsdatum: 13. Oktober 2022

> **Note**
>
> Wenn Sie Greengrass-Kernversion 2.8.0 verwenden, empfehlen wir dringend, ein Upgrade auf Greengrass-Kernversion 2.8.1 durchzuführen.

Versionsdetails

- Aktualisierungen öffentlicher Komponenten

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind AWSvon bereitgestellte Komponenten aufgeführt, die neue und aktualisierte Funktionen enthalten.

Important

Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren CoreAWS-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente einzubeziehen. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.8.1 des Greengrass-Kerns ist verfügbar. Fehlerbehebungen und Verbesserungen • Behebt ein Problem, bei dem Bereitstellungsfehlercodes aufgrund von Greengrass-API-Fehlern nicht korrekt generiert wurden. • Behebt ein Problem, bei dem Flottenstatusaktualisierungen ungenaue Informationen senden, wenn eine Komponente während einer Bereitstellung einen -ERRORED Status erreicht. • Behebt ein Problem, bei dem Bereitstellungen nicht abgeschlossen werden konnten, wenn Greengrass mehr als 50 bestehende Abonnements hatte.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.8.0 am 7. Oktober 2022

Diese Version enthält Version 2.8.0 der Greengrass-Kernkomponente und Version 1.1.0 der MQTT 5-Broker (EMQX)-Komponente.

Veröffentlichungsdatum: 7. Oktober 2022

Veröffentlichungs-Merkmale

- Bereitstellungsfehlercodes – Der Greengrass-Kern meldet jetzt eine Statusantwort der Bereitstellung, die detaillierte Fehlercodes enthält, wenn eine Komponentenbereitstellung nicht abgeschlossen werden kann. Weitere Informationen finden Sie unter Detaillierte Bereitstellungsfehlercodes.
- Komponentenfehlerstatus – Der Greengrass-Kernus meldet jetzt eine Antwort auf den Komponentenzustand, die detaillierte Fehlerstatus enthält, wenn eine Komponente in den `ERRORED` Status `BROKEN` oder wechselt. Weitere Informationen finden Sie unter Detaillierte Komponenten-Statuscodes.

Versionsdetails

- Aktualisierungen öffentlicher Komponenten

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind AWSvon bereitgestellte Komponenten aufgeführt, die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren AWS-Core-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
>
> Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente anzugeben. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.8.0 des Greengrass-Kerns ist verfügbar. Neue Features • Aktualisiert den Greengrass-Kern, um eine Antwort auf den Bereitstellungsstatus zu melden, die detaillierte Fehlercodes enthält, wenn ein Problem bei der Bereitstellung von Komponenten auf einem Core-Gerät auftritt. Weitere Informationen finden Sie unter Detaillierte Bereitstellungsfehlercodes. • Aktualisiert den Greengrass-Kern, um eine Antwort auf den Komponentenzustand zu melden, die detaillierte Fehlercodes enthält, wenn eine Komponente in den `ERRORED` Status `BROKEN` oder wechselt. Weitere Informationen finden Sie unter Detaillierte Komponenten-Statuscodes. • Erweitert Statusnachrichtenfelder, um die Cloud-Verfügbarkeitsinformationen für Geräte zu verbessern. • Verbessert die Robustheit des Flottenstatusservice. Fehlerbehebungen und Verbesserungen • Ermöglicht einer fehlerhaften Komponente die Neuinstallation, wenn sich ihre Konfiguration ändert. • Behebt ein Problem, bei dem ein Neustart des Kerns während der Bootstrap-Bereitstellung dazu führt, dass eine Bereitstellung fehlschlägt. • Behebt ein Problem unter Windows, bei dem die Installation fehlschlägt, wenn ein Stammpfad Leerzeichen enthält. • Behebt ein Problem, bei dem eine Komponente, die während einer Bereitstellung heruntergefahren wurde, das Shutdown-Skript der neuen Version verwendet. • Verschiedene Verbesserungen beim Herunterfahren. • Zusätzliche kleinere Korrekturen und Verbesserungen.
MQTT-5-Broker (EMQX)	Version 1.1.0 der MQTT 5-Broker (EMAX) Komponente ist verfügbar. Neue Features • Fügt Unterstützung für EMQX-Konfigurationen hinzu, einschließlich Broker-Optionen und Plug-Ins.

Komponente	Details
	Fehlerbehebungen und Verbesserungen • Aktualisiert EMQX auf Version 4.4.9.

Veröffentlichung: Update der Software AWS IoT Greengrass Core v2.7.0 am 28. Juli 2022

Diese Version enthält Version 2.7.0 der Greengrass-Kernkomponente, Version 2.1.0 der Stream-Manager-Komponente und Version 2.2.5 der Lambda-Manager-Komponente.

Veröffentlichungsdatum: 28. Juli 2022

Veröffentlichungs-Merkmale

- Stream-Manager-Telemetriemetriken – Stream Manager sendet jetzt automatisch Telemetriemetriken an Amazon EventBridge, sodass Sie Cloud-Anwendungen erstellen können, die das Datenvolumen überwachen und analysieren, das Ihre Core-Geräte hochladen. Weitere Informationen finden Sie unter Erfassen von Telemetriedaten zum Systemstatus von -AWS IoT GreengrassCore-Geräten.
- Benutzerdefinierte Zertifizierungsstelle (CA) – Clientzertifikate, die von einer benutzerdefinierten Zertifizierungsstelle signiert sind und bei der die Zertifizierungsstelle nicht registriert istAWS IoT, werden jetzt unterstützt. Weitere Informationen finden Sie unter Verwenden Sie ein Gerätezertifikat, das von einer privaten Zertifizierungsstelle signiert wurde.

Versionsdetails

- Aktualisierungen öffentlicher Komponenten

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind AWSvon bereitgestellte Komponenten aufgeführt, die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren AWS-Core-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
> Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente anzugeben. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.7.0 des Greengrass-Kerns ist verfügbar. Neue Features • Aktualisiert den Greengrass-Kern, um Statusaktualisierungen an die AWS IoT Greengrass Cloud zu senden, wenn das Core-Gerät eine lokale Bereitstellung anwendet. • Fügt Unterstützung für Clientzertifikate hinzu, die von einer benutzerd efinierten Zertifizierungsstelle (CA) signiert sind, bei der die Zertifizi erungsstelle nicht bei registriert istAWS IoT. Um diese Funktion zu verwenden, können Sie die neue `greengrassDataPlan eEndpoint` Konfigurationsoption auf setzen`iotdata`. Weitere Informationen finden Sie unter Verwenden Sie ein Gerätezertifikat, das von einer privaten Zertifizierungsstelle signiert wurde. Fehlerbehebungen und Verbesserungen • Behebt ein Problem, bei dem der Greengrass-Kern in bestimmten Szenarien ein Rollback für eine Bereitstellung durchführt, wenn der Kern gestoppt oder neu gestartet wird. Der Kern setzt die Bereitstellung jetzt fort, nachdem der Kern neu gestartet wurde.

Komponente	Details
	• Aktualisiert das Greengrass-Installationsprogramm so, dass es das `--start` Argument berücksichtigt, wenn Sie angeben, dass die Software als Systemservice eingerichtet werden soll. • Aktualisiert das Verhalten von SubscribeToComponentUpdates, um die Bereitstellungs-ID in Ereignissen festzulegen, bei denen der Kern eine Komponente aktualisiert hat. • Zusätzliche kleinere Korrekturen und Verbesserungen.
Stream-Manager	Version 2.1.0 der Stream-Manager-Komponente ist verfügbar. Neue Features • Aktualisiert diese Komponente, um Telemetriemetriken automatisch an Amazon zu senden EventBridge. Weitere Informationen finden Sie unter Erfassen von Telemetriedaten zum Systemstatus von -AWS IoT GreengrassCore-Geräten. Diese Funktion erfordert v2.7.0 oder höher der Greengrass-Kernkomponente . • Version für Greengrass-Kern Version 2.7.0 aktualisiert.
Lambda-Manager	Version 2.2.5 der Lambda-Manager-Komponente ist verfügbar. Neue Features • Fügt Unterstützung für MQTT-Themen-Platzhalter in Ereignisquellen hinzu, in denen Sie lokale Veröffentlichungs-/Abonnementnachrichten abonnieren. Diese Funktion erfordert v2.6.0 oder höher der Greengrass-Kernkomponente . • Version für Greengrass-Kern Version 2.7.0 aktualisiert.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.6.0 am 27. Juni 2022

Diese Version enthält Version 2.6.0 der Greengrass-Kernkomponente, neue von AWSbereitgestellte Komponenten und Updates für AWSvon bereitgestellte Komponenten.

Veröffentlichungsdatum: 27. Juni 2022

Veröffentlichungs-Merkmale

- Platzhalter in lokalen Veröffentlichungs-/Abonnementthemen – Sie können jetzt MQTT-Platzhalter verwenden, wenn Sie lokale Veröffentlichungs-/Abonnementthemen abonnieren. Weitere Informationen finden Sie unter Lokale Nachrichten veröffentlichen/abonnieren und SubscribeToTopic.
- Unterstützung für Client-Geräteschatten – Sie können jetzt mit Client-Geräteschatten in benutzerdefinierten Komponenten interagieren und Client-Geräteschatten mit synchronisierenAWS IoT Core. Weitere Informationen finden Sie unter Interagieren und Synchronisieren von Client-Geräteschatten.
- Lokale MQTT 5-Unterstützung für Client-Geräte – Sie können jetzt den EMQX MQTT 5-Broker bereitstellen, um MQTT 5-Funktionen für die Kommunikation zwischen Client-Geräten und einem Core-Gerät zu verwenden. Weitere Informationen finden Sie unter MQTT 5-Broker (EMAX) und Verbinden von Client-Geräten mit -Core-Geräten.
- Rezeptvariablen in Komponentenkonfigurationen – Sie können jetzt bestimmte Rezeptvariablen in Komponentenkonfigurationen verwenden. Sie können diese Rezeptvariablen verwenden, wenn Sie die Standardkonfiguration einer Komponente in einem Rezept definieren oder wenn Sie eine Komponente in einer Bereitstellung konfigurieren. Weitere Informationen finden Sie unter Rezeptvariablen und Verwenden Sie Rezeptvariablen bei Merge-Updates.
- Platzhalter in IPC-Autorisierungsrichtlinien – Sie können jetzt den * Platzhalter verwenden, um eine beliebige Kombination von Zeichen in IPC-Autorisierungsrichtlinien (Interprocess Communication) abzugleichen. Mit diesem Platzhalter können Sie den Zugriff auf mehrere Ressourcen in einer einzigen Autorisierungsrichtlinie zulassen. Weitere Informationen finden Sie unter Platzhalter in Autorisierungsrichtlinien.
- IPC-Operationen, die lokale Bereitstellungen und Komponenten verwalten – Sie können jetzt benutzerdefinierte Komponenten entwickeln, die lokale Bereitstellungen verwalten und Komponentendetails anzeigen. Weitere Informationen finden Sie unter IPC: Verwalten von lokalen Bereitstellungen und Komponenten.

- IPC-Operationen, die Client-Geräte authentifizieren und autorisieren – Sie können diese Operationen jetzt verwenden, um eine benutzerdefinierte lokale Broker-Komponente zu erstellen. Weitere Informationen finden Sie unter IPC: Authentifizieren und Autorisieren von Client-Geräten.

Versionsdetails

- Aktualisierungen öffentlicher Komponenten

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind AWSvon bereitgestellte Komponenten aufgeführt, die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren CoreAWS-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
> Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente anzugeben. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.6.0 des Greengrass-Kerns ist verfügbar. Neue Features • Fügt Unterstützung für MQTT-Platzhalter hinzu, wenn Sie lokale Veröffentlichungs-/Abonnementthemen abonnieren. Weitere Informati

Komponente	Details
	onen finden Sie unter Lokale Nachrichten veröffentlichen/abonnieren und SubscribeToTopic. • Fügt Unterstützung für Rezeptvariablen in Komponentenkonfigurationen hinzu, außer der *component_dependency_name* `:configuration:` *json_pointer* Rezeptvariablen. Sie können diese Rezeptvariablen verwenden, wenn Sie die einer Komponente `DefaultConfiguration` in einem Rezept definieren oder wenn Sie eine Komponente in einer Bereitstellung konfigurieren. Um dieses Feature zu aktivieren, setzen Sie die interpolateComponentConfiguration Konfigurationsoption auf `true`. Weitere Informationen finden Sie unter Rezeptvariablen und Verwenden Sie Rezeptvariablen bei Merge-Updates. • Fügt volle Unterstützung für den * Platzhalter in IPC-Autorisierungsrichtlinien (Interprocess Communication) hinzu. Sie können jetzt das * Zeichen in einer Ressourcenzeichenfolge angeben, um einer beliebigen Kombination von Zeichen zu entsprechen. Weitere Informationen finden Sie unter Platzhalter in Autorisierungsrichtlinien. • Fügt Unterstützung für benutzerdefinierte Komponenten hinzu, um IPC-Operationen aufzurufen, die die Greengrass-CLI verwendet. Sie können diese IPC-Operationen verwenden, um lokale Bereitstellungen zu verwalten, Komponentendetails anzuzeigen und ein Passwort zu generieren, mit dem Sie sich bei der lokalen Debug-Konsole anmelden können. Weitere Informationen finden Sie unter IPC: Verwalten von lokalen Bereitstellungen und Komponenten. Fehlerbehebungen und Verbesserungen • Behebt ein Problem, bei dem abhängige Komponenten nicht reagieren würden, wenn ihre harten Abhängigkeiten in bestimmten Szenarien neu gestartet oder deren Status geändert werden. • Verbessert Fehlermeldungen, die das Core-Gerät an den AWS IoT Greengrass Cloud-Service meldet, wenn eine Bereitstellung fehlschlägt. • Behebt ein Problem, bei dem der Greengrass-Kern in bestimmten Szenarien zweimal eine Objektbereitstellung angewendet hat, wenn der Kern neu gestartet wird.

Komponente	Details
	• Zusätzliche kleinere Korrekturen und Verbesserungen. Weitere Informationen finden Sie in den Versionen auf GitHub.
MQTT-5-Broker (EMQX)	Version 1.0.0 der neuen EMQX MQTT 5-Brokerkomponente ist verfügbar. Neue Features • Fügt Unterstützung für den lokalen EMQX MQTT 5-Broker hinzu. Client-Geräte können eine Verbindung zu diesem MQTT-Broker herstellen, um mit einem Core-Gerät mithilfe von MQTT 5-Funktionen zu kommunizieren.
Schattenmanager	Version 2.2.0 der Shadow Manager-Komponente ist verfügbar. Neue Features • Fügt Unterstützung für den lokalen Schattenservice über die lokale Veröffentlichungs-/Abonnementschnittstelle hinzu. Sie können jetzt mit dem lokalen Message Broker für das Veröffentlichen/Abonnement zu Shadow-MQTT-Themen kommunizieren, um Shadows auf dem Core-Gerät abzurufen, zu aktualisieren und zu löschen. Mit dieser Funktion können Sie Client-Geräte mit dem lokalen Schattenservice verbinden, indem Sie die MQTT-Brücke verwenden, um Nachrichten zu Schattenthemen zwischen Client-Geräten und der lokalen Veröffentlichungs-/Abonnementschnittstelle weiterzuleiten. Diese Funktion erfordert v2.6.0 oder höher der Greengrass-Kernkomponente . Um Client-Geräte mit dem lokalen Schattenservice zu verbinden, müssen Sie auch v2.2.0 oder höher der MQTT-Bridge-Komponente verwenden. • Fügt die `direction` Option hinzu, die Sie konfigurieren können, um die Richtung für die Synchronisierung von Schatten zwischen dem lokalen Schattenservice und der anzupassenAWS Cloud. Sie können diese Option konfigurieren, um die Bandbreite und Verbindungen zum zu reduzierenAWS Cloud.

Komponente	Details
Client-Geräte-Authentifizierung	Version 2.2.0 der Clientgerät-Authentifizierungskomponente ist verfügbar. Neue Features • Fügt Unterstützung für benutzerdefinierte Komponenten hinzu, um IPC-Operationen (Interprocess Communication) aufzurufen, um Client-Geräte zu authentifizieren und zu autorisieren. Sie können diese Operationen beispielsweise in einer benutzerdefinierten MQTT-Brokerkomponente verwenden. Weitere Informationen finden Sie unter IPC: Authentifizieren und Autorisieren von Client-Geräten. • Fügt die `threadPoolSize` Optionen `maxActiveAuthTokens` , und hinzu, die Sie konfigurieren können`cloudQueueSize` , um die Leistung dieser Komponente zu optimieren.
MQTT-Brücke	Version 2.2.0 der MQTT-Bridge-Komponente ist verfügbar. Neue Features • Fügt Unterstützung für MQTT-Themen-Platzhalter (# und +) hinzu, wenn Sie lokales Veröffentlichen/Abonnieren als Quell-Message-Broker angeben. Diese Funktion erfordert v2.6.0 oder höher der Greengrass-Kernkomponente . • Fügt die `targetTopicPrefix` Option hinzu, die Sie angeben können, um die MQTT-Brücke so zu konfigurieren, dass dem Zielthema beim Weiterleiten einer Nachricht ein Präfix hinzugefügt wird.

Komponente	Details
Greengrass-CLI	Version 2.6.0 der Greengrass-CLI ist verfügbar. Neue Features • Fügt Unterstützung für benutzerdefinierte Komponenten hinzu, um IPC-Operationen (Interprocess Communication) aufzurufen, die die Greengrass-CLI verwendet. Sie können diese IPC-Operationen verwenden, um lokale Bereitstellungen zu verwalten, Komponent endetails anzuzeigen und ein Passwort zu generieren, mit dem Sie sich bei der lokalen Debug-Konsole anmelden können. Weitere Informati onen finden Sie unter IPC: Verwalten von lokalen Bereitstellungen und Komponenten. Fehlerbehebungen und Verbesserungen • Zusätzliche kleinere Korrekturen und Verbesserungen.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.5.6 am 31. Mai 2022

Diese Version enthält Version 2.5.6 der Greengrass-Kernkomponente und Version 2.2.4 der Log-Manager-Komponente.

Veröffentlichungsdatum: 31. Mai 2022

Versionsdetails

- Aktualisierungen öffentlicher Komponenten

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind AWSvon bereitgestellte Komponenten aufgeführt, die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden

neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren CoreAWS-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente einzubeziehen. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.5.6 des Greengrass-Kernus ist verfügbar. Neue Features • Fügt Unterstützung für Hardware-Sicherheitsmodule hinzu, die ECC-Schlüssel verwenden. Sie können ein Hardware-Sicherheitsmodul (HSM) verwenden, um den privaten Schlüssel und das Zertifikat des Geräts sicher zu speichern. Weitere Informationen finden Sie unter Integration von Hardware-Sicherheit. Fehlerbehebungen und Verbesserungen • Behebt ein Problem, bei dem die Bereitstellung in bestimmten Szenarien nie abgeschlossen wird, wenn Sie eine Komponente mit einem fehlerhaften Installationsskript bereitstellen. • Verbessert die Leistung beim Start. • Zusätzliche kleinere Korrekturen und Verbesserungen.
Protokoll manager	Version 2.2.4 der Log Manager-Komponente ist verfügbar. Fehlerbehebungen und Verbesserungen • Verbessert die Stabilität beim Umgang mit ungültigen Konfigurationen. • Zusätzliche kleinere Korrekturen und Verbesserungen.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.5.5 am 6. April 2022

Diese Version enthält Version 2.5.5 der Greengrass-Kernkomponente.

Veröffentlichungsdatum: 6. April 2022

Versionsdetails

- Aktualisierungen öffentlicher Komponenten

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind AWSvon bereitgestellte Komponenten aufgeführt, die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren CoreAWS-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
> Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente einzubeziehen. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.5.5 des Greengrass-Kerns ist verfügbar.

Komponente	Details
	Neue Features • Fügt die `GG_ROOT_CA_PATH` Umgebungsvariable für Komponenten hinzu, sodass Sie in benutzerdefinierten Komponenten auf das Stammzertifizierungsstellenzertifikat (CA) zugreifen können. Fehlerbehebungen und Verbesserungen • Fügt Unterstützung für Windows-Geräte hinzu, die eine andere Anzeigesprache als Englisch verwenden. • Aktualisiert, wie der Greengrass-Kern boolesche Installationsargumente analysiert, sodass Sie ein boolesches Argument ohne boolesche n Wert angeben können, um einen `true` Wert anzugeben. Sie können jetzt beispielsweise angeben, `--provision` anstatt mit automatischer Ressourcenbereitstellung `--provision true` zu installieren. • Behebt ein Problem, bei dem das Core-Gerät seinen Status nach der Bereitstellung in bestimmten Szenarien nicht an den AWS IoT Greengrass Cloud-Service meldete. • Zusätzliche kleinere Korrekturen und Verbesserungen.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.5.4 am 23. März 2022

Diese Version enthält Version 2.5.4 der Greengrass-Kernkomponente und Version 2.0.10 der Lambda-Launcher-Komponente.

Veröffentlichungsdatum: 23. März 2022

Versionsdetails

- Aktualisierungen öffentlicher Komponenten

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind AWSvon bereitgestellte Komponenten aufgeführt, die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren CoreAWS-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
> Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente einzubeziehen. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.5.4 des Greengrass-Kerns ist verfügbar. Fehlerbehebungen und Verbesserungen • Allgemeine Fehlerbehebungen und Verbesserungen.
Lambda-Launcher	Version 2.0.10 der Lambda-Launcher-Komponente ist verfügbar. Fehlerbehebungen und Verbesserungen • Allgemeine Fehlerbehebungen und Verbesserungen.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.5.3 am 6. Januar 2022

Diese Version enthält Version 2.5.3 der Greengrass-Kernkomponente und die neue PKCS#11-Anbieterkomponente.

Veröffentlichungsdatum: 6. Januar 2022

Veröffentlichungs-Merkmale

- Hardware-Sicherheitsintegration – Sie können die -AWS IoT GreengrassCore-Software jetzt so konfigurieren, dass sie einen privaten Schlüssel und ein Zertifikat verwendet, die Sie sicher in einem Hardware-Sicherheitsmodul (HSM) speichern. Weitere Informationen finden Sie unter Integration von Hardware-Sicherheit.

Versionsdetails

- Aktualisierungen öffentlicher Komponenten

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind AWSvon bereitgestellte Komponenten aufgeführt, die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren AWS-Core-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
> Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente einzubeziehen. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.5.3 des Greengrass-Kerns ist verfügbar.

Komponente	Details
	Neue Features • Fügt Unterstützung für die Hardware-Sicherheitsintegration hinzu. Sie können ein Hardware-Sicherheitsmodul (HSM) verwenden, um den privaten Schlüssel und das Zertifikat des Geräts sicher zu speichern . Weitere Informationen finden Sie unter Integration von Hardware-Sicherheit. Fehlerbehebungen und Verbesserungen • Behebt ein Problem mit Laufzeitausnahmen, während der Kern MQTT-Verbindungen mit herstelltAWS IoT Core.
PKCS #11-Anbieter	Version 2.0.0 der PKCS#11-Anbieterkomponente ist verfügbar. Neue Features • Fügt Unterstützung für die Hardware-Sicherheitsintegration hinzu. Sie können ein Hardware-Sicherheitsmodul (HSM) verwenden, um den privaten Schlüssel und das Zertifikat des Geräts sicher zu speichern . Weitere Informationen finden Sie unter Integration von Hardware-Sicherheit.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.5.2 am 3. Dezember 2021

Diese Version enthält Version 2.5.2 der Greengrass-Kernkomponente.

Veröffentlichungsdatum: 3. Dezember 2021

Versionsdetails

- Aktualisierungen öffentlicher Komponenten

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind AWSvon bereitgestellte Komponenten aufgeführt, die neue und aktualisierte Funktionen enthalten.

Important

Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren AWS-Core-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente anzugeben. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.5.2 des Greengrass-Kerns ist verfügbar. Fehlerbehebungen und Verbesserungen • Behebt ein Problem, bei dem der Windows-Service nach der Aktualisierung des Greengrass-Kerns nicht erneut gestartet werden kann, nachdem Sie ihn angehalten oder das Gerät neu gestartet haben.
AWS IoT Device Defender	Version 3.0.1 der AWS IoT Device Defender Komponente ist verfügbar. Diese Version der AWS IoT Device Defender Komponente erwartet andere Konfigurationsparameter als Version 2.x. Wenn Sie eine nicht standardmäßige Konfiguration für Version 2.x verwenden und ein Upgrade von v2.x auf v3.x durchführen möchten, müssen Sie die Konfiguration der Komponente aktualisieren. Weitere Informationen finden Sie unter AWS IoT Device Defender Komponentenkonfiguration . Neue Features • Fügt Unterstützung für -Core-Geräte hinzu, auf denen Windows ausgeführt wird.

Komponente	Details
	• Ändert den Komponententyp von einer Lambda-Komponente in eine generische Komponente. Diese Komponente hängt jetzt nicht mehr von der Legacy-Abonnement-Routerkomponente ab, um Abonnements zu erstellen. • Fügt den neuen `UseInstaller` Konfigurationsparameter hinzu, mit dem Sie optional das Installationsskript deaktivieren können, das Komponentenabhängigkeiten installiert.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.5.1 am 23. November 2021

Diese Version enthält Version 2.5.1 der Greengrass-Kernkomponente.

Veröffentlichungsdatum: 23. November 2021

Versionsdetails

- Aktualisierungen öffentlicher Komponenten

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind AWSvon bereitgestellte Komponenten aufgeführt, die neue und aktualisierte Funktionen enthalten.

Important

Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren AWS-Core-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre

bevorzugte Version dieser Komponente anzugeben. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.5.1 des Greengrass-Kerns ist verfügbar. Fehlerbehebungen und Verbesserungen • Fügt Unterstützung für 32-Bit-Versionen der Java Runtime Environment (JRE) unter Windows hinzu. • Ändert das Entfernungsverhalten von Objektgruppen für -Core-Geräte, deren AWS IoT Richtlinie die -`greengrass:ListThingGroupsForCoreDevice` Berechtigung nicht erteilt. Mit dieser Version wird die Bereitstellung fortgesetzt, protokolliert eine Warnung und entfernt keine Komponenten, wenn Sie das Core-Gerät aus einer Objektgruppe entfernen. Weitere Informationen finden Sie unter Bereitstellen von AWS IoT Greengrass Komponenten auf Geräten. • Behebt ein Problem mit Systemumgebungsvariablen, die der Greengrass-Kernus Greengrass-Komponentenprozessen zur Verfügung stellt. Sie können jetzt eine Komponente neu starten, damit sie die neuesten Systemumgebungsvariablen verwendet.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.5.0 am 12. November 2021

Diese Version enthält Version 2.5.0 der Greengrass-Kernkomponente, neue von AWSbereitgestellte Komponenten und Updates für AWSvon bereitgestellte Komponenten.

Veröffentlichungsdatum: 12. November 2021

Veröffentlichungs-Merkmale

- Windows-Geräteunterstützung – Sie können die -AWS IoT GreengrassCore-Software jetzt auf Geräten ausführen, auf denen Windows-Betriebssysteme ausgeführt werden. Weitere

Informationen finden Sie unter Unterstützte Plattformen und Anforderungen und Greengrass-Funktionskompatibilität nach Betriebssystem.

- Neues Verhalten beim Entfernen von Objektgruppen – Sie können jetzt ein Core-Gerät aus einer Objektgruppe entfernen, um die Komponenten dieser Objektgruppe bei der nächsten Bereitstellung auf diesem Gerät zu entfernen.

  > **Important**
  >
  > Aufgrund dieser Änderung muss die AWS IoT Richtlinie eines Core-Geräts über die `-greengrass:ListThingGroupsForCoreDevice`Berechtigung verfügen. Wenn Sie das AWS IoT Greengrass-Core-Softwareinstallationsprogramm zur Bereitstellung von Ressourcen verwendet haben, erlaubt die AWS IoT Standardrichtlinie `greengrass:*`, einschließlich dieser Berechtigung. Weitere Informationen finden Sie unter Geräteauthentifizierung und -autorisierung für AWS IoT Greengrass.

- Hardware-Sicherheitsunterstützung – Sie können die -AWS IoT GreengrassCore-Software jetzt für die Verwendung eines Hardware-Sicherheitsmoduls (HSM) konfigurieren, sodass Sie den privaten Schlüssel und das Zertifikat des Geräts sicher speichern können. Weitere Informationen finden Sie unter Integration von Hardware-Sicherheit.
- HTTPS-Proxy-Unterstützung – Sie können jetzt die AWS IoT Greengrass -Core-Software so konfigurieren, dass eine Verbindung über HTTPS-Proxys hergestellt wird. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy.

Versionsdetails

- Plattform-Support-Updates
- Aktualisierungen öffentlicher Komponenten

Plattform-Support-Updates

Plattform	Details
Windows	AWS IoT Greengrass unterstützt jetzt die Ausführung der AWS IoT Greengras s Core-Software auf den folgenden Versionen von Windows: • Windows 10

Plattform	Details
	• Windows Server 2019 Weitere Informationen finden Sie unter Unterstützte Plattformen und Anforderungen und Greengrass-Funktionskompatibilität nach Betriebssystem.

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind AWSvon bereitgestellte Komponenten aufgeführt, die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren CoreAWS-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
> Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente einzubeziehen. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.5.0 des Greengrass-Kerns ist verfügbar. Neue Features • Fügt Unterstützung für -Core-Geräte hinzu, auf denen Windows ausgeführt wird.

Komponente	Details
	• Ändern Sie das Verhalten der Entfernung von Objektgruppen. Mit dieser Version können Sie ein Core-Gerät aus einer Objektgruppe entfernen , um die Komponenten dieser Objektgruppe in der nächsten Bereitste llung zu deinstallieren. Aufgrund dieser Änderung muss die AWS IoT Richtlinie eines Core-Geräts über die `-greengrass:ListThingGroupsForCoreDev ice` Berechtigung verfügen. Wenn Sie das AWS IoT Greengrass-Core-Softwareinstallationsprogramm zur Bereitstellung von Ressource n verwendet haben, erlaubt die AWS IoT Standardrichtlinie `greengras s:*` , einschließlich dieser Berechtigung. Weitere Informationen finden Sie unter Geräteauthentifizierung und -autorisierung für AWS IoT Greengrass. • Fügt Unterstützung für HTTPS-Proxy-Konfigurationen hinzu. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy. • Fügt den neuen `windowsUser` Konfigurationsparameter hinzu. Sie können diesen Parameter verwenden, um den Standardbenutzer anzugeben, der zum Ausführen von Komponenten auf einem Windows-Core-Gerät verwendet werden soll. Weitere Informationen finden Sie unter Konfigurieren Sie den Benutzer, der die Komponenten ausführt. • Fügt die neuen `httpClient` Konfigurationsoptionen hinzu, mit denen Sie HTTP-Anforderungs-Timeouts anpassen können, um die Leistung in langsamen Netzwerken zu verbessern. Weitere Informationen finden Sie im Konfigurationsparameter httpClient. Fehlerbehebungen und Verbesserungen • Behebt die Bootstrap-Lebenszyklusoption, um das Core-Gerät von einer Komponente aus neu zu starten. • Fügt Unterstützung für Bindestriche in Rezeptvariablen hinzu. • Behebt die IPC-Autorisierung für On-Demand-Lambda-Funktionsk omponenten. • Verbessert Protokollmeldungen und ändert unkritische Protokolle von `INFO` auf `DEBUG` Ebene, sodass Protokolle nützlicher sind.

Komponente	Details
	• Entfernt die `iot:DescribeCertificate` Berechtigung aus der Standard-Tokenaustauschrolle, die der Greengrass-Kern erstellt, wenn Sie die AWS IoT Greengrass Core-Software mit automatischer Bereitstellung installieren. Diese Berechtigung wird nicht vom Greengrass-Kern verwendet. • Behebt ein Problem, sodass das automatische Bereitstellungsskript die `-iam:GetPolicy` Berechtigung nicht benötigt, wenn für dieselbe Richtlinie verfügbar `iam:CreatePolicy` ist. • Zusätzliche kleinere Korrekturen und Verbesserungen.
Greengrass-CLI	Version 2.5.0 der Greengrass-CLI ist verfügbar. Neue Features • Fügt Unterstützung für -Core-Geräte hinzu, auf denen Windows ausgeführt wird. • Fügt den neuen `AuthorizedWindowsGroups` Konfigurationsparameter hinzu, den Sie angeben können, um Systemgruppen für die Verwendung der Greengrass-CLI auf Windows-Geräten zu autorisieren. • Fügt den `windowsUser` Parameter für lokale Bereitstellungen hinzu. Sie können diesen Parameter verwenden, um den Benutzer anzugeben, der zum Ausführen von Komponenten auf einem Windows-Core-Gerät verwendet werden soll.

Komponente	Details
CloudWatch - Metriken	Version 3.0.0 der CloudWatch Metrikkomponente ist verfügbar. Diese Version der CloudWatch Metrikkomponente erwartet andere Konfigurationsparameter als Version 2.x. Wenn Sie eine nicht standardmäßige Konfiguration für Version 2.x verwenden und ein Upgrade von v2.x auf v3.x durchführen möchten, müssen Sie die Konfiguration der Komponente aktualisieren. Weitere Informationen finden Sie unter CloudWatch Konfiguration der Metrikkomponente . Neue Features • Fügt Unterstützung für -Core-Geräte hinzu, auf denen Windows ausgeführt wird. • Ändert den Komponententyp von einer Lambda-Komponente in eine generische Komponente. Diese Komponente hängt jetzt nicht mehr von der Legacy-Abonnement-Routerkomponente ab, um Abonnements zu erstellen. • Fügt einen neuen `InputTopic` Konfigurationsparameter hinzu, um das Thema anzugeben, das die Komponente für den Empfang von Nachrichten abonniert. • Fügt einen neuen `OutputTopic` Konfigurationsparameter hinzu, um das Thema anzugeben, zu dem die Komponente Statusantworten veröffentlicht. • Fügt einen neuen `PubSubToIoTCore` Konfigurationsparameter hinzu, um anzugeben, ob AWS IoT Core MQTT-Themen veröffentlicht und abonniert werden sollen. • Fügt den neuen `UseInstaller` Konfigurationsparameter hinzu, mit dem Sie optional das Installationsskript deaktivieren können, das Komponentenabhängigkeiten installiert. Fehlerbehebungen und Verbesserungen Fügt Unterstützung für doppelte Zeitstempel in Eingabedaten hinzu.

Komponente	Details
Lambda-Manager	Version 2.2.0 der Lambda-Manager-Komponente ist verfügbar. Fehlerbehebungen und Verbesserungen • Behebt ein Problem, bei dem Lambda-Funktionen nach einem Neustart keine Protokolle schreiben konnten. • Behebt ein Problem, bei dem der Legacy-Abonnement-Router doppelte Nachrichten sendet, wenn das Thema Platzhalter enthält. • Behebt ein Problem, bei dem nicht angeheftete Lambda-Funktionen die Greengrass Interprocess Communication (IPC)-Bibliothek in der nicht verwenden konntenAWS IoT Device SDK.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.4.0 am 3. August 2021

Diese Version enthält Version 2.4.0 der Greengrass-Kernkomponente, neue von AWSbereitgestellte Komponenten und Updates für AWSvon bereitgestellte Komponenten.

Veröffentlichungsdatum: 3. August 2021

Veröffentlichungs-Merkmale

- Systemressourcenlimits – Die Greengrass-Kernkomponente unterstützt jetzt Systemressourcenlimits. Sie können die maximale CPU- und RAM-Auslastung konfigurieren, die die Prozesse jeder Komponente auf dem Core-Gerät verwenden können. Weitere Informationen finden Sie unter Konfigurieren Sie die Systemressourcenlimits für Komponenten.
- Komponenten anhalten/fortsetzen – Der Greengrass-Kern unterstützt jetzt das Anhalten und Fortsetzen von Komponenten. Sie können die Interprocess Communication (IPC)-Bibliothek verwenden, um benutzerdefinierte Komponenten zu entwickeln, die die Prozesse anderer Komponenten anhalten und fortsetzen. Weitere Informationen finden Sie unter PauseComponent und ResumeComponent.
- Installation mit AWS IoT Flottenbereitstellung – Verwenden Sie das neue AWS IoT Flottenbereitstellungs-Plugin, um die AWS IoT Greengrass -Core-Software auf Geräten zu installieren, die eine Verbindung zu herstellenAWS IoT, um erforderliche AWS Ressourcen bereitzustellen. Geräte verwenden zur Bereitstellung ein Anspruchszertifikat. Sie können das

Antragszertifikat während der Herstellung auf Geräten einbetten, sodass jedes Gerät bereitgestellt werden kann, sobald es online ist. Weitere Informationen finden Sie unter Installieren Sie die AWS IoT Greengrass Core-Software mit AWS IoT Flottenbereitstellung.

- Installation mit benutzerdefinierter Bereitstellung – Entwickeln Sie ein benutzerdefiniertes Bereitstellungs-Plugin, um die erforderlichen AWS Ressourcen bereitzustellen, wenn Sie die AWS IoT Greengrass -Core-Software auf Geräten installieren. Sie können eine Java-Anwendung erstellen, die während der Installation ausgeführt wird, um Greengrass-Core-Geräte für Ihren benutzerdefinierten Anwendungsfall einzurichten. Weitere Informationen finden Sie unter Installieren Sie die AWS IoT Greengrass Core-Software mit benutzerdefinierter Ressourcenbereitstellung.

Versionsdetails

- Aktualisierungen öffentlicher Komponenten

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind AWSvon bereitgestellte Komponenten aufgeführt, die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren AWS-Core-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
> Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente anzugeben. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.4.0 des Greengrass-Kerns ist verfügbar. Neue Features • Fügt Unterstützung für Systemressourcenlimits hinzu. Sie können die maximale CPU- und RAM-Auslastung konfigurieren, die die Prozesse jeder Komponente auf dem Core-Gerät verwenden können. Weitere Informationen finden Sie unter Konfigurieren Sie die Systemressourcenlimits für Komponenten. • Fügt IPC-Operationen hinzu, um Komponenten anzuhalten und fortzusetzen. Weitere Informationen finden Sie unter PauseComponent und ResumeComponent. • Fügt Unterstützung für die Bereitstellung von Plugins hinzu. Sie können eine JAR-Datei angeben, die während der Installation ausgeführt werden soll, um die erforderlichen AWS Ressourcen für ein Greengrass-Core-Gerät bereitzustellen. Der Greengrass-Kern enthält eine Schnittstelle, die Sie implementieren können, um benutzerdefinierte Bereitstellungs-Plugins zu entwickeln. Weitere Informationen finden Sie unter Installieren Sie die AWS IoT Greengrass Core-Software mit benutzerdefinierter Ressourcenbereitstellung. • Fügt dem AWS IoT Greengrass Installationsprogramm für Core-Software das optionale Argument `thing-name-policy` hinzu. Sie können diese Option verwenden, um eine vorhandene oder benutzerdefinierte AWS IoT Richtlinie anzugeben, wenn Sie die AWS IoT Greengrass Core-Software mit automatischer Ressourcenbereitstellung installieren. Fehlerbehebungen und Verbesserungen • Aktualisiert die Protokollierungskonfiguration beim Start. Dies behebt ein Problem, bei dem die Protokollierungskonfiguration beim Start nicht angewendet wurde. • Aktualisiert den Symlink des -Kernladers so, dass er während der Installation auf den Komponentenspeicher im Greengrass-Stammordner verweist. Mit diesem Update können Sie die JAR-Datei und andere Kernartefakte löschen, die Sie bei der Installation der AWS IoT Greengrass Core-Software herunterladen.

Komponente	Details
	• Zusätzliche kleinere Korrekturen und Verbesserungen. Weitere Informationen finden Sie in den Versionen auf GitHub.
Greengrass-CLI	Version 2.4.0 der Greengrass-CLI ist verfügbar. Neue Features • Fügt Unterstützung für Systemressourcenlimits hinzu. Wenn Sie eine lokale Bereitstellung erstellen, können Sie die maximale CPU- und RAM-Auslastung konfigurieren, die die Prozesse jeder Komponente auf dem Core-Gerät verwenden können. Weitere Informationen finden Sie unter Konfigurieren Sie die Systemressourcenlimits für Komponenten und im Befehl Bereitstellung erstellen .
AWS IoT Flottenbereitstellung nach Anspruch	Das AWS IoT Flottenbereitstellung nach Anspruchs-Plugin ist jetzt verfügbar. Weitere Informationen finden Sie unter Installieren Sie die AWS IoT Greengrass Core-Software mit AWS IoT Flottenbereitstellung. Neue Features • Fügt Unterstützung für die Installation der AWS IoT Greengrass Core-Software mit AWS IoT Flottenbereitstellung hinzu. Während der Installation stellen Geräte eine Verbindung zu her, AWS IoT um die erforderlichen AWS Ressourcen bereitzustellen und Gerätezertifikate herunterzuladen, die für den regulären Betrieb verwendet werden.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.3.0 am 29. Juni 2021

Diese Version enthält Version 2.3.0 der Greengrass-Kernkomponente.

Veröffentlichungsdatum: 29. Juni 2021

Veröffentlichungs-Merkmale

- Unterstützung für große Konfigurationen – Die Greengrass-Kernkomponente unterstützt jetzt Bereitstellungsdokumente mit bis zu 10 MB. Sie können jetzt größere Konfigurationsaktualisierungen für Greengrass-Komponenten bereitstellen.

> **Note**
>
> Um diese Funktion verwenden zu können, muss die AWS IoT Richtlinie eines Core-Geräts die `-greengrass:GetDeploymentConfiguration`Berechtigung zulassen. Wenn Sie das AWS IoT Greengrass -Core-Softwareinstallationsprogramm zur Bereitstellung von Ressourcen verwendet haben, erlaubt die AWS IoT Richtlinie Ihres Core-Geräts `greengrass:*`, einschließlich dieser Berechtigung. Weitere Informationen finden Sie unter Geräteauthentifizierung und -autorisierung für AWS IoT Greengrass.

Versionsdetails

- Aktualisierungen öffentlicher Komponenten

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind AWSvon bereitgestellte Komponenten aufgeführt, die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren AWS-Core-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
> Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente anzugeben. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.3.0 des Greengrass-Kernus ist verfügbar. Neue Features • Fügt Unterstützung für Bereitstellungskonfigurationsdokumente bis zu 10 MB hinzu, von 7 KB (für Bereitstellungen, die auf Objekte abzielen) oder 31 KB (für Bereitstellungen, die auf Objektgruppen abzielen). Um diese Funktion verwenden zu können, muss die AWS IoT Richtlinie eines Core-Geräts die -`greengrass:GetDeploymentConfiguration` Berechtigung zulassen. Wenn Sie das AWS IoT Greengrass -Core-Softwareinstallationsprogramm zur Bereitstellung von Ressourcen verwendet haben, erlaubt die AWS IoT Richtlinie Ihres Core-Geräts `greengrass:*` , einschließlich dieser Berechtigung. Weitere Informationen finden Sie unter Geräteauthentifizierung und -autorisierung für AWS IoT Greengrass. • Fügt die `iot:thingName` Rezeptvariable hinzu. Sie können diese Rezeptvariable verwenden, um den Namen des AWS IoT Objekts des Core-Geräts in einem Rezept abzurufen. Weitere Informationen finden Sie unter Rezeptvariablen. Fehlerbehebungen und Verbesserungen • Zusätzliche kleinere Korrekturen und Verbesserungen. Weitere Informationen finden Sie in den Versionen auf GitHub.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.2.0 am 18. Juni 2021

Diese Version enthält Version 2.2.0 der Greengrass-Kernkomponente, neue von AWSbereitgestellte Komponenten und Updates für AWSvon bereitgestellte Komponenten.

Veröffentlichungsdatum: 18. Juni 2021

Veröffentlichungs-Merkmale

- Unterstützung von Client-Geräten – Mit den neuen von bereitgestellten ClientAWS-Gerätekomponenten können Sie Client-Geräte mithilfe von Cloud Discovery mit Ihren Core-Geräten verbinden. Sie können Client-Geräte mit synchronisieren AWS IoT Core und in Greengrass-Komponenten mit Client-Geräten interagieren. Weitere Informationen finden Sie unter Interagieren mit lokalen IoT-Geräten.
- Lokaler Schattenservice – Die neue Shadow-Manager-Komponente ermöglicht den lokalen Schattenservice auf Ihren -Core-Geräten. Sie können diesen Schattenservice verwenden, um mit lokalen Schatten zu interagieren, während Sie offline sind, indem Sie die Greengrass Interprocess Communication (IPC)-Bibliotheken in der verwendenAWS IoT Device SDK. Sie können die Schattenmanagerkomponente auch verwenden, um lokale Schattenzustände mit zu synchronisierenAWS IoT Core. Weitere Informationen finden Sie unter Interagieren mit Geräteschatten.

Versionsdetails

- Aktualisierungen öffentlicher Komponenten

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind AWSvon bereitgestellte Komponenten aufgeführt, die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren CoreAWS-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
> Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente anzugeben. Weitere Informationen zum

Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.2.0 des Greengrass-Kerns ist verfügbar. Neue Features • Fügt IPC-Operationen für die lokale Schattenverwaltung hinzu. Fehlerbehebungen und Verbesserungen • Reduziert die Größe der JAR-Datei. • Reduziert die Speichernutzung. • Behebt Probleme, bei denen die Protokollkonfiguration in bestimmten Fällen nicht aktualisiert wurde. • Zusätzliche kleinere Korrekturen und Verbesserungen. Weitere Informationen finden Sie in den Versionen auf GitHub.
Schattenmanager	Version 2.0.0 der neuen Shadow Manager-Komponente ist verfügbar. Neue Features • Fügt Unterstützung für klassische und benannte Schatten hinzu. • Fügt Unterstützung für die lokale Schattenverwaltung mithilfe von IPC hinzu. • Fügt Unterstützung für die Schattensynchronisierung mit hinzuAWS IoT Core.
Client-Geräte-Authentifizierung	Version 2.0.0 der neuen Clientgerät-Authentifizierungskomponente ist verfügbar. Neue Features • Fügt Unterstützung für Greengrass-Clientgeräte hinzu, bei denen es sich um lokale IoT-Geräte handelt, die über MQTT eine Verbindung zu einem Core-Gerät herstellen.

Komponente	Details
	• Fügt Unterstützung für die Authentifizierung und Autorisierung von Client-Geräten und deren MQTT-Aktionen hinzu.
Moquette-MQTT-Broker	Version 2.0.0 der neuen Moquette MQTT-Brokerkomponente ist verfügbar. Neue Features • Fügt Unterstützung für einen lokalen Moquette MQTT-Broker hinzu, der die Kommunikation mit Client-Geräten übernimmt.
MQTT-Brücke	Version 2.0.0 der neuen MQTT-Bridge-Komponente ist verfügbar. Neue Features • Fügt Unterstützung für das Weiterleiten von Nachrichten zwischen dem lokalen MQTT-Broker, dem lokalen Greengrass-Publish/Subscribe-Broker und dem AWS IoT Core MQTT-Broker hinzu.
IP-Detektor	Version 2.0.0 der neuen IP-Detektorkomponente ist verfügbar. Neue Features • Fügt Unterstützung hinzu, um die lokalen MQTT-Broker-Endpunkte eines Core-Geräts an den AWS IoT Greengrass Cloud-Service zu melden, damit Client-Geräte eine Verbindung herstellen können.
Protokoll manager	Version 2.1.1 der Log Manager-Komponente ist verfügbar. Fehlerbehebungen und Verbesserungen • Behebt ein Problem, bei dem die Systemprotokollkonfiguration in bestimmten Fällen nicht aktualisiert wurde.
DLR-Objek terkennung	Version 2.1.2 der DLR-Objekterkennung ist verfügbar. Fehlerbehebungen und Verbesserungen • Behebt ein Problem mit der Bildskalierung, das zu ungenauen Begrenzungsrahmen in den Inferenzergebnissen der DLR-Beispielobjekt erkennung führte.

Komponente	Details
TensorFlow Erkennung von Lite-Objekten	Version 2.1.1 der TensorFlow Lite-Objekterkennung ist verfügbar. Fehlerbehebungen und Verbesserungen • Behebt ein Problem mit der Bildskalierung, das zu ungenauen Begrenzungsrahmen in den Inferenzergebnissen der TensorFlow Lite-Objekterkennung führte.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.1.0 am 26. April 2021

Diese Version enthält Version 2.1.0 der Greengrass-Kernkomponente und aktualisiert AWSvon bereitgestellte Komponenten.

Veröffentlichungsdatum: 26. April 2021

Veröffentlichungs-Merkmale

- Integration von Docker Hub und Amazon Elastic Container Registry (Amazon ECR) – Mit der neuen Docker-Anwendungsmanager-Komponente können Sie öffentliche oder private Images von Amazon ECR herunterladen. Sie können diese Komponente auch verwenden, um öffentliche Images von Docker Hub und herunterzuladenAWS Marketplace. Weitere Informationen finden Sie unter Führen Sie einen Docker-Container aus.
- Dockerfile- und Docker-Images für AWS IoT Greengrass Core-Software – Sie können das Greengrass Docker-Image verwenden, um AWS IoT Greengrass in einem Docker-Container auszuführen, der Amazon Linux 2 als Basisbetriebssystem verwendet. Sie können die AWS IoT Greengrass Dockerfile auch verwenden, um Ihr eigenes Greengrass-Image zu erstellen. Weitere Informationen finden Sie unter Ausführen von AWS IoT Greengrass Core-Software in einem Docker-Container.
- Unterstützung für zusätzliche Frameworks und Plattformen für Machine Learning – Sie können Machine-Learning-Beispielinferenzkomponenten bereitstellen, die vortrainierte Modelle verwenden, um eine Bildklassifizierung und Objekterkennung mit TensorFlow Lite 2.5.0 und DLR 1.6.0 durchzuführen. Diese Version erweitert auch die Unterstützung für Machine Learning für Armv8 (AArch64)-Geräte. Weitere Informationen finden Sie unter Durchführen von Machine Learning-Inferenzen.

Versionsdetails

- Plattform-Support-Updates
- Aktualisierungen öffentlicher Komponenten

Plattform-Support-Updates

Plattform	Details
Docker	Ein Dockerfile und ein Docker-Image für AWS IoT Greengrass sind jetzt verfügbar. Dockerfile AWS IoT Greengrass stellt ein Dockerfile bereit, um ein Container-Image zu erstellen, auf dem AWS IoT Greengrass Core-Software und Abhängigkeiten auf einem Amazon Linux 2 (x86_64)-Basis-Image installiert sind. Sie können das Basis-Image im Dockerfile so ändern, dass es AWS IoT Greengrass auf einer anderen Plattformarchitektur ausgeführt wird. Docker-Image AWS IoT Greengrass bietet ein vorgefertigtes Docker-Image, auf dem AWS IoT Greengrass Core-Software und Abhängigkeiten auf einem Amazon Linux 2 (x86_64)-Basis-Image installiert sind. Weitere Informationen finden Sie unter Ausführen von AWS IoT Greengrass Core-Software in einem Docker-Container.

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind AWSvon bereitgestellte Komponenten aufgeführt, die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden

neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren AWS-Core-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente anzugeben. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.1.0 des Greengrass-Kerns ist verfügbar. Neue Features • Unterstützt das Herunterladen von Docker-Images aus privaten Repositorys in Amazon ECR. • Fügt die folgenden Parameter hinzu, um die MQTT-Konfiguration auf -Core-Geräten anzupassen: • `maxInFlightPublishes` – Die maximale Anzahl unbestätigter MQTT QoS 1-Nachrichten, die gleichzeitig in Bewegung sein können. • `maxPublishRetry` – Die maximale Anzahl von Wiederhol ungsversuchen für Nachrichten, die nicht veröffentlicht werden können. • Fügt den `fleetstatusservice` Konfigurationsparameter hinzu, um das Intervall zu konfigurieren, in dem das Core-Gerät den Gerätestatus in der veröffentlichtAWS Cloud. • Zusätzliche kleinere Korrekturen und Verbesserungen. Weitere Informati onen finden Sie in den Versionen auf GitHub. Fehlerbehebungen und Verbesserungen • Behebt ein Problem, das dazu führte, dass Schattenbereitstellungen dupliziert wurden, wenn der Kern neu gestartet wurde.

Komponente	Details
	• Behebt ein Problem, das dazu führte, dass der Kern abstürzte, wenn er auf eine Ausnahme beim Laden des Services gestoßen ist. • Verbessert die Auflösung der Komponentenabhängigkeit, um eine Bereitstellung mit einer Zirkelabhängigkeit fehlschlagen zu lassen. • Behebt ein Problem, das verhindert hat, dass eine Plugin-Komponente erneut bereitgestellt wurde, wenn diese Komponente zuvor vom Core-Gerät entfernt wurde. • Es wurde ein Problem behoben, das dazu führte, dass die `HOME` Umgebungsvariable für Lambda-Komponenten oder für Komponenten, die als Stamm ausgeführt werden, auf das */greengrass/v2* `/work` Verzeichnis festgelegt wurde. Die `HOME` Variable ist jetzt korrekt auf das Stammverzeichnis für den Benutzer festgelegt, der die Komponente ausführt. • Zusätzliche kleinere Korrekturen und Verbesserungen. Weitere Informationen finden Sie in den Versionen auf GitHub.
Docker-Anwendungsmanager	Version 2.0.0 der neuen Docker-Anwendungsmanager-Komponente ist verfügbar. Neue Features • Verwaltet Anmeldeinformationen zum Herunterladen von Images aus privaten Repositorys in Amazon ECR. • Lädt öffentliche Images von Amazon ECR, Docker Hub und herunterAWS Marketplace.
Lambda-Launcher	Version 2.0.4 der Lambda-Launcher-Komponente ist verfügbar. Fehlerbehebungen und Verbesserungen • Behebt ein Problem, bei dem die Komponente nicht korrekt `AddGroupOwner` an den Lambda-Funktionscontainer übergeben wurde.

Komponente	Details
Legacy-Abonnement-Router	Version 2.1.0 der Legacy-Abonnement-Routerkomponente ist verfügbar. Fehlerbehebungen und Verbesserungen • Fügt Unterstützung für die Angabe von Komponentennamen anstelle von ARNs für `source` und hinzu`target`. Wenn Sie einen Komponentennamen für ein Abonnement angeben, müssen Sie das Abonnement nicht jedes Mal neu konfigurieren, wenn sich die Version der Lambda-Funktion ändert.
Lokale Debug-Konsole	Version 2.1.0 der lokalen Debug-Konsolenkomponente ist verfügbar. Neue Features • Verwendet HTTPS, um Ihre Verbindung zur lokalen Debug-Konsole zu sichern. HTTPS ist standardmäßig aktiviert. Fehlerbehebungen und Verbesserungen • Sie können Flashbar-Nachrichten im Konfigurationseditor verwerfen.
Protokollmanager	Version 2.1.0 der Log Manager-Komponente ist verfügbar. Fehlerbehebungen und Verbesserungen • Verwenden Sie Standardwerte für `logFileDirectoryPath` und `logFileRegex`, die für Greengrass-Komponenten funktionieren, die auf die Standardausgabe (stdout) und den Standardfehler (stderr) gedruckt werden. • Datenverkehr beim Hochladen von Protokollen in CloudWatch Protokolle korrekt über einen konfigurierten Netzwerk-Proxy weiterleiten. • Verarbeiten Sie Doppelpunktzeichen (`:`) in Protokollstreamnamen korrekt. CloudWatch Logs-Protokollstreamnamen unterstützen keine Doppelpunkte. • Vereinfachen Sie die Namen von Protokollstreams, indem Sie Objektgruppennamen aus dem Protokollstream entfernen. • Entfernen Sie eine Fehlermeldung, die während des normalen Verhaltens ausgegeben wird.

Komponente	Details
DLR-Bildklassifizierung	Version 2.1.1 der DLR-Bildklassifizierungskomponente ist verfügbar. Neue Features • Verwenden Sie Deep Learning Runtime v1.6.0. • Unterstützung für die Beispiel-Image-Klassifizierung auf Armv8 (AArch64)-Plattformen hinzufügen. Dadurch wird die Unterstützung für Machine Learning für Greengrass-Core-Geräte erweitert, auf denen NVIDIA Jetson ausgeführt wird, z. B. Jetson Nano. • Aktivieren Sie die Kameraintegration für Beispielinferenzen. Verwenden Sie den neuen `UseCamera` Konfigurationsparameter, um dem Beispiel-Inferenzcode den Zugriff auf die Kamera auf Ihrem Greengrass-Kerngerät und die lokale Ausführung von Inferenzen auf dem aufgenommenen Bild zu ermöglichen. • Unterstützung für die Veröffentlichung von Inferenzergebnissen in der hinzufügenAWS Cloud. Verwenden Sie den neuen `PublishResultsOnTopic` Konfigurationsparameter, um das Thema anzugeben, zu dem Sie Ergebnisse veröffentlichen möchten. • Fügen Sie den neuen `ImageDirectory` Konfigurationsparameter hinzu, mit dem Sie ein benutzerdefiniertes Verzeichnis für das Image angeben können, für das Sie Inferenzen durchführen möchten. Fehlerbehebungen und Verbesserungen • Schreiben Sie Inferenzergebnisse in die Komponentenprotokolldatei anstelle einer separaten Inferenzdatei. • Verwenden Sie das AWS IoT Greengrass-Core-Softwareprotokollierungsmodul, um die Komponentenausgabe zu protokollieren. • Verwenden Sie die AWS IoT Device SDK, um die Komponentenkonfiguration zu lesen und Konfigurationsänderungen anzuwenden.

Komponente	Details
DLR-Objekterkennung	Version 2.1.1 der DLR-Objekterkennungskomponente ist verfügbar. Neue Features • Verwenden Sie Deep Learning Runtime v1.6.0. • Unterstützung für die Erkennung von Beispielobjekten auf Armv8 (AArch64)-Plattformen hinzufügen. Dadurch wird die Unterstützung für Machine Learning für Greengrass-Core-Geräte erweitert, auf denen NVIDIA Jetson ausgeführt wird, z. B. Jetson Nano. • Aktivieren Sie die Kameraintegration für Beispielinferenzen. Verwenden Sie den neuen `UseCamera` Konfigurationsparameter, um dem Beispiel-Inferenzcode den Zugriff auf die Kamera auf Ihrem Greengrass-Kerngerät und die lokale Ausführung von Inferenzen auf dem aufgenommenen Bild zu ermöglichen. • Unterstützung für die Veröffentlichung von Inferenzergebnissen in der hinzufügenAWS Cloud. Verwenden Sie den neuen `PublishResultsOnTopic` Konfigurationsparameter, um das Thema anzugeben, zu dem Sie Ergebnisse veröffentlichen möchten. • Fügen Sie den neuen `ImageDirectory` Konfigurationsparameter hinzu, mit dem Sie ein benutzerdefiniertes Verzeichnis für das Image angeben können, für das Sie Inferenzen durchführen möchten. Fehlerbehebungen und Verbesserungen • Schreiben Sie Inferenzergebnisse in die Komponentenprotokolldatei anstelle einer separaten Inferenzdatei. • Verwenden Sie das AWS IoT Greengrass-Core-Softwareprotokollierungsmodul, um die Komponentenausgabe zu protokollieren. • Verwenden Sie die AWS IoT Device SDK, um die Komponentenkonfiguration zu lesen und Konfigurationsänderungen anzuwenden.

Komponente	Details
DLR-Bildklassifizierungsmodellspeicher	Version 2.1.1 der DLR-Bildklassifizierungsmodellspeicherkomponente ist verfügbar. Neue Features • Fügen Sie ein Beispiel für ein ResNet-50-Image-Klassifizierungsmodell für Armv8-Plattformen (AArch64) hinzu. Dadurch wird die Unterstützung für Machine Learning für Greengrass-Core-Geräte erweitert, auf denen NVIDIA Jetson ausgeführt wird, z. B. Jetson Nano.
Modellspeicher für die DLR-Objekterkennung	Version 2.1.1 der DLR-Objekterkennungsmodellspeicherkomponente ist verfügbar. Neue Features • Fügen Sie ein Beispiel für ein YOLOv3-Objekterkennungsmodell für Armv8-Plattformen (AArch64) hinzu. Dadurch wird die Unterstützung für Machine Learning für Greengrass-Core-Geräte erweitert, auf denen NVIDIA Jetson ausgeführt wird, z. B. Jetson Nano.
DLR-Installationsprogramm	Version 1.6.1 der DLR-Komponente ist verfügbar. Neue Features • Installieren Sie Deep Learning Runtime v1.6.0 und seine Abhängigkeiten. • Unterstützung für die Installation von DLR auf Armv8 (AArch64)-Plattformen hinzufügen. Dadurch wird die Unterstützung für Machine Learning für Greengrass-Core-Geräte erweitert, auf denen NVIDIA Jetson ausgeführt wird, z. B. Jetson Nano. Fehlerbehebungen und Verbesserungen • Installieren Sie die AWS IoT Device SDK in der virtuellen Umgebung, um die Komponentenkonfiguration zu lesen und Konfigurationsänderungen anzuwenden. • Zusätzliche kleinere Fehlerbehebungen und Verbesserungen.

Komponente	Details
TensorFlow Lite-Bildklassifizierung	Version 2.1.0 der neuen TensorFlow Lite-Bildklassifizierungskomponente ist verfügbar. Neue Features • Fügen Sie Unterstützung für die Inferenz der Bildklassifizierung mit TensorFlow Lite hinzu.
TensorFlow Erkennung von Lite-Objekten	Version 2.1.0 der neuen TensorFlow Lite-Objekterkennungskomponente ist verfügbar. Neue Features • Fügen Sie Unterstützung für die Inferenz der Objekterkennung mit TensorFlow Lite hinzu.
TensorFlow Modellspeicher für die Lite-Bildklassifizierung	Version 2.1.0 der neuen TensorFlow Lite-Bildklassifizierungsmodellspeicherkomponente ist verfügbar. Neue Features • Stellen Sie ein vortrainiertes quantisiertes MobileNet v1-Modell für die Inferenz der Beispielbildklassifizierung mit TensorFlow Lite bereit.
TensorFlow Modellspeicher für die Lite-Objekterkennung	Version 2.1.0 der neuen TensorFlow Lite-Objekterkennungsmodellspeicherkomponente ist verfügbar. Neue Features • Stellen Sie ein vortrainiertes Single Shot Detection (SSD)- MobileNet Modell bereit, das im COCO-Datensatz trainiert wurde, um eine Inferenz bei der Objekterkennung mit TensorFlow Lite zu erhalten.
TensorFlow Lite	Version 2.5.0 der neuen TensorFlow Lite-Komponente ist verfügbar. Neue Features • Installieren Sie TensorFlow Lite v1.6.0 und seine Abhängigkeiten in einer virtuellen Umgebung auf den Plattformen Armv7, Armv8 (AArch64) und x86_64.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.0.5 am 09. März 2021

Diese Version enthält Version 2.0.5 der Greengrass-Kernkomponente und aktualisiert AWSvon bereitgestellte Komponenten. Es behebt ein Problem mit der Netzwerk-Proxy-Unterstützung und ein Problem mit dem Greengrass-Datenebenen-Endpunkt in AWS den China-Regionen.

Veröffentlichungsdatum: 09. März 2021

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind AWSvon bereitgestellte Komponenten aufgeführt, die neue und aktualisierte Funktionen enthalten.

> **Important**
>
> Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren CoreAWS-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
> Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente einzubeziehen. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.0.5 des Greengrass-Kerns ist verfügbar.

Komponente	Details
	Fehlerbehebungen und Verbesserungen • Leitet den Datenverkehr korrekt über einen konfigurierten Netzwerk-Proxy weiter, wenn von bereitgestellte Komponenten heruntergeladen AWSwerden. • Verwenden Sie den richtigen Greengrass-Endpunkt auf Datenebene in AWS den China-Regionen.

Veröffentlichung: Softwareupdate für AWS IoT Greengrass Core v2.0.4 am 04. Februar 2021

Diese Version enthält Version 2.0.4 der Greengrass-Kernkomponente. Es enthält den neuen `greengrassDataPlanePort` Parameter zum Konfigurieren der HTTPS-Kommunikation über Port 443 und behebt Fehler. Die minimale IAM-Richtlinie erfordert jetzt und `iam:GetPolicysts:GetCallerIdentity`, wenn das AWS IoT Greengrass Core-Softwareinstallationsprogramm mit ausgeführt wird`--provision true`.

Veröffentlichungsdatum: 04. Februar 2021

Aktualisierungen öffentlicher Komponenten

In der folgenden Tabelle sind AWSvon bereitgestellte Komponenten aufgeführt, die neue und aktualisierte Funktionen enthalten.

Important

Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren AWS-Core-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre

bevorzugte Version dieser Komponente aufzunehmen. Weitere Informationen zum Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass-Kern	Version 2.0.4 des Greengrass-Kerns ist verfügbar. Neue Features • Aktiviert HTTPS-Datenverkehr über Port 443. Sie können den neuen `greengrassDataPlanePort` Konfigurationsparameter für Version 2.0.4 der -Kernkomponente verwenden, um die HTTPS-Kommunikation so zu konfigurieren, dass sie über Port 443 anstelle des Standardports 8443 übertragen wird. Weitere Informationen finden Sie unter Konfigurieren Sie HTTPS über Port 443. • Fügt die Arbeitspfad-Rezeptvariable hinzu. Sie können diese Rezeptvariable verwenden, um den Pfad zu den Arbeitsordnern der Komponenten abzurufen, mit denen Sie Dateien zwischen Komponenten und deren Abhängigkeiten freigeben können. Weitere Informationen finden Sie in der -Arbeitspfad-Rezeptvariable . Fehlerbehebungen und Verbesserungen • Verhindert die Erstellung der IAM-Rollenrichtlinie AWS Identity and Access Management (Token Exchange), wenn bereits eine Rollenrichtlinie vorhanden ist. Aufgrund dieser Änderung benötigt das Installationsprogramm jetzt die `iam:GetPolicy` und , `sts:GetCallerIdentity` wenn es mit ausgeführt wird`--provision true` . Weitere Informationen finden Sie unter Minimale IAM-Richtlinie für das Installationsprogramm zur Bereitstellung von Ressourcen. • Behebt den Abbruch einer Bereitstellung, die noch nicht erfolgreich registriert wurde, korrekt. • Aktualisiert die Konfiguration, um ältere Einträge mit neueren Zeitstempeln beim Rollback einer Bereitstellung zu entfernen.

Komponente	Details
	• Zusätzliche kleinere Korrekturen und Verbesserungen. Weitere Informationen finden Sie in den Versionen auf GitHub.

Migrieren von AWS IoT Greengrass Version 1

AWS IoT Greengrass Version 2 ist eine Hauptversion der -AWS IoT GreengrassCore-Software, APIs und der Konsole. AWS IoT Greengrass V2 führt mehrere Verbesserungen an einAWS IoT Greengrass V1, z. B. modulare Anwendungen, Bereitstellungen für große Geräteflotten und Unterstützung für zusätzliche Plattformen.

> **Note**
>
> Nach dem 30. Juni 2023 erhält es AWS IoT Greengrass Version 1 keine Feature-Updates, Verbesserungen, Fehlerbehebungen oder Sicherheitspatches mehr. Weitere Informationen finden Sie in der AWS IoT Greengrass V1 Wartungsrichtlinie . Wenn Sie verwendenAWS IoT Greengrass V1, empfehlen wir dringend, zu zu migrierenAWS IoT Greengrass V2.

Folgen Sie den Anweisungen in diesem Handbuch, um von AWS IoT Greengrass V1 zu zu migrierenAWS IoT Greengrass V2.

Kann ich meine V1-Anwendungen auf V2 ausführen?

Die meisten V1-Anwendungen können auf V2-Core-Geräten ausgeführt werden, ohne den Anwendungscode ändern zu müssen. Wenn Ihre V1-Anwendungen die folgende Funktion verwenden, können Sie sie nicht auf V2 ausführen.

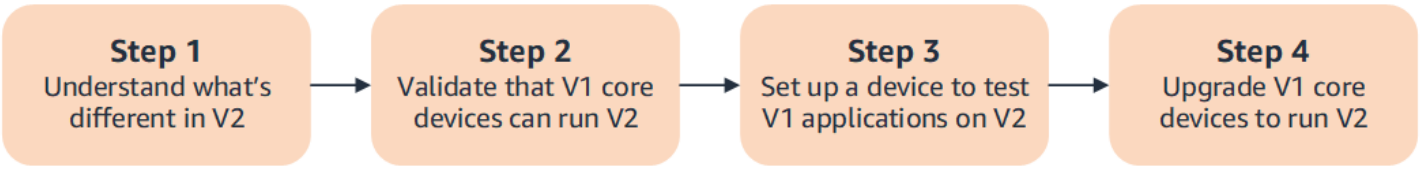
- Die C- und C++-Lambda-Funktionslaufzeiten

Wenn Ihre V1-Anwendungen eine der folgenden Funktionen verwenden, müssen Sie Ihren Anwendungscode so ändern, dass er die AWS IoT Device SDK V2 verwendet, um die Anwendungen auf auszuführenAWS IoT Greengrass V2.

- Interagieren mit dem lokalen Schattenservice
- Veröffentlichen von Nachrichten auf lokal verbundenen Geräten (Greengrass-Geräte)

Migrationsübersicht

Allgemein können Sie das folgende Verfahren verwenden, um Core-Geräte von AWS IoT Greengrass V1 auf zu aktualisierenAWS IoT Greengrass V2. Das genaue Verfahren, das Sie befolgen, hängt von den spezifischen Anforderungen für Ihre Umgebung ab.



1. Die Unterschiede zwischen V1 und V2 verstehen

   AWS IoT Greengrass V2 führt neue grundlegende Konzepte für Geräteflotten und bereitstellbare Software ein, und V2 vereinfacht mehrere Konzepte aus V1.

   Der AWS IoT Greengrass V2 Cloud-Service und die AWS IoT Greengrass Core-Software v2.x sind nicht abwärtskompatibel mit dem AWS IoT Greengrass V1 Cloud-Service und der AWS IoT Greengrass Core-Software v1.x. Daher können AWS IoT Greengrass V1 over-the-air (OTA)-Updates keine Core-Geräte von V1 auf V2 aktualisieren.

2. Überprüfen, ob V1-Core-Geräte V2 ausführen können

   Stellen Sie sicher, dass ein V1-Core-Gerät die AWS IoT Greengrass V2 Funktionen AWS IoT Greengrass Core-Software v2.x und ausführen kann. AWS IoT Greengrass V2 hat andere Geräteanforderungen als AWS IoT Greengrass V1.

3. Einrichten eines neuen Geräts zum Testen von V1-Anwendungen auf V2

   Um das Risiko für Ihre Geräte in der Produktion zu minimieren, erstellen Sie ein neues Gerät, um Ihre V1-Anwendungen auf V2 zu testen. Nachdem Sie die AWS IoT Greengrass Core-Software v2.x installiert haben, können Sie AWS IoT Greengrass V2 Komponenten erstellen und bereitstellen, um Ihre AWS IoT Greengrass V1 Anwendungen zu migrieren und zu testen.

4. Aktualisieren von V1-Core-Geräten zur Ausführung von V2

   Aktualisieren Sie ein vorhandenes V1-Core-Gerät, um die AWS IoT Greengrass V2 Komponenten AWS IoT Greengrass Core-Software v2.x und auszuführen. Um eine Flotte von Geräten von V1 zu V2 zu migrieren, wiederholen Sie diesen Schritt für jedes Gerät in der Flotte.

Unterschiede zwischen AWS IoT Greengrass V1 und AWS IoT Greengrass V2

AWS IoT Greengrass V2 führt neue grundlegende Konzepte für Geräte, Flotten und bereitstellbare Software ein. In diesem Abschnitt werden die V1-Konzepte beschrieben, die sich in V2 unterscheiden.

Konzepte und Terminologie von Greengrass

Konzept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Anwendungscode	In definieren Lambda-Fu nktionen die Software AWS IoT Greengrass V1, die auf -Core-Geräten ausgeführt wird. In jeder Greengrass-Gruppe definieren Sie Abonnements und lokale Ressourcen, die die Funktion verwendet. Für Lambda-Funktionen, die die AWS IoT Greengrass -Core-Software in einer container isierten Lambda-Laufzeitumg ebung ausführt, definieren Sie Containerparameter, z. B. Speicherlimits.	In sind Komponenten die Softwaremodule AWS IoT Greengrass V2, die auf -Core-Geräten ausgeführt werden. • Jede Komponente verfügt über ein Rezept, das die Metadaten, Parameter, Abhängigkeiten und Skripts der Komponente definiert , die bei jedem Schritt des Komponentenlebenszyklus ausgeführt werden sollen. • Das Rezept definiert auch die Artefakte der Komponente , bei denen es sich um Binärdateien wie Skripts, kompilierten Code und statische Ressourcen handelt. • Wenn Sie eine Komponent e auf einem Core-Gerä t bereitstellen, lädt das Core-Gerät das Komponent enrezept und die Artefakte

Konzept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
		herunter, um die Komponente auszuführen. Sie können Ihre V1-Lambda-Funktionen als Komponenten importieren, die in einer Lambda-Laufzeitumgebung in ausgeführt werden AWS IoT Greengrass V2. Wenn Sie die Lambda-Funktion importieren, geben Sie die Abonnements, lokalen Ressourcen und Containerparameter für die Funktion an. Weitere Informationen finden Sie unter Schritt 2: Erstellen und Bereitstellen von AWS IoT Greengrass V2 Komponenten zur Migration von AWS IoT Greengrass V1 Anwendungen. Weitere Informationen zum Erstellen von benutzerdefinierten Komponenten finden Sie unter Entwickeln von AWS IoT Greengrass Komponenten.

Konzept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
AWS IoT Greengrass - Gruppen und -Bereitstellungen	In definiert AWS IoT Greengrass V1eine Gruppe das Core-Gerät, die Einstellungen und Software für dieses Core-Gerät sowie die Liste der AWS IoT Objekte, die eine Verbindung zu diesem Core-Gerät herstellen können. Sie erstellen eine Bereitstellung, um die Konfiguration einer Gruppe an ein Core-Gerät zu senden.	In verwenden Sie Bereitstellungen AWS IoT Greengrass V2, um die Softwarekomponenten und Konfigurationen zu definieren, die auf -Core-Geräten ausgeführt werden. • Jede Bereitstellung zielt auf ein einzelnes Core-Gerät (was ein AWS IoT -Objekt ist) oder eine - AWS IoT Objektgruppe ab, die mehrere Core-Geräte enthalten kann. • Bereitstellungen in Objektgruppen erfolgen kontinuierlich. Wenn Sie also ein Core-Gerät zu einer Objektgruppe hinzufügen, erhält es die Softwarekonfiguration für diese Gruppe. Weitere Informationen finden Sie unter Bereitstellen von AWS IoT Greengrass Komponenten auf Geräten. In können Sie auch lokale Bereitstellungen mit der Greengrass-CLI erstellen AWS IoT Greengrass V2, um benutzerdefinierte Softwarekomponenten auf dem Gerät

Konzept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
		zu testen, auf dem Sie sie entwickeln. Weitere Informationen finden Sie unter Erstellen von AWS IoT Greengrass Komponenten.

Konzept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
AWS IoT Greengrass Core-Software	In ist AWS IoT Greengrass V1die AWS IoT Greengrass Core-Software ein einzelnes Paket, das die Software und alle ihre Funktionen enthält. Das Edge-Gerät, auf dem Sie die AWS IoT Greengrass Core-Software installieren, wird als Greengrass-Kern bezeichnet.	In ist AWS IoT Greengrass V2die - AWS IoT Greengrass Core-Software modular, sodass Sie auswählen können, was installiert werden soll, um den Speicherbedarf zu kontrollieren. • Die Greengrass-Kernkomponente ist die mindestens erforderliche Installation der AWS IoT Greengrass Core-Software. Das Edge-Gerät, auf dem Sie den Kern installieren, wird als Greengrass-Kerngerät bezeichnet. • Der Kern verwaltet Bereitstellungen, Orchestrierung und Lebenszyklusmanagement anderer Komponenten auf dem Core-Gerät. • Funktionen wie Stream Manager, Secret Manager und Log Manager sind Komponenten, die Sie nur bereitstellen, wenn Sie diese Funktionen benötigen. Weitere Informationen finden Sie unter AWSVon bereitgestellte Komponenten.

Konzept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Konnektoren	In sind Konnektoren vorgefert igte Module AWS IoT Greengrass V1, die Sie auf - AWS IoT Greengrass V1 Core-Geräten bereitstellen AWS, um mit lokaler Infrastru ktur, Geräteprotokollen und anderen Cloud-Services zu interagieren.	In stellt Greengrass-Kompone nten AWS IoT Greengras s V2 AWS bereit, die die Funktionalität von Konnektor en in V1 implementieren. Die folgenden AWS IoT Greengras s V2 Komponenten bieten Greengrass-V1-Konnektor-Funktionalität: • CloudWatch Metrikkom ponente • AWS IoT Device Defender Komponente • Firehose-Komponente • Adapterkomponente des Bol-RTU-Protokolls • Amazon SNS-Komponente Weitere Informationen finden Sie unter AWSVon bereitges tellte Komponenten.

Konzept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Verbundene Geräte (Greengrass-Geräte)	In sind verbundene Geräte AWS IoT Dinge AWS IoT Greengrass V1, die Sie einer Greengrass-Gruppe hinzufügen, um eine Verbindung zum Core-Gerät in dieser Gruppe herzustellen und über MQTT zu kommunizieren. Sie müssen diese Gruppe jedes Mal bereitstellen, wenn Sie ein verbundenes Gerät hinzufügen oder entfernen. Sie verwenden Abonnements, um Nachrichten zwischen verbundenen Geräten AWS IoT Coreund Anwendungen auf dem Core-Gerät weiterzuleiten.	In werden AWS IoT Greengrass V2verbundene Geräte als Greengrass-Clientgeräte bezeichnet. • Sie ordnen Client-Geräte - Core-Geräten zu, um sie zu verbinden und über MQTT zu kommunizieren. • Um Client-Geräte für die Verbindung zu autorisieren, definieren Sie Autorisierungsrichtlinien, die für Gruppen von Client-Geräten gelten können, sodass Sie keine Bereitstellung erstellen müssen, um ein Client-Gerät hinzuzufügen oder zu entfernen. • Um Nachrichten zwischen Client-Geräten AWS IoT Coreund Greengrass-Komponenten weiterzuleiten, können Sie eine optionale MQTT-Bridge-Komponente konfigurieren. Sowohl in als auch in können AWS IoT Greengrass V1 AWS IoT Greengrass V2Geräte FreeRTOS ausführen oder die AWS IoT Device SDK oder die Greengrass-Discovery-API verwenden, um Informationen

Konzept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
		über -Core-Geräte abzurufen, mit denen sie eine Verbindung herstellen können. Die Greengrass-Discovery-API ist abwärtskompatibel. Wenn Sie also über Client-Geräte verfügen, die eine Verbindung zu einem V1-Core-Gerät herstellen, können Sie sie mit einem V2-Core-Gerät verbinden, ohne ihren Code zu ändern. Weitere Informationen zu Client-Geräten finden Sie unter Interagieren mit lokalen IoT-Geräten.

Konzept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Lokale Ressourcen	In können Lambda-Funktionen AWS IoT Greengrass V1, die in Containern ausgeführ t werden, für den Zugriff auf Volumes und Geräte im Dateisystem des Core-Geräts konfiguriert werden. Diese Dateisystemressourcen werden als lokale Ressourcen bezeichnet.	In können Sie Komponent en ausführen AWS IoT Greengrass V2, bei denen es sich um Lambda-Funktionen, Docker-Container oder native Betriebssystemprozesse oder benutzerdefinierte Laufzeiten handelt. • Wenn Sie eine container isierte Lambda-Funktion als Komponente importier en, müssen Sie die lokalen Ressourcen angeben, die die Funktion verwendet. • Nicht containerisierte Lambda-Funktionen und Nicht-Lambda-Kompo nenten können direkt mit lokalen Ressourcen auf -Core-Geräten arbeiten, sodass Sie die lokalen Ressourcen, die die Komponente verwendet, nicht angeben müssen.

Konzept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Lokaler Schattenservice	In ist AWS IoT Greengrass V1der lokale Schattenservice standardmäßig aktiviert und unterstützt nur unbenannte klassische Schatten. Sie verwenden das AWS IoT Greengrass Core SDK in Ihren Lambda-Funktionen, um mit Schatten auf Ihren Geräten zu interagieren.	In aktivieren Sie den lokalen Schattenservice AWS IoT Greengrass V2, indem Sie die Shadow-Manager-Komponente bereitstellen. • Sie können AWS IoT Device SDK V2 in Lambda-Funktionen und benutzerdefinierten Komponenten verwenden, um mit Schatten auf Ihren Geräten zu interagieren. • Der lokale Schattenservice unterstützt benannte Schatten. • Mit dem lokalen Schattenservice können Sie Schatten löschen und gelöschte Schatten mit synchronisieren AWS IoT Core. Weitere Informationen finden Sie unter Interagieren mit Geräteschatten.

Konzept	AWS IoT Greengrass V1	AWS IoT Greengrass V2

Subscriptions (Abonnements)	In definieren Sie Abonnements für eine Greengrass-Gruppe AWS IoT Greengrass V1, um Kommunikationskanäle zwischen Lambda-Funktionen, Konnektoren, verbundenen Geräten, dem AWS IoT Core MQTT-Broker und dem lokalen Schattenservice anzugeben. Abonnements geben an, wo Lambda-Funktionen Ereignismeldungen empfangen, die als Funktionsnutzlasten verwendet werden sollen.	In geben Sie Kommunikationskanäle an AWS IoT Greengrass V2, ohne Abonnements zu verwenden.
		• Komponenten verwalten ihre eigenen Kommunikationskanäle, um mit lokalen Veröffentlichungs-/Abonnementnachrichten, AWS IoT Core MQTT-Nachrichten und dem lokalen Schattenservice zu interagieren.
		  • Um eine Komponente zu entwickeln, die auf Nachrichten von einer anderen Komponente oder dem AWS IoT Core MQTT-Broker reagiert, können Sie Schnittstellen für die Interprocess-Kommunikation (IPC) für lokale Veröffentlichungs-/Abonnementnachrichten und AWS IoT Core MQTT-Nachrichten verwenden.
		  • Um eine Komponente zu entwickeln, die mit dem lokalen Schattendienst interagiert, können Sie die IPC-Schnittstelle für den

Konzept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
		lokalen Schattendienst verwenden. • In der Komponent enkonfiguration definiere n Sie Autorisierungsrich tlinien, um die Themen und lokalen Schatten anzugeben, zu deren Verwendung die Komponente berechtigt ist. • Um Kommunikationskanä le zwischen Client-Geräten, dem lokalen Publish/S ubscribe-Broker und dem AWS IoT Core MQTT-Brok er zu konfigurieren, konfiguri eren und stellen Sie die MQTT-Bridge-Komponente bereit. Mit der MQTT-Brid ge-Komponente können Sie mit Client-Geräten in Komponenten interagieren und Nachrichten zwischen Client-Geräten und weiterlei ten AWS IoT Core.

Konzept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Zugreifen auf andere AWS-Services	In fügen Sie eine AWS Identity and Access Management (IAM)- AWS IoT Greengras s V1Rolle, die als Gruppenro lle bezeichnet wird, an eine Greengrass-Gruppe an. Die Gruppenrolle definiert die Berechtigungen, die Lambda-Funktionen und - AWS IoT Greengrass Funktionen auf dem Core-Gerät dieser Gruppe für den Zugriff auf verwenden AWS-Services.	In fügen Sie einen - AWS IoT Roll AWS IoT Greengrass V2enalias an ein Greengrass-Core-Gerät an. Der Rollenali as verweist auf eine IAM-Rolle, die als Token-Exc hange-Rolle bezeichnet wird. Die Token-Exchange-Rolle definiert die Berechtigungen, die Greengrass-Komponenten auf dem Core-Gerät für den Zugriff auf verwenden AWS-Services. Weitere Informati onen finden Sie unter Autorisie ren Sie Kerngeräte für die Interaktion mit Diensten AWS.

Überprüfen Sie, ob V1-Core-Geräte V2-Software ausführen können

DieAWS IoT Greengrass Core-Software v2.x hat andere Anforderungen als dieAWS IoT Greengrass Core-Software v1.x. Bevor Sie V1-Core-Geräte auf V2 aktualisieren, überprüfen Sie die Geräteanforderungen fürAWS IoT Greengrass V2. AWS IoT Greengrass V2unterstützt derzeit keine Migration für benutzerdefinierte Linux-basierte Systeme mithilfe des Yocto-Projekts.

Sie können AWS IoT Device Tester(IDT) verwenden,AWS IoT Greengrass V2 um zu überprüfen, ob Geräte die Anforderungen für die Ausführung derAWS IoT Greengrass Core-Software v2.x erfüllen. IDT ist ein herunterladbares Testframework, das auf Ihrem Host-Computer ausgeführt wird und eine Verbindung zu Geräten herstellt, um zu validieren. Folgen Sie den Anweisungen, um IDT zum Ausführen derAWS IoT Greengrass Qualifikationssuite zu verwenden. Bei der Konfiguration von IDT können Sie wählen, ob Geräte optionale Funktionen wie Docker, maschinelles Lernen (ML), Datenstrommanagement und Hardwaresicherheitsintegration unterstützen.

Wenn IDT V2-Testfehler oder -fehler für ein V1-Core-Gerät meldet, können Sie dieses Gerät nicht von V1 auf V2 aktualisieren.

Einrichten eines neuen V2-Core-Geräts zum Testen von V1-Anwendungen

Richten Sie ein neues - AWS IoT Greengrass V2 Core AWS-Gerät ein, um von bereitgestellte Komponenten und Funktionen für Ihre AWS IoT Greengrass V1 Anwendungen bereitzustellen und AWS Lambda zu testen. Sie können dieses V2-Core-Gerät auch verwenden, um zusätzliche benutzerdefinierte Greengrass-Komponenten zu entwickeln und zu testen, die native Prozesse auf Core-Geräten ausführen. Nachdem Sie Ihre Anwendungen auf einem V2-Core-Gerät getestet haben, können Sie Ihre vorhandenen V1-Core-Geräte auf V2 aktualisieren und die V2-Komponenten bereitstellen, die Ihre V1-Funktionalität bereitstellen.

Schritt 1: Installieren von AWS IoT Greengrass V2 auf einem neuen Gerät

Installieren Sie die AWS IoT Greengrass Core-Software v2.x auf einem neuen Gerät. Sie können dem Tutorial „Erste Schritte“ folgen, um ein Gerät einzurichten und zu erfahren, wie Sie Komponenten entwickeln und bereitstellen. In diesem Tutorial wird die automatische Bereitstellung verwendet, um schnell ein Gerät einzurichten. Wenn Sie die AWS IoT Greengrass Core-Software v2.x installieren, geben Sie das `--deploy-dev-tools` Argument an, um die Greengrass-CLI bereitzustellen, damit Sie Komponenten direkt auf dem Gerät entwickeln, testen und debuggen können. Weitere Informationen zu anderen Installationsoptionen, einschließlich der Installation der AWS IoT Greengrass Core-Software hinter einem Proxy oder der Verwendung eines Hardware-Sicherheitsmoduls (HSM), finden Sie unter Installieren Sie die AWS IoT Greengrass Core-Software..

(Optional) Aktivieren der Protokollierung in Amazon CloudWatch Logs

Damit ein V2-Core AWS-Gerät Protokolle in Amazon CloudWatch Logs hochladen kann, können Sie die von bereitgestellte Protokollmanagerkomponente bereitstellen. Sie können - CloudWatch Protokolle verwenden, um Komponentenprotokolle anzuzeigen, sodass Sie debuggen und Fehler beheben können, ohne Zugriff auf das Dateisystem des Core-Geräts zu haben. Weitere Informationen finden Sie unter Überwachen von AWS IoT Greengrass Protokollen.

Schritt 2: Erstellen und Bereitstellen von AWS IoT Greengrass V2 Komponenten zur Migration von AWS IoT Greengrass V1 Anwendungen

Sie können die meisten AWS IoT Greengrass V1 Anwendungen auf ausführen AWS IoT Greengrass V2. Sie können Lambda-Funktionen als Komponenten importieren, die auf ausgeführt werden AWS

IoT Greengrass V2, und Sie können AWSvon bereitgestellte Komponenten verwenden, die dieselbe Funktionalität wie AWS IoT Greengrass Konnektoren bieten.

Sie können auch benutzerdefinierte Komponenten entwickeln, um jede Funktion oder Laufzeit so zu erstellen, dass sie auf Greengrass-Core-Geräten ausgeführt wird. Informationen zum lokalen Entwickeln und Testen von Komponenten finden Sie unter Erstellen von AWS IoT Greengrass Komponenten.

Themen

- Importieren von V1-Lambda-Funktionen
- Verwenden von V1-Konnektoren
- Ausführen von Docker-Containern
- Machine Learning-Inferenz ausführen
- Verbinden von V1-Greengrass-Geräten
- Aktivieren des lokalen Schattendienstes
- Integrieren mit AWS IoT SiteWise

Importieren von V1-Lambda-Funktionen

Sie können Lambda-Funktionen als AWS IoT Greengrass V2 Komponenten importieren. Wählen Sie aus den folgenden Ansätzen aus:

- Importieren Sie V1-Lambda-Funktionen direkt als Greengrass-Komponenten.
- Aktualisieren Sie Ihre Lambda-Funktionen, um die Greengrass-Bibliotheken in der AWS IoT Device SDK v2 zu verwenden, und importieren Sie dann die Lambda-Funktionen als Greengrass-Komponenten.
- Erstellen Sie benutzerdefinierte Komponenten, die Nicht-Lambda-Code und AWS IoT Device SDK v2 verwenden, um dieselbe Funktionalität wie Ihre Lambda-Funktionen zu implementieren.

Wenn Ihre Lambda AWS-Funktion Funktionen wie Stream-Manager oder lokale Secrets verwendet, müssen Sie Abhängigkeiten von den von bereitgestellten Komponenten definieren, die diese Funktionen verpacken. Wenn Sie die Lambda-Funktionskomponente bereitstellen, enthält die Bereitstellung auch die Komponente für jedes Feature, das Sie als Abhängigkeit definieren. In der Bereitstellung können Sie Parameter konfigurieren, z. B. welche Secrets auf dem Core-Gerät bereitgestellt werden sollen. Nicht alle V1-Funktionen erfordern eine Komponentenabhängigkeit für

Ihre Lambda-Funktion auf V2. In der folgenden Liste wird beschrieben, wie Sie V1-Funktionen in Ihrer V2-Lambda-Funktionskomponente verwenden.

- Zugriff auf andere - AWS Services

  Wenn Ihre Lambda-Funktion AWS Anmeldeinformationen verwendet, um Anfragen an andere AWS -Services zu stellen, muss die Token-Austauschrolle des Core-Geräts dem Core-Gerät erlauben, die von der Lambda-Funktion verwendeten AWS Operationen auszuführen. Weitere Informationen finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

- Stream-Manager

  Wenn Ihre Lambda-Funktion den Stream-Manager verwendet, geben Sie `aws.greengrass.StreamManager` als Komponentenabhängigkeit an, wenn Sie die Funktion importieren. Wenn Sie die Stream-Manager-Komponente bereitstellen, geben Sie die Stream-Manager-Parameter an, die für die Ziel-Core-Geräte festgelegt werden sollen. Die Token-Austauschrolle des Core-Geräts muss dem Core-Gerät den Zugriff auf die AWS Cloud Ziele ermöglichen, die Sie mit dem Stream-Manager verwenden. Weitere Informationen finden Sie unter Stream-Manager.

- Lokale Secrets

  Wenn Ihre Lambda-Funktion lokale Secrets verwendet, geben Sie `aws.greengrass.SecretManager` als Komponentenabhängigkeit an, wenn Sie die Funktion importieren. Wenn Sie die Secret-Manager-Komponente bereitstellen, geben Sie die Secret-Ressourcen an, die auf den Ziel-Core-Geräten bereitgestellt werden sollen. Die Token-Austauschrolle des Core-Geräts muss es dem Core-Gerät ermöglichen, die geheimen Ressourcen für die Bereitstellung abzurufen. Weitere Informationen finden Sie unter Geheimer Manager.

  Wenn Sie Ihre Lambda-Funktionskomponente bereitstellen, konfigurieren Sie sie so, dass sie über eine IPC-Autorisierungsrichtlinie verfügt, die die Berechtigung zur Verwendung der GetSecretValue IPC-Operation in der AWS IoT Device SDK V2 gewährt.

- Lokale Schatten

  Wenn Ihre Lambda-Funktion mit lokalen Schatten interagiert, müssen Sie den Lambda-Funktionscode aktualisieren, um die AWS IoT Device SDK V2 zu verwenden. Sie müssen auch `aws.greengrass.ShadowManager` als Komponentenabhängigkeit angeben, wenn Sie die Funktion importieren. Weitere Informationen finden Sie unter Interagieren mit Geräteschatten.

Wenn Sie Ihre Lambda-Funktionskomponente bereitstellen, konfigurieren Sie sie so, dass sie über eine IPC-Autorisierungsrichtlinie verfügt, die die Berechtigung zur Verwendung der Schatten-IPK-Operationen in der AWS IoT Device SDK V2 gewährt.

- Abonnements
  - Wenn Ihre Lambda-Funktion Nachrichten aus einer Cloud-Quelle abonniert, geben Sie diese Abonnements als Ereignisquellen an, wenn Sie die Funktion importieren.
  - Wenn Ihre Lambda-Funktion Nachrichten von einer anderen Lambda-Funktion abonniert oder wenn Ihre Lambda-Funktion Nachrichten an AWS IoT Core oder andere Lambda-Funktionen veröffentlicht, konfigurieren und stellen Sie die Legacy-Abonnement-Routerkomponente bereit, wenn Sie Ihre Lambda-Funktion bereitstellen. Wenn Sie die Legacy-Abonnement-Routerkomponente bereitstellen, geben Sie die Abonnements an, die die Lambda-Funktion verwendet.

    > **Note**
    >
    > Die Legacy-Abonnement-Routerkomponente ist nur erforderlich, wenn Ihre Lambda-Funktion die `publish()` Funktion im AWS IoT Greengrass Core-SDK verwendet. Wenn Sie Ihren Lambda-Funktionscode aktualisieren, um die Interprocess Communication (IPC)-Schnittstelle in der AWS IoT Device SDK V2 zu verwenden, müssen Sie die Legacy-Abonnement-Routerkomponente nicht bereitstellen. Weitere Informationen finden Sie unter den folgenden prozessübergreifenden Kommunikationsservices:
    >
    > - Lokale Nachrichten veröffentlichen/abonnieren
    > - MQTT-Nachrichten veröffentlichen/abonnieren AWS IoT Core

  - Wenn Ihre Lambda-Funktion Nachrichten von lokal verbundenen Geräten abonniert, geben Sie diese Abonnements als Ereignisquellen an, wenn Sie die Funktion importieren. Sie müssen auch die MQTT-Bridge-Komponente konfigurieren und bereitstellen, um Nachrichten von den verbundenen Geräten an die lokalen Themen zum Veröffentlichen/Abonnieren weiterzuleiten, die Sie als Ereignisquellen angeben.
  - Wenn Ihre Lambda-Funktion Nachrichten auf lokal verbundenen Geräten veröffentlicht, müssen Sie den Lambda-Funktionscode aktualisieren, um die AWS IoT Device SDK V2 zum Veröffentlichen von lokalen Veröffentlichungs-/Abonnementnachrichten zu verwenden. Sie müssen auch die MQTT-Bridge-Komponente konfigurieren und bereitstellen, um Nachrichten vom lokalen Message Broker zum Veröffentlichen/Abonnementieren an die verbundenen Geräte weiterzuleiten.

- Lokale Volumes und Geräte

  Wenn Ihre containerisierte Lambda-Funktion auf lokale Volumes oder Geräte zugreift, geben Sie diese Volumes und Geräte an, wenn Sie die Lambda-Funktion importieren. Für diese Funktion ist keine Komponentenabhängigkeit erforderlich.

Weitere Informationen finden Sie unter Ausführen von -AWS LambdaFunktionen.

Verwenden von V1-Konnektoren

Sie können AWSvon bereitgestellte Komponenten bereitstellen, die die gleiche Funktionalität wie einige AWS IoT Greengrass Konnektoren bieten. Wenn Sie die Bereitstellung erstellen, können Sie die Parameter der Connectors konfigurieren.

Die folgenden AWS IoT Greengrass V2 Komponenten bieten Greengrass-V1-Konnektor-Funktionalität:

- CloudWatch Metrikkomponente
- AWS IoT Device Defender Komponente
- Firehose-Komponente
- Adapterkomponente des Bol-RTU-Protokolls
- Amazon SNS-Komponente

Ausführen von Docker-Containern

AWS IoT Greengrass V2 stellt keine Komponente bereit, um den V1-Docker-Anwendungsbereitstellungs-Connector direkt zu ersetzen. Sie können jedoch die Docker-Application-Manager-Komponente verwenden, um Docker-Images herunterzuladen und dann benutzerdefinierte Komponenten zu erstellen, die Docker-Container aus den heruntergeladenen Images ausführen. Weitere Informationen finden Sie unter Führen Sie einen Docker-Container aus und Docker-Anwendungsmanager.

Machine Learning-Inferenz ausführen

AWS IoT Greengrass V2 bietet eine Amazon SageMaker Edge Manager-Komponente, die den Amazon SageMaker Edge Manager-Agenten installiert und es Ihnen ermöglicht, SageMaker Neo-kompilierte Modelle als Modellkomponenten auf Greengrass-Core-Geräten zu verwenden. bietet AWS IoT Greengrass V2 auch Komponenten, die Deep Learning Runtime und TensorFlow Lite auf

Ihrem Gerät installieren. Sie können die entsprechenden DLR- und TensorFlow Lite-Modell- und Inferenzkomponenten verwenden, um eine Beispielbildklassifizierung und Objekterkennungsinferenz durchzuführen. Um andere Machine Learning-Frameworks wie MXNet und zu verwenden TensorFlow, können Sie Ihre eigenen benutzerdefinierten Komponenten entwickeln, die diese Frameworks verwenden.

Verbinden von V1-Greengrass-Geräten

Verbundene Geräte in AWS IoT Greengrass V1 werden in als Client-Geräte bezeichnet AWS IoT Greengrass V2. AWS IoT Greengrass V2 Die Unterstützung für Client-Geräte ist abwärtskompatibel mit AWS IoT Greengrass V1, sodass Sie V1-Client-Geräte mit V2-Core-Geräten verbinden können, ohne ihren Anwendungscode zu ändern. Damit Client-Geräte eine Verbindung zu einem V2-Core-Gerät herstellen können, stellen Sie Greengrass-Komponenten bereit, die Client-Geräteunterstützung ermöglichen, und ordnen Sie die Client-Geräte dem Core-Gerät zu. Um Nachrichten zwischen Client-Geräten, dem AWS IoT Core Cloud-Service und Greengrass-Komponenten (einschließlich Lambda-Funktionen) weiterzuleiten, stellen Sie die MQTT-Bridge-Komponente bereit und konfigurieren Sie sie. Sie können die IP-Detektorkomponente bereitstellen, um Konnektivitätsinformationen automatisch zu erkennen, oder Sie können Endpunkte manuell verwalten. Weitere Informationen finden Sie unter Interagieren mit lokalen IoT-Geräten.

Aktivieren des lokalen Schattendienstes

In wird AWS IoT Greengrass V2der lokale Schattenservice durch die von AWSbereitgestellte Schattenmanagerkomponente implementiert. bietet AWS IoT Greengrass V2 auch Unterstützung für benannte Schatten. Damit Ihre Komponenten mit lokalen Schatten interagieren und Schattenstatus mit synchronisieren können AWS IoT Core, konfigurieren und stellen Sie die Shadow Manager-Komponente bereit und verwenden Sie die Shadow IPC-Operationen in Ihrem Komponentencode. Weitere Informationen finden Sie unter Interagieren mit Geräteschatten.

Integrieren mit AWS IoT SiteWise

Wenn Sie Ihr V1-Core-Gerät als - AWS IoT SiteWise Gateway verwenden, folgen Sie den Anweisungen, um Ihr neues V2-Core-Gerät als - AWS IoT SiteWise Gateway einzurichten. AWS IoT SiteWise bietet ein Installationsskript, das die AWS IoT SiteWise Komponenten für Sie bereitstellt.

Schritt 3: Testen Ihrer AWS IoT Greengrass V2 Anwendungen

Nachdem Sie V2-Komponenten erstellt und auf Ihrem neuen V2-Core-Gerät bereitgestellt haben, stellen Sie sicher, dass Ihre Anwendungen Ihren Erwartungen entsprechen. Sie können die

Geräteprotokolle überprüfen, um die Standardausgabemeldungen (stdout) und den Standardfehler (stderr) Ihrer Komponenten anzuzeigen. Weitere Informationen finden Sie unter Überwachen von AWS IoT Greengrass Protokollen.

Wenn Sie die Greengrass-CLI auf dem Core-Gerät bereitgestellt haben, können Sie sie zum Debuggen von Komponenten und deren Konfigurationen verwenden. Weitere Informationen finden Sie unter Greengrass-CLI-Befehle.

Nachdem Sie sich vergewissert haben, dass Ihre Anwendungen auf einem V2-Core-Gerät funktionieren, können Sie die Greengrass-Komponenten Ihrer Anwendung auf anderen Core-Geräten bereitstellen. Wenn Sie benutzerdefinierte Komponenten entwickelt haben, die native Prozesse oder Docker-Container ausführen, müssen Sie diese Komponenten zuerst im Service veröffentlichen, um sie auf anderen -Core-Geräten bereitzustellen. AWS IoT Greengrass

Upgrade von Greengrass-V1-Core-Geräten auf Greengrass V2

Nachdem Sie sich vergewissert haben, dass Ihre Anwendungen und Komponenten auf einem -AWS IoT Greengrass V2Core-Gerät funktionieren, können Sie die -AWS IoT GreengrassCore-Software v2.x auf Ihren Geräten installieren, auf denen derzeit v1.x ausgeführt wird, z. B. auf Produktionsgeräten. Stellen Sie dann Greengrass V2-Komponenten bereit, um Ihre Greengrass-Anwendungen auf den Geräten auszuführen.

Um eine Flotte von Geräten von V1 auf V2 zu aktualisieren, führen Sie diese Schritte für jedes Gerät aus, das aktualisiert werden soll. Sie können Objektgruppen verwenden, um V2-Komponenten auf einer Flotte von -Core-Geräten bereitzustellen.

> **Tip**
>
> Wir empfehlen Ihnen, ein Skript zu erstellen, um den Upgrade-Prozess für eine Flotte von Geräten zu automatisieren. Wenn Sie verwenden, AWS Systems Manager um Ihre Flotte zu verwalten, können Sie Systems Manager verwenden, um dieses Skript auf jedem Gerät auszuführen, um Ihre Flotte von V1 auf V2 zu aktualisieren.
> Sie können sich bei Fragen zur optimalen Automatisierung des Upgrade-Prozesses an Ihren AWS Enterprise Support-Mitarbeiter wenden.

Schritt 1: Installieren der AWS IoT Greengrass Core-Software v2.x

Wählen Sie aus den folgenden Optionen, um die AWS IoT Greengrass Core-Software v2.x auf einem V1-Core-Gerät zu installieren:

- Upgrade in weniger Schritten

  Um ein Upgrade in weniger Schritten durchzuführen, können Sie die v1.x-Software deinstallieren, bevor Sie die v2.x-Software installieren.

- Upgrade mit minimalen Ausfallzeiten

  Um ein Upgrade mit minimalen Ausfallzeiten durchzuführen, können Sie beide Versionen der AWS IoT Greengrass -Core-Software gleichzeitig installieren. Nachdem Sie die AWS IoT Greengrass Core-Software v2.x installiert und überprüft haben, ob Ihre V2-Anwendungen ordnungsgemäß funktionieren, deinstallieren Sie die AWS IoT Greengrass Core-Software v1.x. Bevor Sie diese Option wählen, sollten Sie den zusätzlichen RAM berücksichtigen, der erforderlich ist, um beide Versionen der AWS IoT Greengrass Core-Software gleichzeitig auszuführen.

Deinstallieren von AWS IoT Greengrass Core v1.x vor der Installation von v2.x

Wenn Sie sequentiell aktualisieren möchten, deinstallieren Sie die AWS IoT Greengrass Core-Software v1.x, bevor Sie v2.x auf Ihrem Gerät installieren.

So deinstallieren Sie die AWS IoT Greengrass Core-Software v1.x

1. Wenn die AWS IoT Greengrass Core-Software v1.x als Service ausgeführt wird, müssen Sie den Service anhalten, deaktivieren und entfernen.

   a. Beenden Sie den laufenden AWS IoT Greengrass Core-Software-v1.x-Service.

   ```
   sudo systemctl stop greengrass
   ```

   b. Warten Sie, bis der Service beendet ist. Sie können den `list` Befehl verwenden, um den Status des Services zu überprüfen.

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

   c. Deaktivieren Sie den Service.

```
sudo systemctl disable greengrass
```

d. Entfernen Sie den Service.

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

2. Wenn die AWS IoT Greengrass Core-Software v1.x nicht als Service ausgeführt wird, verwenden Sie den folgenden Befehl, um den Daemon zu stoppen. Ersetzen Sie *greengrass-root* durch den Namen Ihres Greengrass-Stammordners. Der Standardspeicherort ist `/greengrass`.

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

3. (Optional) Sichern Sie Ihren Greengrass-Stammordner und gegebenenfalls Ihren benutzerdefinierten Schreibordner in einem anderen Ordner auf Ihrem Gerät.

   a. Verwenden Sie den folgenden Befehl, um den aktuellen Greengrass-Stammordner in einen anderen Ordner zu kopieren und dann den Stammordner zu entfernen.

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

   b. Verwenden Sie den folgenden Befehl, um den Schreibordner in einen anderen Ordner zu verschieben, und entfernen Sie dann den Schreibordner.

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

Anschließend können Sie die Installationsanweisungen für AWS IoT Greengrass V2 verwenden, um die Software auf Ihrem Gerät zu installieren.

> **Tip**
>
> Um die Identität eines Core-Geräts bei der Migration von V1 zu V2 wiederzuverwenden, folgen Sie den Anweisungen, um die AWS IoT Greengrass Core-Software mit manueller Bereitstellung zu installieren. Entfernen Sie zuerst die V1-Core-Software vom Gerät, verwenden Sie dann das AWS IoT Objekt und das Zertifikat des V1-Core-Geräts wieder und

aktualisieren Sie die AWS IoT Richtlinien des Zertifikats, um Berechtigungen zu erteilen, die die v2.x-Software benötigt.

Installieren von AWS IoT Greengrass Core-Software v2.x auf einem Gerät, auf dem bereits v1.x ausgeführt wird

Wenn Sie die AWS IoT Greengrass Core v2.x-Software auf einem Gerät installieren, auf dem die AWS IoT Greengrass Core-Software v1.x bereits ausgeführt wird, beachten Sie Folgendes:

- Der AWS IoT Objektname für Ihr V2-Core-Gerät muss eindeutig sein. Verwenden Sie nicht denselben Objektnamen wie Ihr V1-Core-Gerät.
- Die Ports, die Sie für die AWS IoT Greengrass Core-Software v2.x verwenden, müssen sich von den Ports unterscheiden, die Sie für v1.x verwenden.
  - Konfigurieren Sie den V1-Stream-Manager für die Verwendung eines anderen Ports als 8088. Weitere Informationen finden Sie unter Stream-Manager konfigurieren.
  - Konfigurieren Sie den V1-MQTT-Broker so, dass er einen anderen Port als 8883 verwendet. Weitere Informationen finden Sie unter Konfigurieren des MQTT-Ports für lokales Messaging.
- AWS IoT Greengrass V2 bietet nicht die Möglichkeit, den Greengrass-Systemservice umzubenennen. Wenn Sie Greengrass als Systemservice ausführen, müssen Sie einen der folgenden Schritte ausführen, um widersprüchliche Systemservicenamen zu vermeiden:
  - Benennen Sie den Greengrass-Service für v1.x um, bevor Sie v2.x installieren.
  - Installieren Sie die AWS IoT Greengrass Core-Software v2.x ohne Systemservice und konfigurieren Sie die Software dann manuell als Systemservice mit einem anderen Namen als `greengrass`.

  So benennen Sie den Greengrass-Service für v1.x um

  1. Stoppen Sie den AWS IoT Greengrass Core-Software-v1.x-Service.

     ```
     sudo systemctl stop greengrass
     ```

  2. Warten Sie, bis der Service beendet ist. Der Service kann bis zu ein paar Minuten dauern. Sie können den `list-units` Befehl verwenden, um zu überprüfen, ob der Service gestoppt wurde.

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

3. Deaktivieren Sie den Service.

```
sudo systemctl disable greengrass
```

4. Benennen Sie den Service um.

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

5. Laden Sie den Service neu und starten Sie ihn.

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

Anschließend können Sie die Installationsanweisungen für AWS IoT Greengrass V2 verwenden, um die Software auf Ihrem Gerät zu installieren.

> **Tip**
>
> Um die Identität eines Core-Geräts bei der Migration von V1 zu V2 wiederzuverwenden, folgen Sie den Anweisungen, um die AWS IoT Greengrass Core-Software mit manueller Bereitstellung zu installieren. Entfernen Sie zuerst die V1-Core-Software vom Gerät, verwenden Sie dann das AWS IoT Objekt und das Zertifikat des V1-Core-Geräts wieder und aktualisieren Sie die AWS IoT Richtlinien des Zertifikats, um Berechtigungen zu erteilen, die die v2.x-Software benötigt.

Schritt 2: Bereitstellen von AWS IoT Greengrass V2 Komponenten auf den Core-Geräten

Nachdem Sie die AWS IoT Greengrass Core-Software v2.x auf Ihrem Gerät installiert haben, erstellen Sie eine Bereitstellung, die die folgenden Ressourcen enthält. Um Komponenten auf einer Flotte ähnlicher Geräte bereitzustellen, erstellen Sie eine Bereitstellung für eine Objektgruppe, die diese Geräte enthält.

- Lambda-Funktionskomponenten, die Sie aus Ihren V1-Lambda-Funktionen erstellt haben. Weitere Informationen finden Sie unter Ausführen von -AWS LambdaFunktionen.
- Wenn Sie V1-Abonnements verwenden, die Legacy-Abonnement-Routerkomponente .
- Wenn Sie den Stream-Manager verwenden, die Stream-Manager-Komponente . Weitere Informationen finden Sie unter Verwalten von Datenströmen auf Greengrass-Core-Geräten.
- Wenn Sie lokale Secrets verwenden, die Secret-Manager-Komponente .
- Wenn Sie V1AWS-Konnektoren verwenden, die von bereitgestellten Konnektorkomponenten .
- Wenn Sie Docker-Container verwenden, die Docker-Anwendungsmanager-Komponente . Weitere Informationen finden Sie unter Führen Sie einen Docker-Container aus.
- Wenn Sie Machine Learning-Inferenz verwenden, Komponenten für die Unterstützung von Machine Learning. Weitere Informationen finden Sie unter Durchführen von Machine Learning-Inferenzen.
- Wenn Sie verbundene Geräte verwenden, unterstützen die Komponenten für das Client-Gerät . Sie müssen auch die Client-Geräteunterstützung aktivieren und die Client-Geräte Ihrem Core-Gerät zuordnen. Weitere Informationen finden Sie unter Interagieren mit lokalen IoT-Geräten.
- Wenn Sie Geräteschatten verwenden, die Shadow Manager-Komponente . Weitere Informationen finden Sie unter Interagieren mit Geräteschatten.
- Wenn Sie Protokolle von Greengrass-Core-Geräten auf Amazon CloudWatch Logs hochladen, die Log Manager-Komponente . Weitere Informationen finden Sie unter Überwachen von AWS IoT Greengrass Protokollen.
- Wenn Sie in integrierenAWS IoT SiteWise, folgen Sie den Anweisungen, um das V2-Core-Gerät als -AWS IoT SiteWiseGateway einzurichten. AWS IoT SiteWise bietet ein Installationsskript, das die AWS IoT SiteWise Komponenten für Sie bereitstellt.
- Benutzerdefinierte Komponenten, die Sie zur Implementierung benutzerdefinierter Funktionen entwickelt haben.

Informationen zum Erstellen und Überarbeiten von Bereitstellungen finden Sie unter Bereitstellen von AWS IoT Greengrass Komponenten auf Geräten.

Tutorial: Erste Schritte mit AWS IoT Greengrass V2

Sie können dieses Tutorial für die ersten Schritte abschließen, um die grundlegenden Funktionen von zu erlernenAWS IoT Greengrass V2. In diesem Tutorial führen Sie folgende Aufgaben aus:

1. Installieren und konfigurieren Sie die AWS IoT Greengrass Core-Software auf einem Linux-Gerät, z. B. einem Raspberry Pi oder einem Windows-Gerät. Dieses Gerät ist ein Greengrass-Core-Gerät.
2. Entwickeln Sie eine Hello-World-Komponente auf Ihrem Greengrass-Kerngerät. Komponenten sind Softwaremodule, die auf Greengrass-Core-Geräten ausgeführt werden.
3. Laden Sie diese Komponente in AWS IoT Greengrass V2 in hochAWS Cloud.
4. Stellen Sie diese Komponente vom AWS Cloud auf Ihrem Greengrass-Kerngerät bereit.

> **Note**
>
> In diesem Tutorial wird beschrieben, wie Sie eine Entwicklungsumgebung einrichten und die Funktionen von erkundenAWS IoT Greengrass. Weitere Informationen zum Einrichten und Konfigurieren von Produktionsgeräten finden Sie hier:
>
> - Einrichtung von AWS IoT Greengrass Kerngeräten
> - Installieren Sie die AWS IoT Greengrass Core-Software.

Sie können damit rechnen, 20 bis 30 Minuten für dieses Tutorial zu verbringen.

Themen

- Nächste Schritte

Voraussetzungen

Für dieses Tutorial benötigen Sie Folgendes:

- Ein(e) AWS-Konto. Falls Sie noch keines haben, beachten Sie die Informationen unter Schritt 1: Richten Sie ein AWS Konto ein.
- Die Verwendung eines AWS-Region, der unterstütztAWS IoT Greengrass V2. Eine vollständige Liste der unterstützten Regionen finden Sie unter AWS IoT Greengrass V2-Endpunkte und -Kontingente in Allgemeine AWS-Referenz.
- Ein AWS Identity and Access Management (IAM)-Benutzer mit Administratorberechtigungen.
- Ein Gerät, das als Greengrass-Core-Gerät eingerichtet werden soll, z. B. ein Raspberry Pi mit Raspberry Pi OS (früher Raspbian genannt) oder ein Windows 10-Gerät. Sie müssen über Administratorberechtigungen auf diesem Gerät oder die Möglichkeit verfügen, Administratorrechte zu erwerben, z. B. über `sudo`. Dieses Gerät muss über eine Internetverbindung verfügen.

  Sie können auch ein anderes Gerät verwenden, das die Anforderungen für die Installation und Ausführung der AWS IoT Greengrass Core-Software erfüllt. Weitere Informationen finden Sie unter Unterstützte Plattformen und Anforderungen.

  Wenn Ihr Entwicklungscomputer diese Anforderungen erfüllt, können Sie ihn in diesem Tutorial als Greengrass-Core-Gerät einrichten.
- Python 3.5 oder höher wurde für alle Benutzer auf dem Gerät installiert und der `PATH` Umgebungsvariablen hinzugefügt. Unter Windows muss auch der Python Launcher für Windows für alle Benutzer installiert sein.

  > **Important**
  >
  > Unter Windows installiert Python standardmäßig nicht für alle Benutzer. Wenn Sie Python installieren, müssen Sie die Installation anpassen, um sie für die AWS IoT Greengrass Core-Software zum Ausführen von Python-Skripts zu konfigurieren. Wenn Sie beispielsweise das grafische Python-Installationsprogramm verwenden, gehen Sie wie folgt vor:
  >
  > 1. Wählen Sie Launcher installieren für alle Benutzer aus (empfohlen).
  > 2. Wählen Sie Customize installation.

3. Wählen Sie Next.
4. Wählen Sie Install for all users.
5. Wählen Sie Add Python to environment variables.
6. Wählen Sie Installieren aus.

Weitere Informationen finden Sie unter Verwenden von Python unter Windows in der Python-3-Dokumentation.

- AWS Command Line Interface (AWS CLI) installiert und konfiguriert mit Anmeldeinformationen auf Ihrem Entwicklungscomputer und auf Ihrem Gerät. Stellen Sie sicher, dass Sie dasselbe verwendenAWS-Region, um die AWS CLI auf Ihrem Entwicklungscomputer und auf Ihrem Gerät zu konfigurieren. Um AWS IoT Greengrass V2 mit der verwenden zu könnenAWS CLI, benötigen Sie eine der folgenden Versionen oder höher:
  - Mindestversion AWS CLI V1: v1.18.197
  - VAWS CLIV2-Mindestversion: v2.1.11

  Tip

  Sie können den folgenden Befehl ausführen, um die Version des zu überprüfenAWS CLI, die Sie haben.

  ```
  aws --version
  ```

  Weitere Informationen finden Sie unter Installieren, Aktualisieren und Deinstallieren der AWS CLI und Konfigurieren der AWS CLI im AWS Command Line Interface -Benutzerhandbuch.

  Note

  Wenn Sie ein 32-Bit-ARM-Gerät verwenden, z. B. einen Raspberry Pi mit einem 32-Bit-Betriebssystem, installieren Sie AWS CLI V1. AWS CLI V2 ist für 32-Bit-ARM-Geräte nicht verfügbar. Weitere Informationen finden Sie unter Installieren, Aktualisieren und Deinstallieren der AWS CLI Version 1.

Schritt 1: Richten Sie ein AWS Konto ein

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.

Schritt 2: Einrichten Ihrer Umgebung

Führen Sie die Schritte in diesem Abschnitt aus, um ein Linux- oder Windows-Gerät einzurichten, das als AWS IoT Greengrass Core-Gerät verwendet werden soll.

Einrichten eines Linux-Geräts (Raspberry Pi)

Bei diesen Schritten wird davon ausgegangen, dass Sie einen Raspberry Pi mit Raspberry Pi OS verwenden. Wenn Sie ein anderes Gerät oder Betriebssystem verwenden, lesen Sie die entsprechende Dokumentation für Ihr Gerät.

So richten Sie einen Raspberry Pi für ein AWS IoT Greengrass V2

1. Aktivieren Sie SSH auf Ihrem Raspberry Pi, um eine Remote-Verbindung herzustellen. Weitere Informationen finden Sie unter SSH (Secure Shell) in der Raspberry Pi-Dokumentation.
2. Suchen Sie die IP-Adresse Ihres Raspberry Pi, um eine Verbindung mit ihm über SSH herzustellen. Dazu können Sie den folgenden Befehl auf Ihrem Raspberry Pi ausführen.

   ```
   hostname -I
   ```

3. Stellen Sie mit SSH eine Verbindung zu Ihrem Raspberry Pi her.

   Führen Sie auf Ihrem Entwicklungscomputer den folgenden Befehl aus. Ersetzen Sie *username* durch den Namen des Benutzers, der sich anmelden soll, und ersetzen Sie *pi-ip-address* durch die IP-Adresse, die Sie im vorherigen Schritt gefunden haben.

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

   > **Important**
   >
   > Wenn Ihr Entwicklungscomputer eine frühere Version von Windows verwendet, verfügen Sie möglicherweise nicht über den `ssh` Befehl oder können möglicherweise keine `ssh` Verbindung zu Ihrem Raspberry Pi herstellen. Um eine Verbindung zu Ihrem Raspberry Pi herzustellen, können Sie PuTTY installieren und konfigurieren, einen kostenlosen Open-Source-SSH-Client. Informationen zum Herstellen einer Verbindung mit Ihrem Raspberry Pi finden Sie in der PuTTY-Dokumentation.

4. Installieren Sie die Java-Laufzeit, die die -AWS IoT GreengrassCore-Software zum Ausführen benötigt. Verwenden Sie auf Ihrem Raspberry Pi die folgenden Befehle, um Java 11 zu installieren.

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

   Wenn die Installation abgeschlossen ist, führen Sie den folgenden Befehl aus, um zu überprüfen, ob Java auf Ihrem Raspberry Pi ausgeführt wird.

   ```
   java -version
   ```

   Der Befehl gibt die Version von Java aus, die auf dem Gerät ausgeführt wird. Die Ausgabe könnte dem folgenden Beispiel ähneln.

   ```
   openjdk version "11.0.9.1" 2020-11-04
   OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
   OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
   ```

Tipp: Kernel-Parameter auf einem Raspberry Pi festlegen

Wenn Ihr Gerät ein Raspberry Pi ist, können Sie die folgenden Schritte ausführen, um seine Linux-Kernelparameter anzuzeigen und zu aktualisieren:

1. Öffnen Sie die `/boot/cmdline.txt` Datei. Diese Datei gibt Linux-Kernelparameter an, die angewendet werden sollen, wenn der Raspberry Pi gestartet wird.

   Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Öffnen der Datei zu verwenden.

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

2. Überprüfen Sie, ob die `/boot/cmdline.txt` Datei die folgenden Kernelparameter enthält. Der `systemd.unified_cgroup_hierarchy=0` Parameter gibt an, dass cgroups v1 anstelle von cgroups v2 verwendet werden soll.

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

Wenn die `/boot/cmdline.txt` Datei diese Parameter nicht oder mit unterschiedlichen Werten enthält, aktualisieren Sie die Datei so, dass sie diese Parameter und Werte enthält.

3. Wenn Sie die `/boot/cmdline.txt` Datei aktualisiert haben, starten Sie den Raspberry Pi neu, um die Änderungen anzuwenden.

```
sudo reboot
```

Einrichten eines Linux-Geräts (andere)

So richten Sie ein Linux-Gerät für ein AWS IoT Greengrass V2

1. Installieren Sie die Java-Laufzeit, die die -AWS IoT GreengrassCore-Software zum Ausführen benötigt. Wir empfehlen Ihnen, Amazon Corretto- oder OpenJDK-Langzeit-Supportversionen zu verwenden. Version 8 oder höher ist erforderlich. Die folgenden Befehle zeigen Ihnen, wie Sie OpenJDK auf Ihrem Gerät installieren.

   - Für Debian- oder Ubuntu-basierte Distributionen:

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

   - Für Red Hat-basierte Distributionen:

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

   - Für Amazon Linux 2:

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

   - Für Amazon Linux 2023:

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

   Wenn die Installation abgeschlossen ist, führen Sie den folgenden Befehl aus, um zu überprüfen, ob Java auf Ihrem Linux-Gerät ausgeführt wird.

   ```
   java -version
   ```

Der Befehl gibt die Version von Java aus, die auf dem Gerät ausgeführt wird. Bei einer Debian-basierten Verteilung könnte die Ausgabe beispielsweise dem folgenden Beispiel ähneln.

```
openjdk version "11.0.9.1" 2020-11-04
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
```

2. (Optional) Erstellen Sie den Standardsystembenutzer und die Standardgruppe, die Komponenten auf dem Gerät ausführt. Sie können sich auch dafür entscheiden, dass das AWS IoT Greengrass-Core-Software-Installationsprogramm diesen Benutzer und diese Gruppe während der Installation mit dem `---component-default-user`Installationsprogramm-Argument erstellt. Weitere Informationen finden Sie unter Installer-Argumente.

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

3. Stellen Sie sicher, dass der Benutzer, der die AWS IoT Greengrass Core-Software ausführt (normalerweise `root`), über die Berechtigung verfügt, `sudo` mit jedem Benutzer und jeder Gruppe auszuführen.

   a. Führen Sie den folgenden Befehl aus, um die `/etc/sudoers` Datei zu öffnen.

   ```
   sudo visudo
   ```

   b. Stellen Sie sicher, dass die Berechtigung für den Benutzer wie im folgenden Beispiel aussieht.

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

4. (Optional) Um containerisierte Lambda-Funktionen auszuführen, müssen Sie cgroups v1 aktivieren und die Cgroups für Speicher und Geräte aktivieren und mounten. Wenn Sie keine containerisierten Lambda-Funktionen ausführen möchten, können Sie diesen Schritt überspringen.

   Um diese Cgroups-Optionen zu aktivieren, starten Sie das Gerät mit den folgenden Linux-Kernelparametern.

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

Informationen zum Anzeigen und Festlegen von Kernelparametern für Ihr Gerät finden Sie in der Dokumentation für Ihr Betriebssystem und Ihren Bootloader. Folgen Sie den Anweisungen, um die Kernel-Parameter dauerhaft festzulegen.

5. Installieren Sie alle anderen erforderlichen Abhängigkeiten auf Ihrem Gerät, wie in der Liste der Anforderungen unter angegebenAnforderungen an Speichergeräte.

Einrichten eines Windows-Geräts

So richten Sie ein Windows-Gerät für ein AWS IoT Greengrass V2

1. Installieren Sie die Java-Laufzeit, die die -AWS IoT GreengrassCore-Software zum Ausführen benötigt. Wir empfehlen Ihnen, Amazon Corretto- oder OpenJDK-Langzeit-Supportversionen zu verwenden. Version 8 oder höher ist erforderlich.
2. Überprüfen Sie, ob Java für die PATH-Systemvariable verfügbar ist, und fügen Sie es hinzu, falls nicht. Das LocalSystem Konto führt die AWS IoT Greengrass Core-Software aus, daher müssen Sie Java zur PATH-Systemvariablen anstelle der PATH-Benutzervariablen für Ihren Benutzer hinzufügen. Gehen Sie wie folgt vor:

   a. Drücken Sie die Windows-Taste, um das Startmenü zu öffnen.

   b. Geben Sie ein**`environment variables`**, um im Startmenü nach den Systemoptionen zu suchen.

   c. Wählen Sie im Startmenü die Option Systemumgebungsvariablen bearbeiten aus, um das Fenster Systemeigenschaften zu öffnen.

   d. Wählen Sie Umgebungsvariablen..., um das Fenster Umgebungsvariablen zu öffnen.

   e. Wählen Sie unter Systemvariablen die Option Pfad und dann Bearbeiten aus. Im Fenster Umgebungsvariable bearbeiten können Sie jeden Pfad in einer separaten Zeile anzeigen.

   f. Überprüfen Sie, ob der Pfad zum `bin` Ordner der Java-Installation vorhanden ist. Der Pfad könnte dem folgenden Beispiel ähneln.

   ```
   C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
   ```

   g. Wenn der `bin` Ordner der Java-Installation im Pfad fehlt, wählen Sie Neu, um ihn hinzuzufügen, und wählen Sie dann OK aus.
3. Öffnen Sie die Windows-Eingabeaufforderung (`cmd.exe`) als Administrator.

4. Erstellen Sie den Standardbenutzer im LocalSystem Konto auf dem Windows-Gerät. Ersetzen Sie *das Passwort* durch ein sicheres Passwort.

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

> **Tip**
>
> Abhängig von Ihrer Windows-Konfiguration kann das Passwort des Benutzers so eingestellt sein, dass es an einem Datum in der Zukunft abläuft. Um sicherzustellen, dass Ihre Greengrass-Anwendungen weiterhin funktionieren, verfolgen Sie, wann das Passwort abläuft, und aktualisieren Sie es, bevor es abläuft. Sie können auch festlegen, dass das Passwort des Benutzers nie abläuft.
>
> - Führen Sie den folgenden Befehl aus, um zu überprüfen, wann ein Benutzer und sein Passwort ablaufen.
>
> ```
> net user ggc_user | findstr /C:expires
> ```
>
> - Führen Sie den folgenden Befehl aus, um das Passwort eines Benutzers so festzulegen, dass es nie abläuft.
>
> ```
> wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
> ```
>
> - Wenn Sie Windows 10 oder höher verwenden, wenn der `wmic` Befehl veraltet ist, führen Sie den folgenden PowerShell Befehl aus.
>
> ```
> Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =
>  'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
> ```

5. Laden Sie das PsExec Dienstprogramm von Microsoft herunter und installieren Sie es auf dem Gerät.
6. Verwenden Sie das PsExec Dienstprogramm , um den Benutzernamen und das Passwort für den Standardbenutzer in der Credential Manager-Instance für das LocalSystem Konto zu speichern. Ersetzen Sie *password* durch das Passwort des Benutzers, das Sie zuvor festgelegt haben.

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

Wenn sich die PsExec License Agreement öffnet, wählen Sie ausAccept, ob Sie der Lizenz zustimmen möchten, und führen Sie den Befehl aus.

> **Note**
>
> Auf Windows-Geräten führt das LocalSystem Konto den Greengrass-Kern aus, und Sie müssen das PsExec Dienstprogramm verwenden, um die Standardbenutzerinformationen im LocalSystem Konto zu speichern. Die Verwendung der Credential Manager-Anwendung speichert diese Informationen im Windows-Konto des aktuell angemeldeten Benutzers anstelle des - LocalSystem Kontos.

Schritt 3: Installieren derAWS IoT Greengrass Core-Software: Installieren der Kern-Software: Installieren

Folgen Sie den Schritten in diesem Abschnitt, um Ihren Raspberry Pi alsAWS IoT Greengrass Kerngerät einzurichten, das Sie für lokale Entwicklung verwenden können. In diesem Abschnitt laden Sie ein Installationsprogramm herunter und führen es aus, das dieAWS IoT Greengrass Core-Software für Ihr Gerät wie folgt konfiguriert:

- Installiert die Greengrass-Nucleus-Komponente. Der Nucleus ist eine obligatorische Komponente und die Mindestanforderung, um dieAWS IoT Greengrass Core-Software auf einem Gerät auszuführen. Weitere Informationen finden Sie unter Verwenden Greengrass Nucleus-Komponente.
- Registriert Ihr Gerät alsAWS IoT Ding und lädt ein digitales Zertifikat herunter, mit dem Ihr Gerät eine Verbindung herstellen kannAWS. Weitere Informationen finden Sie unter Geräteauthentifizierung und -autorisierung für AWS IoT Greengrass.
- Fügt dasAWS IoT Ding des Geräts einer Dinggruppe hinzu, bei der es sich um eine Gruppe oder Flotte vonAWS IoT Dingen handelt. Mit Dinggruppen können Sie Flotten von Greengrass-Core-Geräten verwalten. Wenn Sie Softwarekomponenten auf Ihren Geräten bereitstellen, können Sie wählen, ob Sie sie auf einzelnen Geräten oder auf Gerätegruppen bereitstellen möchten. Weitere Informationen finden SieAWS IoT im AWS IoT CoreEntwicklerhandbuch unter Geräte verwalten mit.
- Erzeugt die IAM-Rolle, die es Ihrem Greengrass-Core-Gerät ermöglicht, mitAWS Diensten zu interagieren. Standardmäßig ermöglicht diese Rolle Ihrem Gerät, mit Amazon Logs zu

interagierenAWS IoT und Logs an AmazonCloudWatch Logs zu senden. Weitere Informationen finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

- Installiert dieAWS IoT Greengrass Befehlszeilenschnittstelle (`greengrass-cli`), mit der Sie benutzerdefinierte Komponenten testen können, die Sie auf dem Kerngerät entwickeln. Weitere Informationen finden Sie unter Greengrass-Befehlszeilenschnittstelle.

Installieren der AWS IoT Greengrass -Core-Software (Konsole)

1. Melden Sie sich an der AWS IoT Greengrass-Konsole an.
2. Wählen Sie unter Erste Schritte mit Greengrass die Option Ein Core-Gerät einrichten aus.
3. Geben Sie unter Schritt 1: Registrieren eines Greengrass-Core-Geräts für Core-Gerätename den Namen des AWS IoT Objekts für Ihr Greengrass-Core-Gerät ein. Wenn das Objekt nicht vorhanden ist, erstellt das Installationsprogramm es.
4. Wählen Sie unter Schritt 2: Zu einer Objektgruppe hinzufügen, um eine kontinuierliche Bereitstellung anzuwenden für Objektgruppe die AWS IoT Objektgruppe aus, der Sie Ihr Core-Gerät hinzufügen möchten.

   - Wenn Sie Neuen Gruppennamen eingeben auswählen, geben Sie unter Objektgruppenname den Namen der neuen Gruppe ein, die erstellt werden soll. Das Installationsprogramm erstellt die neue Gruppe für Sie.
   - Wenn Sie Vorhandene Gruppe auswählen auswählen auswählen, wählen Sie unter Objektgruppenname die vorhandene Gruppe aus, die Sie verwenden möchten.
   - Wenn Sie Keine Gruppe auswählen, fügt das Installationsprogramm das Core-Gerät nicht zu einer Objektgruppe hinzu.
5. Führen Sie unter Schritt 3: Installieren der Greengrass Core-Software die folgenden Schritte aus.

   a. Wählen Sie das Betriebssystem Ihres Core-Geräts aus: Linux oder Windows .

   b. Geben Sie Ihre AWS Anmeldeinformationen für das Gerät an, damit das Installationsprogramm die AWS IoT und IAM-Ressourcen für Ihr Core-Gerät bereitstellen kann. Um die Sicherheit zu erhöhen, empfehlen wir Ihnen, temporäre Anmeldeinformationen für eine IAM-Rolle zu erhalten, die nur die Mindestberechtigungen zulässt, die für die Bereitstellung erforderlich sind. Weitere Informationen finden Sie unter Minimale IAM-Richtlinie für das Installationsprogramm zur Bereitstellung von Ressourcen.

> **Note**
>
> Das Installationsprogramm speichert oder speichert Ihre Anmeldeinformationen nicht.

Führen Sie auf Ihrem Gerät einen der folgenden Schritte aus, um Anmeldeinformationen abzurufen und sie für das Installationsprogramm der -AWS IoT GreengrassCore-Software verfügbar zu machen:

- (Empfohlen) Verwenden Sie Temporay-Anmeldeinformationen von AWS IAM Identity Center

  i. Geben Sie die Zugriffsschlüssel-ID, den geheimen Zugriffsschlüssel und das Sitzungstoken aus dem IAM Identity Center an. Weitere Informationen finden Sie unter Manuelle Aktualisierung von Anmeldeinformationen unter Abrufen und Aktualisieren temporärer Anmeldeinformationen im IAM-Identity-Center-Benutzerhandbuch.

  ii. Führen Sie die folgenden Befehle aus, um die Anmeldeinformationen für die AWS IoT Greengrass Core-Software bereitzustellen.

  Linux or Unix

  ```
  export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
  export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/
  bPxRfiCYEXAMPLEKEY
  export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
  ```

  Windows Command Prompt (CMD)

  ```
  set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
  set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
  set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
  ```

  PowerShell

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

```
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o5OytwEXAMPLE="
```

- Verwenden Sie temporäre Sicherheitsanmeldeinformationen aus einer IAM-Rolle:

  i. Geben Sie die Zugriffsschlüssel-ID, den geheimen Zugriffsschlüssel und das Sitzungstoken aus einer von Ihnen übernommenen IAM-Rolle an. Weitere Informationen zum Abrufen dieser Anmeldeinformationen finden Sie unter Anfordern temporärer Sicherheitsanmeldeinformationen im IAM-Benutzerhandbuch.

  ii. Führen Sie die folgenden Befehle aus, um die Anmeldeinformationen für die AWS IoT Greengrass Core-Software bereitzustellen.

  Linux or Unix

  ```
  export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
  export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/
  bPxRfiCYEXAMPLEKEY
  export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
  ```

  Windows Command Prompt (CMD)

  ```
  set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
  set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
  set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
  ```

  PowerShell

  ```
  $env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
  $env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/
  bPxRfiCYEXAMPLEKEY"
  $env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o5OytwEXAMPLE="
  ```

- Verwenden Sie langfristige Anmeldeinformationen von einem IAM-Benutzer:

  i. Geben Sie die Zugriffsschlüssel-ID und den geheimen Zugriffsschlüssel für Ihren IAM-Benutzer an. Sie können einen IAM-Benutzer für die Bereitstellung erstellen, den Sie später löschen. Informationen zur IAM-Richtlinie, die dem Benutzer zugewiesen werden soll, finden Sie unter Minimale IAM-Richtlinie für das Installationsprogramm zur Bereitstellung von Ressourcen. Weitere Informationen

zum Abrufen langfristiger Anmeldeinformationen finden Sie unter Verwalten von Zugriffsschlüsseln für IAM-Benutzer im IAM-Benutzerhandbuch.

ii. Führen Sie die folgenden Befehle aus, um die Anmeldeinformationen für die AWS IoT Greengrass Core-Software bereitzustellen.

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

PowerShell

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

iii. (Optional) Wenn Sie einen IAM-Benutzer zur Bereitstellung Ihres Greengrass-Geräts erstellt haben, löschen Sie den Benutzer.

iv. (Optional) Wenn Sie die Zugriffsschlüssel-ID und den geheimen Zugriffsschlüssel eines vorhandenen IAM-Benutzers verwendet haben, aktualisieren Sie die Schlüssel für den Benutzer, damit sie nicht mehr gültig sind. Weitere Informationen finden Sie unter Aktualisieren von Zugriffsschlüsseln im AWS Identity and Access Management -Benutzerhandbuch.

c. Führen Sie unter Installationsprogramm ausführen die folgenden Schritte aus.

i. Wählen Sie unter Installationsprogramm herunterladen die Option Kopieren aus und führen Sie den kopierten Befehl auf Ihrem Core-Gerät aus. Dieser Befehl lädt die neueste Version der AWS IoT Greengrass Core-Software herunter und entpackt sie auf Ihrem Gerät.

ii. Wählen Sie unter Installationsprogramm ausführen die Option Kopieren und führen Sie den kopierten Befehl auf Ihrem Core-Gerät aus. Dieser Befehl verwendet die AWS IoT Objekt- und Objektgruppennamen, die Sie zuvor angegeben haben, um das AWS IoT

Greengrass Core-Softwareinstallationsprogramm auszuführen und AWS Ressourcen für Ihr Core-Gerät einzurichten.

Dieser Befehl führt auch Folgendes aus:

- Richten Sie die AWS IoT Greengrass Core-Software als Systemservice ein, der beim Booten ausgeführt wird. Auf Linux-Geräten erfordert dies das Systemd init-System.

  > **Important**
  >
  > Auf Windows-Core-Geräten müssen Sie die AWS IoT Greengrass Core-Software als Systemservice einrichten.

- Stellen Sie die AWS IoT Greengrass CLI-Komponente bereit. Dabei handelt es sich um ein Befehlszeilen-Tool, mit dem Sie benutzerdefinierte Greengrass-Komponenten auf dem Core-Gerät entwickeln können.
- Geben Sie an, um den `ggc_user` Systembenutzer zum Ausführen von Softwarekomponenten auf dem Core-Gerät zu verwenden. Auf Linux-Geräten gibt dieser Befehl auch an, die `ggc_group` Systemgruppe zu verwenden, und das Installationsprogramm erstellt den Systembenutzer und die Systemgruppe für Sie.

Wenn Sie diesen Befehl ausführen, sollten Sie die folgenden Meldungen sehen, um anzuzeigen, dass das Installationsprogramm erfolgreich war.

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

> **Note**
>
> Wenn Sie über ein Linux-Gerät verfügen und es nicht über systemd verfügt, richtet das Installationsprogramm die Software nicht als Systemservice ein und es wird keine Erfolgsmeldung für die Einrichtung des Kerns als Systemservice angezeigt.

Installieren der AWS IoT Greengrass Core-Software (CLI)

So installieren und konfigurieren Sie die AWS IoT Greengrass -Core-Software

1. Führen Sie auf Ihrem Greengrass-Core-Gerät den folgenden Befehl aus, um zum Stammverzeichnis zu wechseln.

   Linux or Unix

   ```
   cd ~
   ```

   Windows Command Prompt (CMD)

   ```
   cd %USERPROFILE%
   ```

   PowerShell

   ```
   cd ~
   ```

2. Laden Sie die AWS IoT Greengrass Core-Software auf Ihr Core-Gerät in eine Datei mit dem Namen herunter`greengrass-nucleus-latest.zip`.

   Linux or Unix

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
   latest.zip > greengrass-nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
   latest.zip > greengrass-nucleus-latest.zip
   ```

   PowerShell

   ```
   iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
   latest.zip -OutFile greengrass-nucleus-latest.zip
   ```

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

3. Entpacken Sie die AWS IoT Greengrass Core-Software in einen Ordner auf Ihrem Gerät. Ersetzen Sie durch *GreengrassInstaller* den Ordner, den Sie verwenden möchten.

   Linux or Unix

   ```
   unzip greengrass-nucleus-latest.zip -d GreengrassInstaller && rm greengrass-nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir GreengrassInstaller && tar -xf greengrass-nucleus-latest.zip -C GreengrassInstaller && del greengrass-nucleus-latest.zip
   ```

   PowerShell

   ```
   Expand-Archive -Path greengrass-nucleus-latest.zip -DestinationPath .\GreengrassInstaller
   rm greengrass-nucleus-latest.zip
   ```

4. Führen Sie den folgenden Befehl aus, um das AWS IoT Greengrass Core-Softwareinstallationsprogramm zu starten. Der Befehl hat folgende Auswirkungen:

   - Erstellen Sie die AWS Ressourcen, die das Core-Gerät für den Betrieb benötigt.
   - Richten Sie die AWS IoT Greengrass Core-Software als Systemservice ein, der beim Booten ausgeführt wird. Auf Linux-Geräten erfordert dies das Systemd-Init-System.

     > **Important**
     >
     > Auf Windows-Core-Geräten müssen Sie die AWS IoT Greengrass Core-Software als Systemservice einrichten.

   - Stellen Sie die AWS IoT Greengrass CLI-Komponente bereit. Dabei handelt es sich um ein Befehlszeilen-Tool, mit dem Sie benutzerdefinierte Greengrass-Komponenten auf dem Core-Gerät entwickeln können.

- Geben Sie an, um den `ggc_user` Systembenutzer zum Ausführen von Softwarekomponenten auf dem Core-Gerät zu verwenden. Auf Linux-Geräten gibt dieser Befehl auch an, die `ggc_group` Systemgruppe zu verwenden, und das Installationsprogramm erstellt den Systembenutzer und die Systemgruppe für Sie.

Ersetzen Sie die Argumentwerte in Ihrem Befehl wie folgt.

a. *`/greengrass/v2`* oder *`C:\greengrass\v2`*: Der Pfad zum Stammordner, der zur Installation der AWS IoT Greengrass Core-Software verwendet werden soll.

b. *`GreengrassInstaller`*. Der Pfad zu dem Ordner, in dem Sie das AWS IoT Greengrass Core-Softwareinstallationsprogramm entpackt haben.

c. *`Region`*. Die , AWS-Region in der Ressourcen gefunden oder erstellt werden sollen.

d. *`MyGreengrassCore`*. Der Name des AWS IoT Objekts für Ihr Greengrass-Core-Gerät. Wenn das Objekt nicht vorhanden ist, erstellt das Installationsprogramm es. Das Installationsprogramm lädt die Zertifikate herunter, um sich als das AWS IoT Objekt zu authentifizieren. Weitere Informationen finden Sie unter Geräteauthentifizierung und -autorisierung für AWS IoT Greengrass.

> **Note**
>
> Der Objektname darf keine Doppelpunktzeichen (`:`) enthalten.

e. *`MyGreengrassCoreGroup`*. Der Name der AWS IoT Objektgruppe für Ihr Greengrass-Kerngerät. Wenn die Objektgruppe nicht vorhanden ist, erstellt das Installationsprogramm sie und fügt ihr das Objekt hinzu. Wenn die Objektgruppe vorhanden ist und über eine aktive Bereitstellung verfügt, lädt das Core-Gerät die von der Bereitstellung angegebene Software herunter und führt sie aus.

> **Note**
>
> Der Objektgruppenname darf keine Doppelpunktzeichen (`:`) enthalten.

f. *`GreengrassV2IoTThingPolicy`* . Der Name der AWS IoT Richtlinie, die den Greengrass-Core-Geräten die Kommunikation mit AWS IoT und ermöglichtAWS IoT Greengrass. Wenn die AWS IoT Richtlinie nicht vorhanden ist, erstellt das Installationsprogramm eine permissive AWS IoT Richtlinie mit diesem Namen. Sie können die Berechtigungen dieser Richtlinie für Ihren Anwendungsfall einschränken. Weitere

Informationen finden Sie unter Minimale AWS IoT Richtlinie für -AWS IoT Greengrass V2Core-Geräte.

g. *GreengrassV2TokenExchangeRole*. Der Name der IAM-Rolle, die es dem Greengrass-Kerngerät ermöglicht, temporäre AWS Anmeldeinformationen zu erhalten. Wenn die Rolle nicht vorhanden ist, erstellt das Installationsprogramm sie und erstellt und fügt eine Richtlinie mit dem Namen an*GreengrassV2TokenExchangeRole*`Access`. Weitere Informationen finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

h. *GreengrassCoreTokenExchangeRoleAlias*. Der Alias für die IAM-Rolle, die es dem Greengrass-Core-Gerät ermöglicht, später temporäre Anmeldeinformationen zu erhalten. Wenn der Rollenalias nicht vorhanden ist, erstellt das Installationsprogramm ihn und verweist ihn auf die von Ihnen angegebene IAM-Rolle. Weitere Informationen finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

Linux or Unix

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
  -jar ./GreengrassInstaller/lib/Greengrass.jar \
  --aws-region region \
  --thing-name MyGreengrassCore \
  --thing-group-name MyGreengrassCoreGroup \
  --thing-policy-name GreengrassV2IoTThingPolicy \
  --tes-role-name GreengrassV2TokenExchangeRole \
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias \
  --component-default-user ggc_user:ggc_group \
  --provision true \
  --setup-system-service true \
  --deploy-dev-tools true
```

Windows Command Prompt (CMD)

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
  -jar ./GreengrassInstaller/lib/Greengrass.jar ^
  --aws-region region ^
  --thing-name MyGreengrassCore ^
  --thing-group-name MyGreengrassCoreGroup ^
  --thing-policy-name GreengrassV2IoTThingPolicy ^
  --tes-role-name GreengrassV2TokenExchangeRole ^
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias ^
  --component-default-user ggc_user ^
```

```
  --provision true ^
  --setup-system-service true ^
  --deploy-dev-tools true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
  -jar ./GreengrassInstaller/lib/Greengrass.jar `
  --aws-region region `
  --thing-name MyGreengrassCore `
  --thing-group-name MyGreengrassCoreGroup `
  --thing-policy-name GreengrassV2IoTThingPolicy `
  --tes-role-name GreengrassV2TokenExchangeRole `
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias `
  --component-default-user ggc_user `
  --provision true `
  --setup-system-service true `
  --deploy-dev-tools true
```

> **Note**
>
> Wenn Sie AWS IoT Greengrass auf einem Gerät mit eingeschränktem Speicher ausführen, können Sie die Speichermenge steuern, die die -AWS IoT GreengrassCore-Software verwendet. Um die Speicherzuweisung zu steuern, können Sie Optionen für die JVM-Heap-Größe im `jvmOptions` Konfigurationsparameter in Ihrer Kernkomponente festlegen. Weitere Informationen finden Sie unter Steuern Sie die Speicherzuweisung mit JVM-Optionen.

Wenn Sie diesen Befehl ausführen, sollten Sie die folgenden Meldungen sehen, um anzuzeigen, dass das Installationsprogramm erfolgreich war.

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

> **Note**
>
> Wenn Sie über ein Linux-Gerät verfügen und es nicht über systemd verfügt, richtet das Installationsprogramm die Software nicht als Systemservice ein und es wird keine Erfolgsmeldung für die Einrichtung des Kerns als Systemservice angezeigt.

(Optional) Ausführen der Greengrass-Software (Linux)

Wenn Sie die Software als Systemservice installiert haben, führt das Installationsprogramm die Software für Sie aus. Andernfalls müssen Sie die Software ausführen. Um zu sehen, ob das Installationsprogramm die Software als Systemservice eingerichtet hat, suchen Sie in der Ausgabe des Installationsprogramms nach der folgenden Zeile.

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

Wenn diese Meldung nicht angezeigt wird, führen Sie die folgenden Schritte aus, um die Software auszuführen:

1. Führen Sie den folgenden Befehl aus, um die Software auszuführen.

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

   Die Software gibt die folgende Meldung aus, wenn sie erfolgreich gestartet wird.

   ```
   Launched Nucleus successfully.
   ```

2. Sie müssen die aktuelle Befehls-Shell geöffnet lassen, damit die AWS IoT Greengrass Core-Software weiter ausgeführt wird. Wenn Sie SSH verwenden, um eine Verbindung zum Core-Gerät herzustellen, führen Sie den folgenden Befehl auf Ihrem Entwicklungscomputer aus, um eine zweite SSH-Sitzung zu öffnen, mit der Sie zusätzliche Befehle auf dem Core-Gerät ausführen können. Ersetzen Sie *username* durch den Namen des Benutzers, der sich anmelden soll, und ersetzen Sie *pi-ip-address* durch die IP-Adresse des Geräts.

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

Weitere Informationen zur Interaktion mit dem Greengrass-Systemservice finden Sie unter Den Greengrass Nucleus als Systemdienst konfigurieren.

Überprüfen Sie die Greengrass CLI-Installation auf dem Gerät

Die Bereitstellung der Greengrass-CLI kann bis zu einer Minute dauern. Führen Sie den folgenden Befehl aus, um den Status der Bereitstellung zu überprüfen. Ersetzen Sie *MyGreengrassCore* durch den Namen Ihres Core-Geräts.

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

Das `coreDeviceExecutionStatus` gibt den Status der Bereitstellung auf dem Kerngerät an. Wenn der Status lautet`SUCCEEDED`, führen Sie den folgenden Befehl aus, um zu überprüfen, ob die Greengrass-CLI installiert ist und ausgeführt wird. */greengrass/v2*Ersetzen Sie ihn durch den Pfad zum Stammordner.

Linux or Unix

```
/greengrass/v2/bin/greengrass-cli help
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli help
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli help
```

Der Befehl gibt Hilfeinformationen für die Greengrass-CLI aus. Wenn das `greengrass-cli` nicht gefunden wird, konnte die Greengrass-CLI bei der Bereitstellung möglicherweise nicht installiert werden. Weitere Informationen finden Sie unter Problembehebung AWS IoT Greengrass V2.

Sie können auch den folgenden Befehl ausführen, um die AWS IoT Greengrass CLI manuell auf Ihrem Gerät bereitzustellen.

- Ersetzen Sie die *Region* durch die AWS-Region , die Sie verwenden. Stellen Sie sicher, dass Sie dasselbe verwenden AWS-Region , das Sie zur Konfiguration AWS CLI auf Ihrem Gerät verwendet haben.

- Ersetzen Sie die *Konto-ID* durch Ihre AWS-Konto ID.
- Ersetzen Sie *MyGreengrassCore* durch den Namen Ihres Core-Geräts.

Linux, macOS, or Unix

```
aws greengrassv2 create-deployment \
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" \
  --components '{
    "aws.greengrass.Cli": {
      "componentVersion": "2.12.6"
    }
  }'
```

Windows Command Prompt (CMD)

```
aws greengrassv2 create-deployment ^
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" ^
  --components "{\"aws.greengrass.Cli\":{\"componentVersion\":\"2.12.6\"}}"
```

PowerShell

```
aws greengrassv2 create-deployment `
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" `
  --components '{\"aws.greengrass.Cli\":{\"componentVersion\":\"2.12.6\"}}'
```

> **Tip**
>
> Sie können Ihrer PATH Umgebungsvariablen */greengrass/v2*/bin (Linux) oder *C:\greengrass\v2*\bin (Windows) hinzufügen, um sie `greengrass-cli` ohne ihren absoluten Pfad auszuführen.

DieAWS IoT Greengrass Core-Software und die lokalen Entwicklungstools werden auf Ihrem Gerät ausgeführt. Als Nächstes können Sie eine HelloAWS IoT Greengrass World-Komponente auf Ihrem Gerät entwickeln.

Schritt 4: Entwickeln und Testen einer Komponente auf Ihrem Gerät

Eine Komponente ist ein Softwaremodul, das auf - AWS IoT Greengrass Core-Geräten ausgeführt wird. Mit Komponenten können Sie komplexe Anwendungen als diskrete Bausteine erstellen und verwalten, die Sie von einem Greengrass-Kerngerät zu einem anderen wiederverwenden können. Jede Komponente besteht aus einem Rezept und Artefakten .

- Rezepte

  Jede Komponente enthält eine Rezeptdatei, die ihre Metadaten definiert. Das Rezept gibt auch die Konfigurationsparameter, Komponentenabhängigkeiten, den Lebenszyklus und die Plattformkompatibilität der Komponente an. Der Komponentenlebenszyklus definiert die Befehle, die die Komponente installieren, ausführen und herunterfahren. Weitere Informationen finden Sie unter AWS IoT Greengrass Referenz zum Komponenten-Rezept.

  Sie können Rezepte im JSON- oder YAML-Format definieren.

- Artefakte

  Komponenten können eine beliebige Anzahl von Artefakten haben, bei denen es sich um Komponenten-Binärdateien handelt. Artefakte können Skripte, kompilierten Code, statische Ressourcen und alle anderen Dateien enthalten, die eine Komponente verbraucht. Komponenten können auch Artefakte aus Komponentenabhängigkeiten verwenden.

Mit können Sie die Greengrass-CLI verwenden AWS IoT Greengrass, um Komponenten lokal auf einem Greengrass-Kerngerät zu entwickeln und zu testen, ohne mit der AWS Cloud zu interagieren. Wenn Sie Ihre lokale Komponente abgeschlossen haben, können Sie das Komponentenrezept und die Artefakte verwenden, um diese Komponente im AWS IoT Greengrass Service in der AWS Cloud zu erstellen und sie dann auf allen Ihren Greengrass-Core-Geräten bereitzustellen. Weitere Informationen zu -Komponenten finden Sie unter Entwickeln von AWS IoT Greengrass Komponenten.

In diesem Abschnitt erfahren Sie, wie Sie eine grundlegende Hello-World-Komponente lokal auf Ihrem Core-Gerät erstellen und ausführen.

So entwickeln Sie eine Hello-World-Komponente auf Ihrem Gerät

1. Erstellen Sie einen Ordner für Ihre Komponenten mit Unterordnern für Rezepte und Artefakte. Führen Sie die folgenden Befehle auf Ihrem Greengrass-Core-Gerät aus, um diese Ordner zu

erstellen und zum Komponentenordner zu wechseln. Ersetzen Sie *~/greengrassv2* oder *%USERPROFILE%\greengrassv2* durch den Pfad zum Ordner, der für die lokale Entwicklung verwendet werden soll.

Linux or Unix

```
mkdir -p ~/greengrassv2/{recipes,artifacts}
cd ~/greengrassv2
```

Windows Command Prompt (CMD)

```
mkdir %USERPROFILE%\greengrassv2\\recipes, %USERPROFILE%\greengrassv2\\artifacts
cd %USERPROFILE%\greengrassv2
```

PowerShell

```
mkdir ~/greengrassv2/recipes, ~/greengrassv2/artifacts
cd ~/greengrassv2
```

2. Verwenden Sie einen Texteditor, um eine Rezeptdatei zu erstellen, die die Metadaten, Parameter, Abhängigkeiten, den Lebenszyklus und die Plattformfunktionen Ihrer Komponente definiert. Fügen Sie die Komponentenversion in den Rezeptdateinamen ein, damit Sie identifizieren können, welches Rezept welche Komponentenversion widerspiegelt. Sie können das YAML- oder JSON-Format für Ihr Rezept auswählen.

   Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen der Datei zu verwenden.

   JSON

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

   YAML

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

> **Note**
>
> AWS IoT Greengrass verwendet semantische Versionen für -Komponenten. Semantische Versionen folgen einem größeren Patch-Nummernsystem. Die -Version `1.0.0` stellt beispielsweise die erste Hauptversion für eine Komponente dar. Weitere Informationen finden Sie in der semantischen Versionsspezifikation .

3. Fügen Sie das folgende Rezept in die Datei ein.

   JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "My first AWS IoT Greengrass component.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "Message": "world"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/hello_world.py {configuration:/
Message}"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "run": "py -3 -u {artifacts:path}/hello_world.py {configuration:/
Message}"
      }
    }
```

```
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
ComponentDescription: My first AWS IoT Greengrass component.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    Message: world
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      run: |
        python3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
  - Platform:
      os: windows
    Lifecycle:
      run: |
        py -3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
```

Der `ComponentConfiguration` Abschnitt dieses Rezepts definiert einen Parameter, `Message`, der standardmäßig ist`world`. Der `Manifests` Abschnitt definiert ein Manifest , bei dem es sich um eine Reihe von Lebenszyklusanweisungen und Artefakten für eine Plattform handelt. Sie können mehrere Manifeste definieren, um beispielsweise unterschiedliche Installationsanweisungen für verschiedene Plattformen anzugeben. Im Manifest weist der `Lifecycle` Abschnitt das Greengrass-Kerngerät an, das Hello-World-Skript mit dem `Message` Parameterwert als Argument auszuführen.

4. Führen Sie den folgenden Befehl aus, um einen Ordner für die Komponentenartefakte zu erstellen.

   Linux or Unix

   ```
   mkdir -p artifacts/com.example.HelloWorld/1.0.0
   ```

Windows Command Prompt (CMD)

```
mkdir artifacts\com.example.HelloWorld\1.0.0
```

PowerShell

```
mkdir artifacts\com.example.HelloWorld\1.0.0
```

> **Important**
>
> Sie müssen das folgende Format für den Pfad des Artefaktordners verwenden. Fügen Sie den Komponentennamen und die Version ein, die Sie im Rezept angeben.
>
> ```
> artifacts/componentName/componentVersion/
> ```

5. Verwenden Sie einen Texteditor, um eine Python-Skriptartefaktdatei für Ihre Hello-World-Komponente zu erstellen.

   Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen der Datei zu verwenden.

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

   Kopieren Sie das folgende Python-Skript und fügen Sie es in die Datei ein.

   ```
   import sys

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

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

6. Verwenden Sie die lokale AWS IoT Greengrass CLI, um Komponenten auf Ihrem Greengrass-Kerngerät zu verwalten.

   Führen Sie den folgenden Befehl aus, um die Komponente auf dem AWS IoT Greengrass Core bereitzustellen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch Ihren AWS

IoT Greengrass V2 Stammordner und ersetzen Sie *~/greengrassv2* oder *%USERPROFILE%\greengrassv2* durch Ihren Komponentenentwicklungsordner.

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli deployment create \
  --recipeDir ~/greengrassv2/recipes \
  --artifactDir ~/greengrassv2/artifacts \
  --merge "com.example.HelloWorld=1.0.0"
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli deployment create ^
  --recipeDir %USERPROFILE%\greengrassv2\recipes ^
  --artifactDir %USERPROFILE%\greengrassv2\artifacts ^
  --merge "com.example.HelloWorld=1.0.0"
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli deployment create `
  --recipeDir ~/greengrassv2/recipes `
  --artifactDir ~/greengrassv2/artifacts `
  --merge "com.example.HelloWorld=1.0.0"
```

Dieser Befehl fügt die Komponente hinzu, die das -Rezept in `recipes` und das Python-Skript in verwendet`artifacts`. Die `--merge` Option fügt die von Ihnen angegebene Komponente und Version hinzu oder aktualisiert sie.

7. Die AWS IoT Greengrass Core-Software speichert stdout aus dem Komponentenprozess in Protokolldateien im `logs` Ordner . Führen Sie den folgenden Befehl aus, um zu überprüfen, ob die Hello-World-Komponente ausgeführt wird und Nachrichten druckt.

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
   ```

   Windows Command Prompt (CMD)

   ```
   type C:\greengrass\v2\logs\com.example.HelloWorld.log
   ```

Der `type` Befehl schreibt den Inhalt der Datei in das Terminal. Führen Sie diesen Befehl mehrmals aus, um Änderungen in der Datei zu beobachten.

PowerShell

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

Sie sollten Meldungen ähnlich dem folgenden Beispiel sehen.

```
Hello, world!
```

Note

Wenn die Datei nicht vorhanden ist, ist die lokale Bereitstellung möglicherweise noch nicht abgeschlossen. Wenn die Datei nicht innerhalb von 15 Sekunden existiert, ist die Bereitstellung wahrscheinlich fehlgeschlagen. Dies kann beispielsweise der Fall sein, wenn Ihr Rezept ungültig ist. Führen Sie den folgenden Befehl aus, um die AWS IoT Greengrass Core-Protokolldatei anzuzeigen. Diese Datei enthält Protokolle aus dem Bereitstellungsservice des Greengrass-Core-Geräts.

Linux or Unix

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

Windows Command Prompt (CMD)

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

Der `type` Befehl schreibt den Inhalt der Datei in das Terminal. Führen Sie diesen Befehl mehrmals aus, um Änderungen in der Datei zu beobachten.

PowerShell

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

8. Ändern Sie die lokale Komponente, um Ihren Code zu iterieren und zu testen. Öffnen Sie `hello_world.py` in einem Texteditor und fügen Sie den folgenden Code in Zeile 4 hinzu, um die Nachricht zu bearbeiten, die der AWS IoT Greengrass Kern protokolliert.

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

   Das `hello_world.py` Skript sollte jetzt den folgenden Inhalt haben.

   ```
   import sys

   message = "Hello, %s!" % sys.argv[1]
   message += " Greetings from your first Greengrass component."

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

9. Führen Sie den folgenden Befehl aus, um die Komponente mit Ihren Änderungen zu aktualisieren.

   Linux or Unix

   ```
   sudo /greengrass/v2/bin/greengrass-cli deployment create \
     --recipeDir ~/greengrassv2/recipes \
     --artifactDir ~/greengrassv2/artifacts \
     --merge "com.example.HelloWorld=1.0.0"
   ```

   Windows Command Prompt (CMD)

   ```
   C:\greengrass\v2\bin\greengrass-cli deployment create ^
     --recipeDir %USERPROFILE%\greengrassv2\recipes ^
     --artifactDir %USERPROFILE%\greengrassv2\artifacts ^
     --merge "com.example.HelloWorld=1.0.0"
   ```

   PowerShell

   ```
   C:\greengrass\v2\bin\greengrass-cli deployment create `
     --recipeDir ~/greengrassv2/recipes `
     --artifactDir ~/greengrassv2/artifacts `
     --merge "com.example.HelloWorld=1.0.0"
   ```

Dieser Befehl aktualisiert die `com.example.HelloWorld` Komponente mit dem neuesten Hello World-Artefakt.

10. Führen Sie den folgenden Befehl aus, um die Komponente neu zu starten. Wenn Sie eine Komponente neu starten, verwendet das Core-Gerät die neuesten Änderungen.

    Linux or Unix

    ```
    sudo /greengrass/v2/bin/greengrass-cli component restart \
      --names "com.example.HelloWorld"
    ```

    Windows Command Prompt (CMD)

    ```
    C:\greengrass\v2\bin\greengrass-cli component restart ^
      --names "com.example.HelloWorld"
    ```

    PowerShell

    ```
    C:\greengrass\v2\bin\greengrass-cli component restart `
      --names "com.example.HelloWorld"
    ```

11. Überprüfen Sie das Protokoll erneut, um zu überprüfen, ob die Hello-World-Komponente die neue Nachricht ausgibt.

    Linux or Unix

    ```
    sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
    ```

    Windows Command Prompt (CMD)

    ```
    type C:\greengrass\v2\logs\com.example.HelloWorld.log
    ```

    Der `type` Befehl schreibt den Inhalt der Datei in das Terminal. Führen Sie diesen Befehl mehrmals aus, um Änderungen in der Datei zu beobachten.

    PowerShell

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

Sie sollten Meldungen ähnlich dem folgenden Beispiel sehen.

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

12. Sie können die Konfigurationsparameter der Komponente aktualisieren, um verschiedene Konfigurationen zu testen. Wenn Sie eine Komponente bereitstellen, können Sie eine Konfigurationsaktualisierung angeben, die definiert, wie die Konfiguration der Komponente auf dem Core-Gerät geändert wird. Sie können angeben, welche Konfigurationswerte auf Standardwerte zurückgesetzt werden sollen, und die neuen Konfigurationswerte, die auf dem Core-Gerät zusammengeführt werden sollen. Weitere Informationen finden Sie unter Komponentenkonfigurationen aktualisieren.

    Gehen Sie wie folgt vor:

    a. Verwenden Sie einen Texteditor, um eine Datei mit dem Namen zu erstellen`hello-world-config-update.json`, die das Konfigurationsupdate enthält

        Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen der Datei zu verwenden.

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

    b. Kopieren Sie das folgende JSON-Objekt und fügen Sie es in die Datei ein. Dieses JSON-Objekt definiert eine Konfigurationsaktualisierung, die den Wert mit dem `Message` Parameter zusammenführt`friend`, um seinen Wert zu aktualisieren. Dieses Konfigurationsupdate gibt keine Werte an, die zurückgesetzt werden sollen. Sie müssen den `Message` Parameter nicht zurücksetzen, da die Zusammenführungsaktualisierung den vorhandenen Wert ersetzt.

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

    c. Führen Sie den folgenden Befehl aus, um das Konfigurationsupdate für die Hello World-Komponente bereitzustellen.

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli deployment create \
  --merge "com.example.HelloWorld=1.0.0" \
  --update-config hello-world-config-update.json
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli deployment create ^
  --merge "com.example.HelloWorld=1.0.0" ^
  --update-config hello-world-config-update.json
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli deployment create `
  --merge "com.example.HelloWorld=1.0.0" `
  --update-config hello-world-config-update.json
```

d. Überprüfen Sie das Protokoll erneut, um zu überprüfen, ob die Hello World-Komponente die neue Nachricht ausgibt.

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
```

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\logs\com.example.HelloWorld.log
```

Der `type` Befehl schreibt den Inhalt der Datei in das Terminal. Führen Sie diesen Befehl mehrmals aus, um Änderungen in der Datei zu beobachten.

PowerShell

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

Sie sollten Meldungen ähnlich dem folgenden Beispiel sehen.

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

13. Nachdem Sie Ihre Komponente getestet haben, entfernen Sie sie von Ihrem Core-Gerät. Führen Sie den folgenden Befehl aus.

    Linux or Unix

    ```
    sudo /greengrass/v2/bin/greengrass-cli deployment create --
    remove="com.example.HelloWorld"
    ```

    Windows Command Prompt (CMD)

    ```
    C:\greengrass\v2\bin\greengrass-cli deployment create --
    remove="com.example.HelloWorld"
    ```

    PowerShell

    ```
    C:\greengrass\v2\bin\greengrass-cli deployment create --
    remove="com.example.HelloWorld"
    ```

    > **Important**
    >
    > Dieser Schritt ist erforderlich, damit Sie die Komponente nach dem Hochladen auf wieder auf dem Core-Gerät bereitstellen können AWS IoT Greengrass. Andernfalls schlägt die Bereitstellung mit einem Versionskompatibilitätsfehler fehl, da die lokale Bereitstellung eine andere Version der Komponente angibt.

    Führen Sie den folgenden Befehl aus und stellen Sie sicher, dass die `com.example.HelloWorld` Komponente nicht in der Liste der Komponenten auf Ihrem Gerät angezeigt wird.

    Linux or Unix

    ```
    sudo /greengrass/v2/bin/greengrass-cli component list
    ```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli component list
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli component list
```

Ihre Hello-World-Komponente ist abgeschlossen und Sie können sie jetzt in den AWS IoT Greengrass Cloud-Service hochladen. Anschließend können Sie die Komponente auf Greengrass-Core-Geräten bereitstellen.

Schritt 5: Erstellen Sie Ihre Komponente im AWS IoT Greengrass Service

Wenn Sie mit der Entwicklung einer Komponente auf Ihrem Kerngerät fertig sind, können Sie sie in den AWS IoT Greengrass Dienst hochladen AWS Cloud. Sie können die Komponente auch direkt in der AWS IoT Greengrass Konsole erstellen. AWS IoT Greengrass bietet einen Komponentenverwaltungsdienst, der Ihre Komponenten hostet, sodass Sie sie auf einzelnen Geräten oder Geräteflotten bereitstellen können. Um eine Komponente in den AWS IoT Greengrass Dienst hochzuladen, führen Sie die folgenden Schritte aus:

- Laden Sie Komponentenartefakte in einen S3-Bucket hoch.
- Fügen Sie den Amazon Simple Storage Service (Amazon S3) -URI jedes Artefakts zum Komponentenrezept hinzu.
- Erstellen Sie eine Komponente AWS IoT Greengrass aus dem Komponentenrezept.

In diesem Abschnitt führen Sie diese Schritte auf Ihrem Greengrass-Core-Gerät aus, um Ihre Hello World-Komponente in den AWS IoT Greengrass Dienst hochzuladen.

Erstellen Sie Ihre Komponente in AWS IoT Greengrass (Konsole)

1. Verwenden Sie einen S3-Bucket in Ihrem AWS Konto, um AWS IoT Greengrass Komponentenartefakte zu hosten. Wenn Sie die Komponente auf einem Kerngerät bereitstellen, lädt das Gerät die Artefakte der Komponente aus dem Bucket herunter.

Sie können einen vorhandenen S3-Bucket verwenden oder einen neuen Bucket erstellen.

a. Wählen Sie in der Amazon S3 S3-Konsole unter Buckets die Option Create Bucket aus.

b. Geben Sie als Bucket-Namen einen eindeutigen Bucket-Namen ein. Sie können beispielsweise die Datei **`greengrass-component-artifacts-`*`region`*`-`*`123456789012`** verwenden. Ersetzen Sie *`123456789012`* durch Ihre AWS Konto-ID und die *`Region`* durch die AWS-Region , die Sie für dieses Tutorial verwenden.

c. Wählen Sie als AWS Region die AWS Region aus, die Sie für dieses Tutorial verwenden.

d. Wählen Sie Bucket erstellen aus.

e. Wählen Sie unter Buckets den Bucket aus, den Sie erstellt haben, und laden Sie das `hello_world.py` Skript in den `artifacts/com.example.HelloWorld/1.0.0` Ordner im Bucket hoch. Informationen zum Hochladen von Objekten in S3-Buckets finden Sie unter Hochladen von Objekten im Amazon Simple Storage Service-Benutzerhandbuch.

f. Kopieren Sie die S3-URI des `hello_world.py` Objekts im S3-Bucket. Diese URI sollte dem folgenden Beispiel ähneln. Ersetzen Sie DOC-EXAMPLE-BUCKET durch den Namen des S3-Buckets.

```
s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
```

2. Erlauben Sie dem Core-Gerät den Zugriff auf Komponentenartefakte im S3-Bucket.

   Jedes Kerngerät hat eine IAM-Rolle für das Kerngerät, die es ihm ermöglicht, mit der Cloud zu interagieren AWS IoT und Logs an die AWS Cloud zu senden. Diese Geräterolle erlaubt standardmäßig keinen Zugriff auf S3-Buckets. Sie müssen also eine Richtlinie erstellen und anhängen, die es dem Kerngerät ermöglicht, Komponentenartefakte aus dem S3-Bucket abzurufen.

   Wenn die Rolle Ihres Geräts bereits den Zugriff auf den S3-Bucket ermöglicht, können Sie diesen Schritt überspringen. Andernfalls erstellen Sie eine IAM-Richtlinie, die den Zugriff ermöglicht, und hängen Sie sie wie folgt an die Rolle an:

   a. Wählen Sie im Navigationsmenü der IAM-Konsole Richtlinien und dann Richtlinie erstellen aus.

   b. Ersetzen Sie auf der Registerkarte JSON den Platzhalterinhalt durch die folgende Richtlinie. Ersetzen Sie DOC-EXAMPLE-BUCKET durch den Namen des S3-Buckets, der Komponentenartefakte für das Kerngerät zum Herunterladen enthält.

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

c. Wählen Sie Weiter aus.

d. Geben Sie im Abschnitt Richtliniendetails für Name den Text ein. **`MyGreengrassV2ComponentArtifactPolicy`**

e. Wählen Sie Richtlinie erstellen aus.

f. Wählen Sie im Navigationsmenü der IAM-Konsole die Option Rolle und dann den Namen der Rolle für das Kerngerät aus. Sie haben diesen Rollennamen bei der Installation der AWS IoT Greengrass Core-Software angegeben. Wenn Sie keinen Namen angegeben haben, lautet die Standardeinstellung`GreengrassV2TokenExchangeRole`.

g. Wählen Sie unter Berechtigungen die Option Berechtigungen hinzufügen und anschließend Richtlinien anhängen aus.

h. Aktivieren Sie auf der Seite Berechtigungen hinzufügen das Kontrollkästchen neben der `MyGreengrassV2ComponentArtifactPolicy` Richtlinie, die Sie erstellt haben, und wählen Sie dann Berechtigungen hinzufügen aus.

3. Verwenden Sie das Komponentenrezept, um eine Komponente in der AWS IoT Greengrass Konsole zu erstellen.

   a. Wählen Sie im Navigationsmenü der AWS IoT Greengrass Konsole Komponenten und anschließend Komponente erstellen aus.

   b. Wählen Sie unter Komponenteninformationen die Option Rezept als JSON eingeben aus. Das Platzhalterrezept sollte dem folgenden Beispiel ähneln.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.HelloWorld",
```

```
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "My first AWS IoT Greengrass component.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "Message": "world"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/hello_world.py \"{configuration:/
Message}\""
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.HelloWorld/1.0.0/hello_world.py"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "run": "py -3 -u {artifacts:path}/hello_world.py \"{configuration:/
Message}\""
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.HelloWorld/1.0.0/hello_world.py"
        }
      ]
    }
  ]
}
```

c. Ersetzen Sie die Platzhalter-URI in jedem `Artifacts` Abschnitt durch die S3-URI Ihres `hello_world.py` Objekts.

d. Wählen Sie Komponente erstellen.

e. Auf dem com.example. HelloWorldÜberprüfen Sie auf der Komponentenseite, ob der Status der Komponente Deployable lautet.

Erstellen Sie Ihre Komponente in AWS IoT Greengrass ()AWS CLI

Um deine Hello World-Komponente hochzuladen

1. Verwenden Sie einen S3-Bucket in Ihrem AWS-Konto , um AWS IoT Greengrass Komponentenartefakte zu hosten. Wenn Sie die Komponente auf einem Core-Gerät bereitstellen, lädt das Gerät die Artefakte der Komponente aus dem Bucket herunter.

   Sie können einen vorhandenen S3-Bucket verwenden oder den folgenden Befehl ausführen, um einen Bucket zu erstellen. Dieser Befehl erstellt einen Bucket mit Ihrer AWS-Konto ID und AWS-Region bildet einen eindeutigen Bucket-Namen. Ersetzen Sie *123456789012* durch Ihre AWS-Konto ID und die *Region* durch die AWS-Region , die Sie für dieses Tutorial verwenden.

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

   Der Befehl gibt die folgenden Informationen aus, wenn die Anfrage erfolgreich ist.

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

2. Erlauben Sie dem Core-Gerät den Zugriff auf Komponentenartefakte im S3-Bucket.

   Jedes Kerngerät hat eine IAM-Rolle für das Kerngerät, die es ermöglicht, mit dem zu interagieren AWS IoT und Protokolle an das AWS Cloud zu senden. Diese Geräterolle erlaubt standardmäßig keinen Zugriff auf S3-Buckets. Sie müssen also eine Richtlinie erstellen und anhängen, die es dem Kerngerät ermöglicht, Komponentenartefakte aus dem S3-Bucket abzurufen.

   Wenn die Rolle des Kerngeräts bereits den Zugriff auf den S3-Bucket ermöglicht, können Sie diesen Schritt überspringen. Andernfalls erstellen Sie eine IAM-Richtlinie, die den Zugriff ermöglicht, und hängen Sie sie wie folgt an die Rolle an:

   a. Erstellen Sie eine Datei mit dem Namen `component-artifact-policy.json` und kopieren Sie den folgenden JSON-Code in die Datei. Diese Richtlinie ermöglicht den Zugriff auf alle Dateien in einem S3-Bucket. Ersetzen Sie DOC-EXAMPLE-BUCKET durch den Namen des S3-Buckets.

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

b. Führen Sie den folgenden Befehl aus, um die Richtlinie aus dem Richtliniendokument in zu erstellen. `component-artifact-policy.json`

Linux or Unix

```
aws iam create-policy \\
  --policy-name MyGreengrassV2ComponentArtifactPolicy \\
  --policy-document file://component-artifact-policy.json
```

Windows Command Prompt (CMD)

```
aws iam create-policy ^
  --policy-name MyGreengrassV2ComponentArtifactPolicy ^
  --policy-document file://component-artifact-policy.json
```

PowerShell

```
aws iam create-policy `
  --policy-name MyGreengrassV2ComponentArtifactPolicy `
  --policy-document file://component-artifact-policy.json
```

Kopieren Sie den Amazon Resource Name (ARN) der Richtlinie aus den Richtlinienmetadaten in der Ausgabe. Sie verwenden diesen ARN, um diese Richtlinie im nächsten Schritt an die zentrale Geräterolle anzuhängen.

c. Führen Sie den folgenden Befehl aus, um die Richtlinie an die zentrale Geräterolle anzuhängen. Ersetzen Sie *GreengrassV2 TokenExchangeRole* durch den Namen der

Rolle für das Kerngerät. Sie haben diesen Rollennamen bei der Installation der AWS IoT Greengrass Core-Software angegeben. Ersetzen Sie den Richtlinien-ARN durch den ARN aus dem vorherigen Schritt.

Linux or Unix

```
aws iam attach-role-policy \\
  --role-name GreengrassV2TokenExchangeRole \\
  --policy-arn
 arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
```

Windows Command Prompt (CMD)

```
aws iam attach-role-policy ^
  --role-name GreengrassV2TokenExchangeRole ^
  --policy-arn
 arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
```

PowerShell

```
aws iam attach-role-policy `
  --role-name GreengrassV2TokenExchangeRole `
  --policy-arn
 arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
```

Wenn der Befehl keine Ausgabe hat, war er erfolgreich. Das Core-Gerät kann jetzt auf Artefakte zugreifen, die Sie in diesen S3-Bucket hochladen.

3. Laden Sie das Python-Skript-Artefakt Hello World in den S3-Bucket hoch.

   Führen Sie den folgenden Befehl aus, um das Skript in denselben Pfad im Bucket hochzuladen, in dem sich das Skript auf Ihrem AWS IoT Greengrass Core befindet. Ersetzen Sie DOC-EXAMPLE-BUCKET durch den Namen des S3-Buckets.

   Linux or Unix

   ```
   aws s3 cp \
     artifacts/com.example.HelloWorld/1.0.0/hello_world.py \
     s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
   ```

Windows Command Prompt (CMD)

```
aws s3 cp ^
  artifacts/com.example.HelloWorld/1.0.0/hello_world.py ^
  s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
```

PowerShell

```
aws s3 cp `
  artifacts/com.example.HelloWorld/1.0.0/hello_world.py `
  s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
```

Der Befehl gibt eine Zeile aus, die mit `upload:` „Wenn die Anfrage erfolgreich ist“ beginnt.

4. Fügen Sie den Amazon S3 S3-URI des Artefakts zum Komponentenrezept hinzu.

   Die Amazon S3 S3-URI besteht aus dem Bucket-Namen und dem Pfad zum Artefaktobjekt im Bucket. Die Amazon S3 S3-URI Ihres Skript-Artefakts ist die URI, in die Sie das Artefakt im vorherigen Schritt hochgeladen haben. Diese URI sollte dem folgenden Beispiel ähneln. Ersetzen Sie DOC-EXAMPLE-BUCKET durch den Namen des S3-Buckets.

   ```
   s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
   ```

   Um das Artefakt zum Rezept hinzuzufügen, fügen Sie eine Liste hinzu, `Artifacts` die eine Struktur mit dem Amazon S3 S3-URI enthält.

   JSON

   ```
   "Artifacts": [
     {
       "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/
   hello_world.py"
     }
   ]
   ```

   Öffnen Sie die Rezeptdatei in einem Texteditor.

   Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um die Datei mit GNU Nano zu erstellen.

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

Fügen Sie das Artefakt dem Rezept hinzu. Ihre Rezeptdatei sollte dem folgenden Beispiel ähneln.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "My first AWS IoT Greengrass component.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "Message": "world"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/hello_world.py \"{configuration:/
Message}\""
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.HelloWorld/1.0.0/hello_world.py"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "run": "py -3 -u {artifacts:path}/hello_world.py \"{configuration:/
Message}\""
      },
      "Artifacts": [
        {
```

```
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.HelloWorld/1.0.0/hello_world.py"
        }
      ]
    }
  ]
}
```

YAML

```
Artifacts:
  - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/
hello_world.py
```

Öffnen Sie die Rezeptdatei in einem Texteditor.

Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um die Datei mit GNU Nano zu erstellen.

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

Fügen Sie das Artefakt dem Rezept hinzu. Ihre Rezeptdatei sollte dem folgenden Beispiel ähneln.

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
ComponentDescription: My first AWS IoT Greengrass component.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    Message: world
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      run: |
        python3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
    Artifacts:
```

```
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/
hello_world.py
  - Platform:
      os: windows
    Lifecycle:
      run: |
        py -3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/
hello_world.py
```

5. Erstellen Sie eine Komponentenressource AWS IoT Greengrass aus dem Rezept. Führen Sie den folgenden Befehl aus, um die Komponente aus dem Rezept zu erstellen, das Sie als Binärdatei bereitstellen.

   JSON

   ```
   aws greengrassv2 create-component-version --inline-recipe fileb://recipes/
   com.example.HelloWorld-1.0.0.json
   ```

   YAML

   ```
   aws greengrassv2 create-component-version --inline-recipe fileb://recipes/
   com.example.HelloWorld-1.0.0.yaml
   ```

   Die Antwort ähnelt dem folgenden Beispiel, wenn die Anfrage erfolgreich ist.

   ```
   {
     "arn":
    "arn:aws:greengrass:region:123456789012:components:com.example.HelloWorld:versions:1.0.0",
     "componentName": "com.example.HelloWorld",
     "componentVersion": "1.0.0",
     "creationTimestamp": "Mon Nov 30 09:04:05 UTC 2020",
     "status": {
       "componentState": "REQUESTED",
       "message": "NONE",
       "errors": {}
     }
   }
   ```

Kopieren Sie das `arn` aus der Ausgabe, um im nächsten Schritt den Status der Komponente zu überprüfen.

> **Note**
>
> Sie können Ihre Hello World-Komponente auch in der AWS IoT Greengrass Konsole auf der Komponentenseite sehen.

6. Stellen Sie sicher, dass die Komponente erstellt wird und bereit ist, bereitgestellt zu werden. Wenn Sie eine Komponente erstellen, lautet ihr Status`REQUESTED`. AWS IoT Greengrass Überprüft dann, ob die Komponente bereitgestellt werden kann. Sie können den folgenden Befehl ausführen, um den Status der Komponente abzufragen und zu überprüfen, ob Ihre Komponente bereitgestellt werden kann. Ersetzen Sie das `arn` durch den ARN aus dem vorherigen Schritt.

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

Wenn die Komponente validiert wird, gibt die Antwort an, dass der Status der Komponente lautet`DEPLOYABLE`.

```
{
  "arn":
 "arn:aws:greengrass:region:123456789012:components:com.example.HelloWorld:versions:1.0.0",
  "componentName": "com.example.HelloWorld",
  "componentVersion": "1.0.0",
  "creationTimestamp": "2020-11-30T18:04:05.823Z",
  "publisher": "Amazon",
  "description": "My first Greengrass component.",
  "status": {
    "componentState": "DEPLOYABLE",
    "message": "NONE",
    "errors": {}
  },
  "platforms": [
    {
      "os": "linux",
      "architecture": "all"
    }
  ]
```

```
}
```

Ihre Hello World-Komponente ist jetzt in AWS IoT Greengrass verfügbar. Sie können es wieder auf diesem Greengrass-Core-Gerät oder auf anderen Core-Geräten bereitstellen.

Schritt 6: Bereitstellen Ihrer Komponente

Mit können AWS IoT GreengrassSie Komponenten auf einzelnen Geräten oder Gerätegruppen bereitstellen. Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert und führt die Software dieser Komponente auf jedem Zielgerät aus. Sie geben an, welche Komponenten bereitgestellt werden sollen, und das Konfigurationsupdate, das für jede Komponente bereitgestellt werden soll. Sie können auch steuern, wie die Bereitstellung auf den Geräten bereitgestellt wird, auf die die Bereitstellung abzielt. Weitere Informationen finden Sie unter Bereitstellen von AWS IoT Greengrass Komponenten auf Geräten.

In diesem Abschnitt stellen Sie Ihre Hello-World-Komponente wieder auf Ihrem Greengrass-Kerngerät bereit.

Bereitstellen Ihrer Komponente (Konsole)

1. Wählen Sie im Navigationsmenü der AWS IoT GreengrassKonsole Komponenten aus.
2. Wählen Sie auf der Seite Komponenten auf der Registerkarte Meine Komponenten die Option auscom.example.HelloWorld.
3. Wählen Sie auf der com.example.HelloWorld Seite Bereitstellen aus.
4. Wählen Sie unter Zur Bereitstellung hinzufügen die Option Neue Bereitstellung erstellen und dann Weiter aus.
5. Gehen Sie auf der Seite Ziel angeben wie folgt vor:
   a. Geben Sie im Feld Name (Name) **`Deployment for MyGreengrassCore`** ein.
   b. Wählen Sie für Bereitstellungsziel die Option Core-Gerät und den Namen des AWS IoT Objekts für Ihr Core-Gerät aus. Der Standardwert in diesem Tutorial ist *`MyGreengrassCore`*.
   c. Wählen Sie Weiter aus.
6. Überprüfen Sie auf der Seite Komponenten auswählen unter Meine Komponenten, ob die com.example.HelloWorld Komponente ausgewählt ist, und wählen Sie Weiter aus.

7. Wählen Sie auf der Seite Komponenten konfigurieren die Option aus com.example.HelloWorldund gehen Sie wie folgt vor:

   a. Wählen Sie Komponente konfigurieren aus.

   b. Geben Sie unter Konfigurationsupdate unter Zusammenzuführende Konfiguration die folgende Konfiguration ein.

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

   Dieses Konfigurationsupdate legt den `Message` Parameter Hello World `universe` für das Gerät in dieser Bereitstellung auf fest.

   c. Wählen Sie Bestätigen aus.

   d. Wählen Sie Weiter aus.

8. Behalten Sie auf der Seite Erweiterte Einstellungen konfigurieren die Standardkonfigurationseinstellungen bei und wählen Sie Weiter.

9. Wählen Sie auf der Seite Review (Prüfen) die Option Deploy (Bereitstellen) aus.

10. Überprüfen Sie, ob die Bereitstellung erfolgreich abgeschlossen wurde. Es kann einige Minuten dauern, bis die Bereitstellung abgeschlossen ist. Überprüfen Sie das Hello World-Protokoll, um die Änderung zu überprüfen. Führen Sie den folgenden Befehl auf Ihrem Greengrass-Core-Gerät aus.

    Linux or Unix

    ```
    sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
    ```

    Windows Command Prompt (CMD)

    ```
    type C:\greengrass\v2\\logs\\com.example.HelloWorld.log
    ```

    PowerShell

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

    Sie sollten Meldungen ähnlich dem folgenden Beispiel sehen.

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

> **Note**
>
> Wenn sich die Protokollmeldungen nicht ändern, ist die Bereitstellung fehlgeschlagen oder hat das Core-Gerät nicht erreicht. Dies kann passieren, wenn Ihr Core-Gerät nicht mit dem Internet verbunden ist oder nicht über die Berechtigung zum Abrufen von Artefakten aus Ihrem S3-Bucket verfügt. Führen Sie den folgenden Befehl auf Ihrem Core-Gerät aus, um die AWS IoT Greengrass Core-Softwareprotokolldatei anzuzeigen. Diese Datei enthält Protokolle aus dem Bereitstellungsservice des Greengrass-Core-Geräts.
>
> Linux or Unix
>
> ```
> sudo tail -f /greengrass/v2/logs/greengrass.log
> ```
>
> Windows Command Prompt (CMD)
>
> ```
> type C:\greengrass\v2\\logs\\greengrass.log
> ```
>
> Der `type` Befehl schreibt den Inhalt der Datei in das Terminal. Führen Sie diesen Befehl mehrmals aus, um Änderungen in der Datei zu beobachten.
>
> PowerShell
>
> ```
> gc C:\greengrass\v2\\logs\\greengrass.log -Tail 10 -Wait
> ```
>
> Weitere Informationen finden Sie unter Problembehebung AWS IoT Greengrass V2.

Bereitstellen Ihrer Komponente (AWS CLI)

So stellen Sie Ihre Hello-World-Komponente bereit

1. Erstellen Sie auf Ihrem Entwicklungscomputer eine Datei mit dem Namen `hello-world-deployment.json` und kopieren Sie den folgenden JSON-Code in die Datei . Diese Datei definiert die Komponenten und Konfigurationen, die bereitgestellt werden sollen.

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

Diese Konfigurationsdatei gibt an, dass die Version `1.0.0` der Hello-World-Komponente bereitgestellt werden soll, die Sie im vorherigen Verfahren entwickelt und veröffentlicht haben. `configurationUpdate` gibt an, dass die Komponentenkonfiguration in einer JSON-kodierten Zeichenfolge zusammengeführt werden soll. Dieses Konfigurationsupdate legt den `Message` Parameter Hello World `universe` für das Gerät in dieser Bereitstellung auf fest.

2. Führen Sie den folgenden Befehl aus, um die Komponente auf Ihrem Greengrass-Kerngerät bereitzustellen. Sie können auf Objekten bereitstellen, bei denen es sich um einzelne Geräte handelt, oder auf Objektgruppen, bei denen es sich um Gerätegruppen handelt. Ersetzen Sie *`MyGreengrassCore`* durch den Namen des AWS IoT Objekts für Ihr Core-Gerät.

   Linux or Unix

   ```
   aws greengrassv2 create-deployment \
     --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" \
     --cli-input-json file://hello-world-deployment.json
   ```

   Windows Command Prompt (CMD)

   ```
   aws greengrassv2 create-deployment ^
     --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" ^
     --cli-input-json file://hello-world-deployment.json
   ```

   PowerShell

   ```
   aws greengrassv2 create-deployment `
     --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" `
     --cli-input-json file://hello-world-deployment.json
   ```

Der Befehl gibt eine Antwort ähnlich dem folgenden Beispiel aus.

```
{
  "deploymentId": "deb69c37-314a-4369-a6a1-3dff9fce73a9",
  "iotJobId": "b5d92151-6348-4941-8603-bdbfb3e02b75",
  "iotJobArn": "arn:aws:iot:region:account-id:job/b5d92151-6348-4941-8603-
bdbfb3e02b75"
}
```

3. Überprüfen Sie, ob die Bereitstellung erfolgreich abgeschlossen wurde. Es kann einige Minuten dauern, bis die Bereitstellung abgeschlossen ist. Überprüfen Sie das Hello World-Protokoll, um die Änderung zu überprüfen. Führen Sie den folgenden Befehl auf Ihrem Greengrass-Core-Gerät aus.

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
   ```

   Windows Command Prompt (CMD)

   ```
   type C:\greengrass\v2\\logs\\com.example.HelloWorld.log
   ```

   PowerShell

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

   Sie sollten Meldungen ähnlich dem folgenden Beispiel sehen.

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

   > **Note**
   >
   > Wenn sich die Protokollmeldungen nicht ändern, ist die Bereitstellung fehlgeschlagen oder hat das Core-Gerät nicht erreicht. Dies kann passieren, wenn Ihr Core-Gerät nicht mit dem Internet verbunden ist oder nicht über die Berechtigung zum Abrufen von Artefakten aus Ihrem S3-Bucket verfügt. Führen Sie den folgenden Befehl auf Ihrem Core-Gerät aus, um die AWS IoT Greengrass Core-Softwareprotokolldatei anzuzeigen.

Diese Datei enthält Protokolle aus dem Bereitstellungsservice des Greengrass-Core-Geräts.

Linux or Unix

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

Windows Command Prompt (CMD)

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

Der `type` Befehl schreibt den Inhalt der Datei in das Terminal. Führen Sie diesen Befehl mehrmals aus, um Änderungen in der Datei zu beobachten.

PowerShell

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

Weitere Informationen finden Sie unter Problembehebung AWS IoT Greengrass V2.

Nächste Schritte

Sie haben dieses Tutorial abgeschlossen. DieAWS IoT Greengrass Core-Software und Ihre Hello World-Komponente werden auf Ihrem Gerät ausgeführt. Außerdem ist Ihre Hello World-Komponente imAWS IoT Greengrass Cloud-Dienst verfügbar, um sie auf anderen Geräten bereitzustellen. Weitere Informationen zu den Themen, die in diesem Tutorial ausführen, finden Sie unter:

- Erstellen von AWS IoT Greengrass Komponenten
- Veröffentlichen Sie Komponenten zur Bereitstellung auf Ihren Kerngeräten
- Bereitstellen von AWS IoT Greengrass Komponenten auf Geräten

Einrichtung von AWS IoT Greengrass Kerngeräten

Führen Sie die Aufgaben in diesem Abschnitt aus, um die AWS IoT Greengrass Core-Software zu installieren, zu konfigurieren und auszuführen.

> **Note**
>
> In diesem Abschnitt wird die erweiterte Installation und Konfiguration der AWS IoT Greengrass Core-Software beschrieben. Wenn Sie zum ersten Mal Benutzer von sind AWS IoT Greengrass V2, empfehlen wir Ihnen, zunächst das Tutorial „Erste Schritte“ zu absolvieren, um ein Kerngerät einzurichten und die Funktionen von AWS IoT Greengrass zu erkunden.

Unterstützte Plattformen und Anforderungen

Bevor Sie beginnen, stellen Sie sicher, dass Sie die folgenden Voraussetzungen für die Installation und Ausführung der AWS IoT Greengrass Core-Software erfüllen.

> **Tip**
>
> Sie können im AWS Partner-Gerätekatalog nach Geräten suchen, die dafür AWS IoT Greengrass V2 in Frage kommen.

Themen

Unterstützte Plattformen

AWS IoT Greengrass unterstützt offiziell Geräte, auf denen die folgenden Plattformen ausgeführt werden. Geräte mit Plattformen, die nicht in dieser Liste aufgeführt sind, funktionieren

möglicherweise, AWS IoT Greengrass Tests werden jedoch nur auf diesen angegebenen Plattformen durchgeführt.

Linux

Architekturen:

- Armv7l
- Armv8 (AArch64)
- x86_64

Windows

Architekturen:

- x86_64

Versionen:

- Windows 10
- Windows 11
- Windows Server 2019
- Windows Server 2022

> **Note**
>
> Einige AWS IoT Greengrass Funktionen werden derzeit auf Windows-Geräten nicht unterstützt. Weitere Informationen finden Sie unter Greengrass-Funktionskompatibilität nach Betriebssystem und Überlegungen zu Funktionen für Windows-Geräte.

Linux-Plattformen können auch AWS IoT Greengrass V2 in einem Docker-Container ausgeführt werden. Weitere Informationen finden Sie unter Ausführen von AWS IoT Greengrass Core-Software in einem Docker-Container.

Um ein benutzerdefiniertes Linux-basiertes Betriebssystem zu erstellen, können Sie das BitBake Rezept für AWS IoT Greengrass V2 im `meta-aws` Projekt verwenden. Das `meta-aws` Projekt bietet

Rezepte, mit denen Sie AWS Edge-Softwarefunktionen in eingebetteten Linux-Systemen erstellen können, die mit OpenEmbeddedBuild-Frameworks von Yocto Project erstellt wurden. Das Yocto-Projekt ist ein Open-Source-Kooperationsprojekt, das Ihnen hilft, maßgeschneiderte Linux-basierte Systeme für eingebettete Anwendungen unabhängig von der Hardwarearchitektur zu entwickeln. Das BitBake Rezept für AWS IoT Greengrass V2 installiert, konfiguriert und führt die AWS IoT Greengrass Core-Software automatisch auf Ihrem Gerät aus.

Anforderungen an Speichergeräte

Geräte müssen die folgenden Anforderungen erfüllen, um die AWS IoT Greengrass Core-Software v2.x installieren und ausführen zu können.

> **Note**
>
> Mithilfe AWS IoT Device Tester von können Sie überprüfen AWS IoT Greengrass , ob Ihr Gerät die AWS IoT Greengrass Core-Software ausführen und mit dem kommunizieren kann. AWS Cloud Weitere Informationen finden Sie unter AWS IoT Device Tester Für AWS IoT Greengrass V2 verwenden.

Linux

- Die Verwendung eines AWS-Region, der unterstützt AWS IoT Greengrass V2. Eine vollständige Liste der unterstützten Regionen finden Sie unter AWS IoT Greengrass V2 -Endpunkte und -Kontingente in Allgemeine AWS-Referenz.
- Für die AWS IoT Greengrass Core-Software stehen mindestens 256 MB Festplattenspeicher zur Verfügung. Diese Anforderung umfasst keine Komponenten, die auf dem Kerngerät bereitgestellt werden.
- Der AWS IoT Greengrass Core-Software sind mindestens 96 MB RAM zugewiesen. Diese Anforderung umfasst keine Komponenten, die auf dem Kerngerät ausgeführt werden. Weitere Informationen finden Sie unter Steuern Sie die Speicherzuweisung mit JVM-Optionen.
- Java Runtime Environment (JRE) Version 8 oder höher. Java muss in der Umgebungsvariablen PATH auf dem Gerät verfügbar sein. Um Java zur Entwicklung benutzerdefinierter Komponenten zu verwenden, müssen Sie ein Java Development Kit (JDK) installieren. Wir empfehlen, dass Sie die Langzeit-Support-Versionen von Amazon Corretto oder OpenJDK verwenden. Version 8 oder höher ist erforderlich.
- GNU C-Bibliothek (Glibc) Version 2.25 oder höher.

- Sie müssen die AWS IoT Greengrass Core-Software als Root-Benutzer ausführen. Verwenden `sudo` Sie zum Beispiel.
- Der Root-Benutzer, der die AWS IoT Greengrass Core-Software ausführt`root`, muss beispielsweise über die Berechtigung verfügen, `sudo` mit jedem Benutzer und jeder Gruppe zu arbeiten. Die `/etc/sudoers` Datei muss diesem Benutzer die Berechtigung geben, zusammen mit anderen Gruppen ausgeführt `sudo` zu werden. Die Zugriffsrechte für den Benutzer `/etc/sudoers` sollten wie im folgenden Beispiel aussehen.

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

- Das Kerngerät muss in der Lage sein, ausgehende Anfragen an eine Reihe von Endpunkten und Ports auszuführen. Weitere Informationen finden Sie unter Zulassen von Gerätedatenverkehr über einen Proxy oder eine Firewall.
- Das `/tmp` Verzeichnis muss mit `exec` entsprechenden Berechtigungen bereitgestellt werden.
- Alle folgenden Shell-Befehle:
  - `ps -ax -o pid,ppid`
  - `sudo`
  - `sh`
  - `kill`
  - `cp`
  - `chmod`
  - `rm`
  - `ln`
  - `echo`
  - `exit`
  - `id`
  - `uname`
  - `grep`
- Ihr Gerät benötigt möglicherweise auch die folgenden optionalen Shell-Befehle:
  - (Optional) `systemctl`. Dieser Befehl wird verwendet, um die AWS IoT Greengrass Core-Software als Systemdienst einzurichten.
  - (Optional) `useraddgroupadd`, un`dusermod`. Diese Befehle werden verwendet, um den `ggc_user` Systembenutzer und die `ggc_group` Systemgruppe einzurichten.

  - (Optional) `mkfifo`. Dieser Befehl wird verwendet, um Lambda-Funktionen als Komponenten auszuführen.
- Um Systemressourcenlimits für Komponentenprozesse zu konfigurieren, muss auf Ihrem Gerät die Linux-Kernel-Version 2.6.24 oder höher ausgeführt werden.
- Um Lambda-Funktionen ausführen zu können, muss Ihr Gerät zusätzliche Anforderungen erfüllen. Weitere Informationen finden Sie unter Anforderungen an die Lambda-Funktion.

Windows

- Die Verwendung eines AWS-Region, der unterstützt AWS IoT Greengrass V2. Eine vollständige Liste der unterstützten Regionen finden Sie unter AWS IoT Greengrass V2 -Endpunkte und -Kontingente in Allgemeine AWS-Referenz.
- Für die AWS IoT Greengrass Core-Software stehen mindestens 256 MB Festplattenspeicher zur Verfügung. Diese Anforderung umfasst keine Komponenten, die auf dem Kerngerät bereitgestellt werden.
- Der AWS IoT Greengrass Core-Software sind mindestens 160 MB RAM zugewiesen. Diese Anforderung umfasst keine Komponenten, die auf dem Kerngerät ausgeführt werden. Weitere Informationen finden Sie unter Steuern Sie die Speicherzuweisung mit JVM-Optionen.
- Java Runtime Environment (JRE) Version 8 oder höher. Java muss in der Systemvariablen PATH auf dem Gerät verfügbar sein. Um Java zur Entwicklung benutzerdefinierter Komponenten zu verwenden, müssen Sie ein Java Development Kit (JDK) installieren. Wir empfehlen, dass Sie die Langzeit-Support-Versionen von Amazon Corretto oder OpenJDK verwenden. Version 8 oder höher ist erforderlich..

  > **Note**
  >
  > Um Version 2.5.0 von Greengrass Nucleus verwenden zu können, müssen Sie eine 64-Bit-Version von Java Runtime Environment (JRE) verwenden. Greengrass Nucleus Version 2.5.1 unterstützt 32-Bit- und 64-Bit-JREs.

- Der Benutzer, der die AWS IoT Greengrass Core-Software installiert, muss Administrator sein.
- Sie müssen die AWS IoT Greengrass Core-Software als Systemdienst installieren. Geben Sie an`--setup-system-service true`, wann Sie die Software installieren.
- Jeder Benutzer, der Komponentenprozesse ausführt, muss in dem LocalSystem Konto vorhanden sein, und der Name und das Passwort des Benutzers müssen sich in der Credential

Manager-Instanz für das LocalSystem Konto befinden. Sie können diesen Benutzer einrichten, wenn Sie den Anweisungen zur Installation der AWS IoT Greengrass Core-Software folgen.

- Das Core-Gerät muss in der Lage sein, ausgehende Anfragen an eine Reihe von Endpunkten und Ports auszuführen. Weitere Informationen finden Sie unter Zulassen von Gerätedatenverkehr über einen Proxy oder eine Firewall.

Anforderungen an die Lambda-Funktion

Ihr Gerät muss die folgenden Anforderungen erfüllen, um Lambda-Funktionen ausführen zu können:

- Ein Linux-basiertes Betriebssystem.
- Ihr Gerät muss über den `mkfifo` Shell-Befehl verfügen.
- Ihr Gerät muss die Programmiersprachenbibliotheken ausführen, die für eine Lambda-Funktion erforderlich sind. Sie müssen die erforderlichen Bibliotheken auf dem Gerät installieren und sie der `PATH` Umgebungsvariablen hinzufügen. Greengrass unterstützt alle von Lambda unterstützten Versionen von Python-, Node.js- und Java-Runtimes. Greengrass wendet keine zusätzlichen Einschränkungen auf veraltete Lambda-Laufzeitversionen an. Weitere Informationen zur AWS IoT Greengrass Unterstützung von Lambda-Laufzeiten finden Sie unter. Ausführen von -AWS LambdaFunktionen
- Um containerisierte Lambda-Funktionen ausführen zu können, muss Ihr Gerät die folgenden Anforderungen erfüllen:
  - Linux-Kernel-Version 4.4 oder höher.
  - Der Kernel muss cgroups v1 unterstützen, und Sie müssen die folgenden Cgroups aktivieren und mounten:
    - Die Speicher-Cgroup für AWS IoT Greengrass die Festlegung des Speicherlimits für containerisierte Lambda-Funktionen.
    - Die Gerätegruppe für containerisierte Lambda-Funktionen für den Zugriff auf Systemgeräte oder Volumes.

    Die AWS IoT Greengrass Core-Software unterstützt cgroups v2 nicht.

    Um diese Anforderung zu erfüllen, starten Sie das Gerät mit den folgenden Linux-Kernelparametern.

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

> **Tip**
>
> Bearbeiten Sie auf einem Raspberry Pi die `/boot/cmdline.txt` Datei, um die Kernel-Parameter des Geräts festzulegen.

- Sie müssen die folgenden Linux-Kernelkonfigurationen auf dem Gerät aktivieren:
  - Namespace
    - `CONFIG_IPC_NS`
    - `CONFIG_UTS_NS`
    - `CONFIG_USER_NS`
    - `CONFIG_PID_NS`
  - Cgroups:
    - `CONFIG_CGROUP_DEVICE`
    - `CONFIG_CGROUPS`
    - `CONFIG_MEMCG`
  - Weitere:
    - `CONFIG_POSIX_MQUEUE`
    - `CONFIG_OVERLAY_FS`
    - `CONFIG_HAVE_ARCH_SECCOMP_FILTER`
    - `CONFIG_SECCOMP_FILTER`
    - `CONFIG_KEYS`
    - `CONFIG_SECCOMP`
    - `CONFIG_SHMEM`

> **Tip**
>
> In der Dokumentation zu Ihrer Linux-Distribution erfahren Sie, wie Sie die Linux-Kernelparameter überprüfen und festlegen. Sie können auch AWS IoT Device Tester für verwenden AWS IoT Greengrass , um zu überprüfen, ob Ihr Gerät diese Anforderungen erfüllt. Weitere Informationen finden Sie unter AWS IoT Device Tester Für AWS IoT Greengrass V2 verwenden.

Überlegungen zu Funktionen für Windows-Geräte

Einige AWS IoT Greengrass Funktionen werden derzeit auf Windows-Geräten nicht unterstützt. Überprüfen Sie die Funktionsunterschiede, um zu überprüfen, ob ein Windows-Gerät Ihren Anforderungen entspricht. Weitere Informationen finden Sie unter Greengrass-Funktionskompatibilität nach Betriebssystem.

Richten Sie eine 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.

Wählen Sie zum Erstellen eines Administratorbenutzers eine der folgenden Optionen aus.

Wählen Sie eine Möglichkeit zur Verwaltung Ihres Administrators aus.	Bis	Von	Sie können auch
Im IAM Identity Center (Empfohlen)	Verwendung von kurzfristigen Anmeldeinformationen für den Zugriff auf AWS. Dies steht im Einklang mit den bewährten Methoden für die Sicherheit. Weitere Informationen zu bewährten Methoden finden Sie unter Bewährte Methoden für die Sicherheit in IAM im IAM-Benutzerhandbuch.	Beachtung der Anweisungen unter Erste Schritte im AWS IAM Identity Center - Benutzerhandbuch.	Konfigurieren Sie den programmatischen Zugriff, indem Sie den AWS IAM Identity Center im AWS Command Line Interface Benutzerhandbuch AWS CLI zu verwendenden konfigurieren.
In IAM (Nicht empfohlen)	Verwendung von langfristigen Anmeldeinformationen für den Zugriff auf AWS.	Beachtung der Anweisungen unter Erstellen Ihres ersten IAM-Administratorbenutzers und Ihrer ersten Benutzergruppe im IAM-Benutzerhandbuch.	Programmgesteuerten Zugriff unter Verwendung der Informationen unter Verwalten der Zugriffsschlüssel für IAM-Benutzer im IAM-Benutzerhandbuch konfigurieren.

Installieren Sie die AWS IoT Greengrass Core-Software.

AWS IoT Greengrass erweitert AWS auf Edge-Geräte, sodass sie auf die von ihnen generierten Daten reagieren können, während sie die AWS Cloud für Verwaltung, Analysen und dauerhafte Speicherung verwenden. Installieren Sie die -AWS IoT GreengrassCore-Software auf Edge-Geräten, um sie in AWS IoT Greengrass und zu integrierenAWS Cloud.

> **Important**
>
> Bevor Sie die AWS IoT Greengrass Core-Software herunterladen und installieren, überprüfen Sie, ob Ihr Core-Gerät die Anforderungen für die Installation und Ausführung der AWS IoT Greengrass Core-Software v2.0 erfüllt.

Die -AWS IoT GreengrassCore-Software enthält ein Installationsprogramm, das Ihr Gerät als Greengrass-Core-Gerät einrichtet. Wenn Sie das Installationsprogramm ausführen, können Sie Optionen wie den Stammordner und die AWS-Region zu verwendende konfigurieren. Sie können festlegen, dass das Installationsprogramm die erforderlichen AWS IoT und IAM-Ressourcen für Sie erstellt. Sie können auch lokale Entwicklungstools bereitstellen, um ein Gerät zu konfigurieren, das Sie für die Entwicklung benutzerdefinierter Komponenten verwenden.

Die -AWS IoT GreengrassCore-Software benötigt die folgenden AWS IoT und IAM-Ressourcen, um eine Verbindung mit dem herzustellen AWS Cloud und zu arbeiten:

- Ein AWS IoT-Objekt. Wenn Sie ein Gerät als -AWS IoTObjekt registrieren, kann dieses Gerät ein digitales Zertifikat verwenden, um sich bei zu authentifizierenAWS. Dieses Zertifikat ermöglicht dem Gerät die Kommunikation mit AWS IoT und AWS IoT Greengrass. Weitere Informationen finden Sie unter Geräteauthentifizierung und -autorisierung für AWS IoT Greengrass.
- (Optional) Eine AWS IoT-Objektgruppe. Sie verwenden Objektgruppen, um Flotten von Greengrass-Core-Geräten zu verwalten. Wenn Sie Softwarekomponenten auf Ihren Geräten bereitstellen, können Sie wählen, ob Sie auf einzelnen Geräten oder auf Gerätegruppen bereitstellen möchten. Sie können ein Gerät zu einer Objektgruppe hinzufügen, um die Softwarekomponenten dieser Objektgruppe auf dem Gerät bereitzustellen. Weitere Informationen finden Sie unter Bereitstellen von AWS IoT Greengrass Komponenten auf Geräten.
- Eine IAM-Rolle. Greengrass-Core-Geräte verwenden den AWS IoT Core Anmeldeinformationsanbieter, um Aufrufe von -AWSServices mit einer IAM-Rolle zu autorisieren. Diese Rolle ermöglicht es Ihrem GerätAWS IoT, mit zu interagieren, Protokolle an Amazon

CloudWatch Logs zu senden und benutzerdefinierte Komponentenartefakte von Amazon Simple Storage Service (Amazon S3) herunterzuladen. Weitere Informationen finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

- Ein -AWS IoTRollenalias. Greengrass-Core-Geräte verwenden den Rollenalias, um die zu verwendende IAM-Rolle zu identifizieren. Mit dem Rollenalias können Sie die IAM-Rolle ändern, die Gerätekonfiguration jedoch beibehalten. Weitere Informationen finden Sie unter Autorisieren von direkten Aufrufen an -AWSServices im AWS IoT Core -Entwicklerhandbuch.

Wählen Sie eine der folgenden Optionen, um die AWS IoT Greengrass Core-Software auf Ihrem Core-Gerät zu installieren.

- Schnellinstallation

  Wählen Sie diese Option, um ein Greengrass-Core-Gerät in so wenigen Schritten wie möglich einzurichten. Das Installationsprogramm erstellt die erforderlichen - AWS IoT und IAM-Ressourcen für Sie. Für diese Option müssen Sie dem Installationsprogramm AWS Anmeldeinformationen bereitstellen, um Ressourcen in Ihrem zu erstellenAWS-Konto.

  Sie können diese Option nicht verwenden, um hinter einer Firewall oder einem Netzwerk-Proxy zu installieren. Wenn sich Ihre Geräte hinter einer Firewall oder einem Netzwerk-Proxy befinden, sollten Sie eine manuelle Installation in Betracht ziehen.

  Weitere Informationen finden Sie unter Installieren von AWS IoT Greengrass Core-Software mit automatischer Ressourcenbereitstellung.

- Manuelle Installation

  Wählen Sie diese Option, um die erforderlichen AWS Ressourcen manuell zu erstellen oder hinter einer Firewall oder einem Netzwerk-Proxy zu installieren. Durch die Verwendung einer manuellen Installation müssen Sie dem Installationsprogramm keine Berechtigung zum Erstellen von Ressourcen in Ihrem erteilenAWS-Konto, da Sie die erforderlichen - AWS IoT und IAM-Ressourcen erstellen. Sie können Ihr Gerät auch so konfigurieren, dass eine Verbindung über Port 443 oder über einen Netzwerk-Proxy hergestellt wird. Sie können die -AWS IoT GreengrassCore-Software auch so konfigurieren, dass sie einen privaten Schlüssel und ein Zertifikat verwendet, die Sie in einem Hardwaresicherheitsmodul (HSM), einem Trusted Platform Module (TPM) oder einem anderen kryptografischen Element speichern.

  Weitere Informationen finden Sie unter Installieren Sie die AWS IoT Greengrass Core-Software mit manueller Ressourcenbereitstellung.

- Installation mit AWS IoT Flottenbereitstellung

  Wählen Sie diese Option, um die erforderlichen AWS Ressourcen aus einer AWS IoT Flottenbereitstellungsvorlage zu erstellen. Sie können diese Option wählen, um ähnliche Geräte in einer Flotte zu erstellen, oder wenn Sie Geräte herstellen, die Ihre Kunden später aktivieren, z. B. Fahrzeuge oder Smart-Home-Geräte. Geräte verwenden Anspruchszertifikate, um AWS Ressourcen zu authentifizieren und bereitzustellen, einschließlich eines X.509-Clientzertifikats, das das Gerät verwendet, um eine Verbindung mit dem AWS Cloud für den normalen Betrieb herzustellen. Sie können die Anspruchszertifikate während der Herstellung in die Hardware des Geräts einbetten oder Flashen, und Sie können dasselbe Anspruchszertifikat und denselben Schlüssel verwenden, um mehrere Geräte bereitzustellen. Sie können Geräte auch so konfigurieren, dass eine Verbindung über Port 443 oder über einen Netzwerk-Proxy hergestellt wird.

  Weitere Informationen finden Sie unter Installieren Sie die AWS IoT Greengrass Core-Software mit AWS IoT Flottenbereitstellung.

- Installation mit benutzerdefinierter Bereitstellung

  Wählen Sie diese Option, um eine benutzerdefinierte Java-Anwendung zu entwickeln, die die erforderlichen AWS Ressourcen bereitstellt. Sie können diese Option wählen, wenn Sie Ihre eigenen X.509-Clientzertifikate erstellen oder mehr Kontrolle über den Bereitstellungsprozess wünschen. AWS IoT Greengrass bietet eine Schnittstelle, die Sie implementieren können, um Informationen zwischen Ihrer benutzerdefinierten Bereitstellungsanwendung und dem AWS IoT Greengrass Core-Softwareinstallationsprogramm auszutauschen.

  Weitere Informationen finden Sie unter Installieren Sie die AWS IoT Greengrass Core-Software mit benutzerdefinierter Ressourcenbereitstellung.

AWS IoT Greengrass bietet auch containerisierte Umgebungen, in denen die AWS IoT Greengrass Core-Software ausgeführt wird. Sie können eine Docker-Datei verwenden, um AWS IoT Greengrass in einem Docker-Container auszuführen.

Themen

- Installieren von AWS IoT Greengrass Core-Software mit automatischer Ressourcenbereitstellung
- Installieren Sie die AWS IoT Greengrass Core-Software mit manueller Ressourcenbereitstellung
- Installieren Sie die AWS IoT Greengrass Core-Software mit AWS IoT Flottenbereitstellung

- Installieren Sie die AWS IoT Greengrass Core-Software mit benutzerdefinierter Ressourcenbereitstellung
- Installer-Argumente

Installieren von AWS IoT Greengrass Core-Software mit automatischer Ressourcenbereitstellung

Die -AWS IoT GreengrassCore-Software enthält ein Installationsprogramm, das Ihr Gerät als Greengrass-Core-Gerät einrichtet. Um ein Gerät schnell einzurichten, kann das Installationsprogramm das AWS IoT Objekt, die AWS IoT Objektgruppe, die IAM-Rolle und den AWS IoT Rollenalias bereitstellen, den das Core-Gerät für den Betrieb benötigt. Das Installationsprogramm kann auch die lokalen Entwicklungstools auf dem Core-Gerät bereitstellen, sodass Sie das Gerät verwenden können, um benutzerdefinierte Softwarekomponenten zu entwickeln und zu testen. Das Installationsprogramm benötigt AWS Anmeldeinformationen, um diese Ressourcen bereitzustellen und die Bereitstellung zu erstellen.

Wenn Sie dem Gerät keine AWS Anmeldeinformationen bereitstellen können, können Sie die AWS Ressourcen bereitstellen, die das Core-Gerät für den Betrieb benötigt. Sie können die Entwicklungstools auch auf einem Core-Gerät bereitstellen, das als Entwicklungsgerät verwendet werden soll. Auf diese Weise können Sie dem Gerät weniger Berechtigungen erteilen, wenn Sie das Installationsprogramm ausführen. Weitere Informationen finden Sie unter Installieren Sie die AWS IoT Greengrass Core-Software mit manueller Ressourcenbereitstellung.

> **Important**
>
> Bevor Sie die AWS IoT Greengrass Core-Software herunterladen, überprüfen Sie, ob Ihr Core-Gerät die Anforderungen für die Installation und Ausführung der AWS IoT Greengrass Core-Software v2.0 erfüllt.

Themen

- Einrichten der Geräteumgebung
- Bereitstellen von AWS Anmeldeinformationen für das Gerät
- Herunterladen der AWS IoT Greengrass Core-Software
- Installieren Sie die AWS IoT Greengrass Core-Software.

Einrichten der Geräteumgebung

Führen Sie die Schritte in diesem Abschnitt aus, um ein Linux- oder Windows-Gerät einzurichten, das als AWS IoT Greengrass Core-Gerät verwendet werden soll.

Einrichten eines Linux-Geräts

So richten Sie ein Linux-Gerät für ein AWS IoT Greengrass V2

1. Installieren Sie die Java-Laufzeit, die die -AWS IoT GreengrassCore-Software zum Ausführen benötigt. Wir empfehlen Ihnen, Amazon Corretto- oder OpenJDK-Langzeit-Supportversionen zu verwenden. Version 8 oder höher ist erforderlich. Die folgenden Befehle zeigen Ihnen, wie Sie OpenJDK auf Ihrem Gerät installieren.

   - Für Debian- oder Ubuntu-basierte Distributionen:

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

   - Für Red Hat-basierte Distributionen:

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

   - Für Amazon Linux 2:

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

   - Für Amazon Linux 2023:

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

   Wenn die Installation abgeschlossen ist, führen Sie den folgenden Befehl aus, um zu überprüfen, ob Java auf Ihrem Linux-Gerät ausgeführt wird.

   ```
   java -version
   ```

   Der Befehl gibt die Version von Java aus, die auf dem Gerät ausgeführt wird. Bei einer Debian-basierten Verteilung könnte die Ausgabe beispielsweise dem folgenden Beispiel ähneln.

   ```
   openjdk version "11.0.9.1" 2020-11-04
   ```

```
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
```

2. (Optional) Erstellen Sie den Standardsystembenutzer und die Standardgruppe, die Komponenten auf dem Gerät ausführt. Sie können sich auch dafür entscheiden, dass das AWS IoT Greengrass-Core-Software-Installationsprogramm diesen Benutzer und diese Gruppe während der Installation mit dem `--component-default-user`Installationsprogramm-Argument erstellt. Weitere Informationen finden Sie unter Installer-Argumente.

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

3. Stellen Sie sicher, dass der Benutzer, der die AWS IoT Greengrass Core-Software ausführt (normalerweise `root`), über die Berechtigung verfügt, `sudo` mit jedem Benutzer und jeder Gruppe auszuführen.

   a. Führen Sie den folgenden Befehl aus, um die `/etc/sudoers` Datei zu öffnen.

      ```
      sudo visudo
      ```

   b. Stellen Sie sicher, dass die Berechtigung für den Benutzer wie im folgenden Beispiel aussieht.

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

4. (Optional) Um containerisierte Lambda-Funktionen auszuführen, müssen Sie cgroups v1 aktivieren und die Cgroups für Speicher und Geräte aktivieren und mounten. Wenn Sie keine containerisierten Lambda-Funktionen ausführen möchten, können Sie diesen Schritt überspringen.

   Um diese Cgroups-Optionen zu aktivieren, starten Sie das Gerät mit den folgenden Linux-Kernelparametern.

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

   Informationen zum Anzeigen und Festlegen von Kernelparametern für Ihr Gerät finden Sie in der Dokumentation für Ihr Betriebssystem und Ihren Bootloader. Folgen Sie den Anweisungen, um die Kernel-Parameter dauerhaft festzulegen.

5. Installieren Sie alle anderen erforderlichen Abhängigkeiten auf Ihrem Gerät, wie in der Liste der Anforderungen unter angegebenAnforderungen an Speichergeräte.

Einrichten eines Windows-Geräts

> **Note**
>
> Diese Funktion ist für v2.5.0 und höher der Greengrass-Kernkomponente verfügbar.

So richten Sie ein Windows-Gerät für ein AWS IoT Greengrass V2

1. Installieren Sie die Java-Laufzeit, die die -AWS IoT GreengrassCore-Software zum Ausführen benötigt. Wir empfehlen Ihnen, Amazon Corretto- oder OpenJDK-Langzeit-Supportversionen zu verwenden. Version 8 oder höher ist erforderlich.
2. Überprüfen Sie, ob Java für die PATH-Systemvariable verfügbar ist, und fügen Sie es hinzu, falls nicht. Das LocalSystem Konto führt die AWS IoT Greengrass Core-Software aus, daher müssen Sie Java zur PATH-Systemvariablen anstelle der PATH-Benutzervariablen für Ihren Benutzer hinzufügen. Gehen Sie wie folgt vor:
   a. Drücken Sie die Windows-Taste, um das Startmenü zu öffnen.
   b. Geben Sie ein**`environment variables`**, um im Startmenü nach den Systemoptionen zu suchen.
   c. Wählen Sie im Startmenü die Option Systemumgebungsvariablen bearbeiten aus, um das Fenster Systemeigenschaften zu öffnen.
   d. Wählen Sie Umgebungsvariablen..., um das Fenster Umgebungsvariablen zu öffnen.
   e. Wählen Sie unter Systemvariablen die Option Pfad und dann Bearbeiten aus. Im Fenster Umgebungsvariable bearbeiten können Sie jeden Pfad in einer separaten Zeile anzeigen.
   f. Überprüfen Sie, ob der Pfad zum `bin` Ordner der Java-Installation vorhanden ist. Der Pfad könnte dem folgenden Beispiel ähneln.

      ```
      C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
      ```

   g. Wenn der `bin` Ordner der Java-Installation in Pfad fehlt, wählen Sie Neu, um ihn hinzuzufügen, und wählen Sie dann OK aus.
3. Öffnen Sie die Windows-Eingabeaufforderung (`cmd.exe`) als Administrator.

4. Erstellen Sie den Standardbenutzer im LocalSystem Konto auf dem Windows-Gerät. Ersetzen Sie *das Passwort* durch ein sicheres Passwort.

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

> **Tip**
>
> Abhängig von Ihrer Windows-Konfiguration kann das Passwort des Benutzers so eingestellt sein, dass es an einem Datum in der Zukunft abläuft. Um sicherzustellen, dass Ihre Greengrass-Anwendungen weiterhin funktionieren, verfolgen Sie, wann das Passwort abläuft, und aktualisieren Sie es, bevor es abläuft. Sie können auch festlegen, dass das Passwort des Benutzers nie abläuft.
>
> - Führen Sie den folgenden Befehl aus, um zu überprüfen, wann ein Benutzer und sein Passwort ablaufen.
>
>   ```
>   net user ggc_user | findstr /C:expires
>   ```
>
> - Führen Sie den folgenden Befehl aus, um das Passwort eines Benutzers so festzulegen, dass es nie abläuft.
>
>   ```
>   wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
>   ```
>
> - Wenn Sie Windows 10 oder höher verwenden, wenn der wmic Befehl veraltet ist, führen Sie den folgenden PowerShell Befehl aus.
>
>   ```
>   Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =
>    'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
>   ```

5. Laden Sie das PsExec Dienstprogramm von Microsoft herunter und installieren Sie es auf dem Gerät.
6. Verwenden Sie das PsExec Dienstprogramm , um den Benutzernamen und das Passwort für den Standardbenutzer in der Credential Manager-Instance für das LocalSystem Konto zu speichern. Ersetzen Sie *password* durch das Passwort des Benutzers, das Sie zuvor festgelegt haben.

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

Wenn sich der PsExec License Agreement öffnet, wählen Sie ausAccept, ob Sie der Lizenz zustimmen möchten, und führen Sie den Befehl aus.

> **Note**
>
> Auf Windows-Geräten führt das LocalSystem Konto den Greengrass-Kern aus, und Sie müssen das PsExec Dienstprogramm verwenden, um die Standardbenutzerinformationen im LocalSystem Konto zu speichern. Die Verwendung der Credential Manager-Anwendung speichert diese Informationen im Windows-Konto des aktuell angemeldeten Benutzers anstelle des - LocalSystem Kontos.

Bereitstellen von AWS Anmeldeinformationen für das Gerät

Geben Sie Ihre AWS Anmeldeinformationen an Ihr Gerät an, damit das Installationsprogramm die erforderlichen AWS Ressourcen bereitstellen kann. Weitere Informationen zu den erforderlichen Berechtigungen finden Sie unter Minimale IAM-Richtlinie für das Installationsprogramm zur Bereitstellung von Ressourcen.

So stellen Sie dem Gerät AWS Anmeldeinformationen bereit

- Geben Sie Ihre AWS Anmeldeinformationen für das Gerät an, damit das Installationsprogramm die AWS IoT und IAM-Ressourcen für Ihr Core-Gerät bereitstellen kann. Um die Sicherheit zu erhöhen, empfehlen wir Ihnen, temporäre Anmeldeinformationen für eine IAM-Rolle zu erhalten, die nur die Mindestberechtigungen zulässt, die für die Bereitstellung erforderlich sind. Weitere Informationen finden Sie unter Minimale IAM-Richtlinie für das Installationsprogramm zur Bereitstellung von Ressourcen.

  > **Note**
  >
  > Das Installationsprogramm speichert oder speichert Ihre Anmeldeinformationen nicht.

  Führen Sie auf Ihrem Gerät einen der folgenden Schritte aus, um Anmeldeinformationen abzurufen und sie für das Installationsprogramm der -AWS IoT GreengrassCore-Software verfügbar zu machen:

  - (Empfohlen) Verwenden Sie Temporay-Anmeldeinformationen von AWS IAM Identity Center

a. Geben Sie die Zugriffsschlüssel-ID, den geheimen Zugriffsschlüssel und das Sitzungstoken aus dem IAM Identity Center an. Weitere Informationen finden Sie unter Manuelle Aktualisierung von Anmeldeinformationen unter Abrufen und Aktualisieren temporärer Anmeldeinformationen im IAM-Identity-Center-Benutzerhandbuch.

b. Führen Sie die folgenden Befehle aus, um die Anmeldeinformationen für die AWS IoT Greengrass Core-Software bereitzustellen.

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o5OytwEXAMPLE="
```

- Verwenden Sie temporäre Sicherheitsanmeldeinformationen aus einer IAM-Rolle:

a. Geben Sie die Zugriffsschlüssel-ID, den geheimen Zugriffsschlüssel und das Sitzungstoken aus einer von Ihnen übernommenen IAM-Rolle an. Weitere Informationen zum Abrufen dieser Anmeldeinformationen finden Sie unter Anfordern temporärer Sicherheitsanmeldeinformationen im IAM-Benutzerhandbuch.

b. Führen Sie die folgenden Befehle aus, um die Anmeldeinformationen für die AWS IoT Greengrass Core-Software bereitzustellen.

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

```
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o5OytwEXAMPLE="
```

- Verwenden Sie langfristige Anmeldeinformationen von einem IAM-Benutzer:

  a. Geben Sie die Zugriffsschlüssel-ID und den geheimen Zugriffsschlüssel für Ihren IAM-Benutzer an. Sie können einen IAM-Benutzer für die Bereitstellung erstellen, den Sie später löschen. Informationen zur IAM-Richtlinie, die dem Benutzer zugewiesen werden soll, finden Sie unter Minimale IAM-Richtlinie für das Installationsprogramm zur Bereitstellung von Ressourcen. Weitere Informationen zum Abrufen langfristiger Anmeldeinformationen finden Sie unter Verwalten von Zugriffsschlüsseln für IAM-Benutzer im IAM-Benutzerhandbuch.

  b. Führen Sie die folgenden Befehle aus, um die Anmeldeinformationen für die AWS IoT Greengrass Core-Software bereitzustellen.

  Linux or Unix

  ```
  export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
  export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
  ```

  Windows Command Prompt (CMD)

  ```
  set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
  set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
  ```

  PowerShell

  ```
  $env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
  ```

```
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
```

c. (Optional) Wenn Sie einen IAM-Benutzer zur Bereitstellung Ihres Greengrass-Geräts erstellt haben, löschen Sie den Benutzer.

d. (Optional) Wenn Sie die Zugriffsschlüssel-ID und den geheimen Zugriffsschlüssel eines vorhandenen IAM-Benutzers verwendet haben, aktualisieren Sie die Schlüssel für den Benutzer, damit sie nicht mehr gültig sind. Weitere Informationen finden Sie unter Aktualisieren von Zugriffsschlüsseln im AWS Identity and Access Management - Benutzerhandbuch.

Herunterladen der AWS IoT Greengrass Core-Software

Sie können die neueste Version der AWS IoT Greengrass-Core-Software vom folgenden Speicherort herunterladen:

- https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip

> **Note**
>
> Sie können eine bestimmte Version der AWS IoT Greengrass Core-Software vom folgenden Speicherort herunterladen. Ersetzen Sie *Version* durch die Version, die Sie herunterladen möchten.
>
> ```
> https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
> ```

So laden Sie die AWS IoT Greengrass Core-Software herunter

1. Laden Sie die AWS IoT Greengrass Core-Software auf Ihr Core-Gerät in eine Datei mit dem Namen herunter`greengrass-nucleus-latest.zip`.

   Linux or Unix

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
   latest.zip > greengrass-nucleus-latest.zip
   ```

Windows Command Prompt (CMD)

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
latest.zip > greengrass-nucleus-latest.zip
```

PowerShell

```
iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
latest.zip -OutFile greengrass-nucleus-latest.zip
```

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

2. (Optional) So überprüfen Sie die Softwaresignatur des Greengrass-Kerns

> **Note**
>
> Diese Funktion ist mit Greengrass-Kernversion 2.9.5 und höher verfügbar.

a. Verwenden Sie den folgenden Befehl, um die Signatur Ihres Greengrass-Kernartefakts zu überprüfen:

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

Der Dateiname sieht je nach installierter JDK-Version möglicherweise anders aus. Ersetzen Sie durch *jdk17.0.6_10* die installierte JDK-Version.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

Der Dateiname sieht je nach installierter JDK-Version möglicherweise anders aus. Ersetzen Sie durch *jdk17.0.6_10* die installierte JDK-Version.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -
verify -certs -verbose greengrass-nucleus-latest.zip
```

b. Der `jarsigner` Aufruf führt zu einer Ausgabe, die die Ergebnisse der Verifizierung angibt.

   i. Wenn die Greengrass-Kern-ZIP-Datei signiert ist, enthält die Ausgabe die folgende Anweisung:

   ```
   jar verified.
   ```

   ii. Wenn die Greengrass-Kern-ZIP-Datei nicht signiert ist, enthält die Ausgabe die folgende Anweisung:

   ```
   jar is unsigned.
   ```

c. Wenn Sie die Jarsigner-`--certs`Option zusammen mit den `-verbose` Optionen `-verify` und angegeben haben, enthält die Ausgabe auch detaillierte Informationen zum Ausstellerzertifikat.

3. Entpacken Sie die AWS IoT Greengrass Core-Software in einen Ordner auf Ihrem Gerät. Ersetzen Sie durch *GreengrassInstaller* den Ordner, den Sie verwenden möchten.

   Linux or Unix

   ```
   unzip greengrass-nucleus-latest.zip -d GreengrassInstaller && rm greengrass-
   nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir GreengrassInstaller && tar -xf greengrass-nucleus-latest.zip -
   C GreengrassInstaller && del greengrass-nucleus-latest.zip
   ```

PowerShell

```
Expand-Archive -Path greengrass-nucleus-latest.zip -DestinationPath .\
\GreengrassInstaller
rm greengrass-nucleus-latest.zip
```

4. (Optional) Führen Sie den folgenden Befehl aus, um die Version der AWS IoT Greengrass Core-Software anzuzeigen.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

> **Important**
>
> Wenn Sie eine Version des Greengrass-Kerns vor v2.4.0 installieren, entfernen Sie diesen Ordner nicht, nachdem Sie die AWS IoT Greengrass Core-Software installiert haben.
> Die AWS IoT Greengrass -Core-Software verwendet die Dateien in diesem Ordner, um auszuführen.
> Wenn Sie die neueste Version der Software heruntergeladen haben, installieren Sie v2.4.0 oder höher und können diesen Ordner nach der Installation der AWS IoT Greengrass Core-Software entfernen.

Installieren Sie die AWS IoT Greengrass Core-Software.

Führen Sie das Installationsprogramm mit Argumenten aus, die Folgendes angeben:

- Erstellen Sie die AWS Ressourcen, die das Core-Gerät für den Betrieb benötigt.
- Geben Sie an, um den `ggc_user` Systembenutzer zum Ausführen von Softwarekomponenten auf dem Core-Gerät zu verwenden. Auf Linux-Geräten gibt dieser Befehl auch an, die `ggc_group` Systemgruppe zu verwenden, und das Installationsprogramm erstellt den Systembenutzer und die Systemgruppe für Sie.
- Richten Sie die AWS IoT Greengrass Core-Software als Systemservice ein, der beim Booten ausgeführt wird. Auf Linux-Geräten erfordert dies das Systemd init-System.

> **Important**
>
> Auf Windows-Core-Geräten müssen Sie die AWS IoT Greengrass Core-Software als Systemservice einrichten.

Um ein Entwicklungsgerät mit lokalen Entwicklungstools einzurichten, geben Sie das `--deploy-dev-tools true` -Argument an. Die Bereitstellung der lokalen Entwicklungstools kann nach Abschluss der Installation bis zu einer Minute dauern.

Weitere Informationen zu den Argumenten, die Sie angeben können, finden Sie unter Installer-Argumente.

> **Note**
>
> Wenn Sie AWS IoT Greengrass auf einem Gerät mit eingeschränktem Speicher ausführen, können Sie die Speichermenge steuern, die die -AWS IoT GreengrassCore-Software verwendet. Um die Speicherzuweisung zu steuern, können Sie Optionen für die JVM-Heap-Größe im `jvmOptions` Konfigurationsparameter in Ihrer Kernkomponente festlegen. Weitere Informationen finden Sie unter Steuern Sie die Speicherzuweisung mit JVM-Optionen.

So installieren Sie die AWS IoT Greengrass Core-Software:

1. Führen Sie das AWS IoT Greengrass Core-Installationsprogramm aus. Ersetzen Sie die Argumentwerte in Ihrem Befehl wie folgt.

   a. */greengrass/v2* oder *C:\greengrass\v2*: Der Pfad zum Stammordner, der zur Installation der AWS IoT Greengrass Core-Software verwendet werden soll.

   b. *GreengrassInstaller*. Der Pfad zu dem Ordner, in dem Sie das AWS IoT Greengrass Core-Softwareinstallationsprogramm entpackt haben.

   c. *Region*. Die , AWS-Region in der Ressourcen gefunden oder erstellt werden sollen.

   d. *MyGreengrassCore*. Der Name des AWS IoT Objekts für Ihr Greengrass-Kerngerät. Wenn das Objekt nicht vorhanden ist, erstellt das Installationsprogramm es. Das Installationsprogramm lädt die Zertifikate herunter, um sich als das AWS IoT Objekt zu authentifizieren. Weitere Informationen finden Sie unter Geräteauthentifizierung und -autorisierung für AWS IoT Greengrass.

> **Note**
>
> Der Objektname darf keine Doppelpunktzeichen (`:`) enthalten.

e. *`MyGreengrassCoreGroup`*. Der Name der AWS IoT Objektgruppe für Ihr Greengrass-Kerngerät. Wenn die Objektgruppe nicht vorhanden ist, erstellt das Installationsprogramm sie und fügt ihr das Objekt hinzu. Wenn die Objektgruppe vorhanden ist und über eine aktive Bereitstellung verfügt, lädt das Core-Gerät die Software herunter und führt sie aus, die in der Bereitstellung angegeben ist.

> **Note**
>
> Der Objektgruppenname darf keine Doppelpunktzeichen (`:`) enthalten.

f. *`GreengrassV2IoTThingPolicy`* . Der Name der AWS IoT Richtlinie, die den Greengrass-Core-Geräten die Kommunikation mit AWS IoT und ermöglichtAWS IoT Greengrass. Wenn die AWS IoT Richtlinie nicht vorhanden ist, erstellt das Installationsprogramm eine permissive AWS IoT Richtlinie mit diesem Namen. Sie können die Berechtigungen dieser Richtlinie für Ihren Anwendungsfall einschränken. Weitere Informationen finden Sie unter Minimale AWS IoT Richtlinie für -AWS IoT Greengrass V2Core-Geräte.

g. *`GreengrassV2TokenExchangeRole`*. Der Name der IAM-Rolle, die es dem Greengrass-Core-Gerät ermöglicht, temporäre AWS Anmeldeinformationen zu erhalten. Wenn die Rolle nicht vorhanden ist, erstellt das Installationsprogramm sie und erstellt und fügt eine Richtlinie mit dem Namen an*`GreengrassV2TokenExchangeRole`*`Access`. Weitere Informationen finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

h. *`GreengrassCoreTokenExchangeRoleAlias`*. Der Alias für die IAM-Rolle, die es dem Greengrass-Kerngerät ermöglicht, später temporäre Anmeldeinformationen zu erhalten. Wenn der Rollenalias nicht vorhanden ist, erstellt das Installationsprogramm ihn und verweist ihn auf die von Ihnen angegebene IAM-Rolle. Weitere Informationen finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

Linux or Unix

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
  -jar ./GreengrassInstaller/lib/Greengrass.jar \
```

```
  --aws-region region \
  --thing-name MyGreengrassCore \
  --thing-group-name MyGreengrassCoreGroup \
  --thing-policy-name GreengrassV2IoTThingPolicy \
  --tes-role-name GreengrassV2TokenExchangeRole \
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias \
  --component-default-user ggc_user:ggc_group \
  --provision true \
  --setup-system-service true
```

Windows Command Prompt (CMD)

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
  -jar ./GreengrassInstaller/lib/Greengrass.jar ^
  --aws-region region ^
  --thing-name MyGreengrassCore ^
  --thing-group-name MyGreengrassCoreGroup ^
  --thing-policy-name GreengrassV2IoTThingPolicy ^
  --tes-role-name GreengrassV2TokenExchangeRole ^
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias ^
  --component-default-user ggc_user ^
  --provision true ^
  --setup-system-service true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
  -jar ./GreengrassInstaller/lib/Greengrass.jar `
  --aws-region region `
  --thing-name MyGreengrassCore `
  --thing-group-name MyGreengrassCoreGroup `
  --thing-policy-name GreengrassV2IoTThingPolicy `
  --tes-role-name GreengrassV2TokenExchangeRole `
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias `
  --component-default-user ggc_user `
  --provision true `
  --setup-system-service true
```

> **Important**
>
> Auf Windows-Core-Geräten müssen Sie angeben, `--setup-system-service true` um die AWS IoT Greengrass Core-Software als Systemservice einzurichten.

Das Installationsprogramm druckt die folgenden Meldungen, wenn es erfolgreich ist:

- Wenn Sie angeben`--provision`, wird das Installationsprogramm gedruckt, `Successfully configured Nucleus with provisioned resource details` wenn die Ressourcen erfolgreich konfiguriert wurden.
- Wenn Sie angeben`--deploy-dev-tools`, wird das Installationsprogramm gedruckt, `Configured Nucleus to deploy aws.greengrass.Cli component` wenn die Bereitstellung erfolgreich erstellt wurde.
- Wenn Sie angeben`--setup-system-service true`, wird das Installationsprogramm gedruckt, `Successfully set up Nucleus as a system service` wenn es die Software als Service eingerichtet und ausgeführt hat.
- Wenn Sie nicht angeben`--setup-system-service true`, druckt das Installationsprogramm, `Launched Nucleus successfully` wenn es erfolgreich war, und führt die Software aus.

2. Überspringen Sie diesen Schritt, wenn Sie Grüngraskern v2.0.4 oder höher installiert haben. Wenn Sie die neueste Version der Software heruntergeladen haben, haben Sie v2.0.4 oder höher installiert.

   Führen Sie den folgenden Befehl aus, um die erforderlichen Dateiberechtigungen für Ihren Stammordner der -AWS IoT GreengrassCore-Software festzulegen. Ersetzen Sie durch */greengrass/v2* den Stammordner, den Sie in Ihrem Installationsbefehl angegeben haben, und ersetzen Sie */greengrass* durch den übergeordneten Ordner für Ihren Stammordner.

   ```
   sudo chmod 755 /greengrass/v2 && sudo chmod 755 /greengrass
   ```

Wenn Sie die AWS IoT Greengrass Core-Software als Systemservice installiert haben, führt das Installationsprogramm die Software für Sie aus. Andernfalls müssen Sie die Software manuell

ausführen. Weitere Informationen finden Sie unter Ausführen der AWS IoT Greengrass -Core-Software.

> **Note**
>
> Standardmäßig erlaubt die vom Installationsprogramm erstellte IAM-Rolle keinen Zugriff auf Komponentenartefakte in S3-Buckets. Um benutzerdefinierte Komponenten bereitzustellen, die Artefakte in Amazon S3 definieren, müssen Sie der Rolle Berechtigungen hinzufügen, damit Ihr Core-Gerät Komponentenartefakte abrufen kann. Weitere Informationen finden Sie unter Erlauben Sie den Zugriff auf S3-Buckets für Komponentenartefakte.
> Wenn Sie noch keinen S3-Bucket für Komponentenartefakte haben, können Sie diese Berechtigungen später hinzufügen, nachdem Sie einen Bucket erstellt haben.

Weitere Informationen zum Konfigurieren und Verwenden der Software und finden AWS IoT GreengrassSie im Folgenden:

- Konfigurieren Sie die AWS IoT Greengrass Core-Software
- Entwickeln von AWS IoT Greengrass Komponenten
- Bereitstellen von AWS IoT Greengrass Komponenten auf Geräten
- Greengrass-Befehlszeilenschnittstelle

Installieren Sie die AWS IoT Greengrass Core-Software mit manueller Ressourcenbereitstellung

Die AWS IoT Greengrass Core-Software enthält ein Installationsprogramm, das Ihr Gerät als Greengrass-Core-Gerät einrichtet. Um ein Gerät manuell einzurichten, können Sie die erforderlichen AWS IoT und IAM-Ressourcen für das Gerät erstellen. Wenn Sie diese Ressourcen manuell erstellen, müssen Sie keine AWS Anmeldeinformationen für das Installationsprogramm angeben.

Wenn Sie die AWS IoT Greengrass Core-Software manuell installieren, können Sie das Gerät auch so konfigurieren, dass es einen Netzwerk-Proxy verwendet oder eine Verbindung AWS über Port 443 herstellt. Möglicherweise müssen Sie diese Konfigurationsoptionen angeben, wenn Ihr Gerät beispielsweise hinter einer Firewall oder einem Netzwerk-Proxy läuft. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy.

Sie können die AWS IoT Greengrass Core-Software auch so konfigurieren, dass sie ein Hardware-Sicherheitsmodul (HSM) über die PKCS #11 -Schnittstelle verwendet. Mit dieser Funktion können Sie private Schlüssel- und Zertifikatsdateien sicher speichern, sodass sie nicht offengelegt oder in der Software dupliziert werden. Sie können private Schlüssel und Zertifikate auf einem Hardwaremodul wie einem HSM, einem Trusted Platform Module (TPM) oder einem anderen kryptografischen Element speichern. Diese Funktion ist nur auf Linux-Geräten verfügbar. Weitere Informationen zur Hardwaresicherheit und zu den Voraussetzungen für ihre Verwendung finden Sie unterIntegration von Hardware-Sicherheit.

> **Important**
>
> Bevor Sie die AWS IoT Greengrass Core-Software herunterladen, überprüfen Sie, ob Ihr Core-Gerät die Anforderungen für die Installation und Ausführung der AWS IoT Greengrass Core-Software v2.0 erfüllt.

Themen

Endpunkte abrufen AWS IoT

Holen Sie sich die AWS IoT Endpunkte für Sie und speichern Sie sie AWS-Konto, um sie später zu verwenden. Ihr Gerät verwendet diese Endpunkte, um eine Verbindung herzustellen. AWS IoT Gehen Sie wie folgt vor:

1. Holen Sie sich den AWS IoT Datenendpunkt für Ihren AWS-Konto.

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

Die Antwort sieht dem folgenden Beispiel ähnlich, wenn die Anfrage erfolgreich ist.

```
{
  "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
}
```

2. Rufen Sie den Endpunkt der AWS IoT Anmeldeinformationen für Ihren AWS-Konto ab.

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

Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anfrage erfolgreich ist.

```
{
  "endpointAddress": "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
}
```

Erstelle ein Ding AWS IoT

AWS IoT Dinge stehen für Geräte und logische Entitäten, mit denen eine Verbindung hergestellt wird AWS IoT. Greengrass-Core-Geräte sind AWS IoT Dinge. Wenn Sie ein Gerät als Objekt registrieren AWS IoT , kann dieses Gerät ein digitales Zertifikat zur Authentifizierung verwenden. AWS

In diesem Abschnitt erstellen Sie eine AWS IoT Sache, die Ihr Gerät repräsentiert.

Um ein AWS IoT Ding zu erstellen

1. Erstelle AWS IoT etwas für dein Gerät. Führen Sie auf Ihrem Entwicklungscomputer den folgenden Befehl aus.

   - Ersetzen Sie *MyGreengrassCore* durch den Namen des Dings, den Sie verwenden möchten. Dieser Name ist auch der Name Ihres Greengrass-Core-Geräts.

     > **Note**
     >
     > Der Name der Sache darf keine Doppelpunkte (`:`) enthalten.

```
aws iot create-thing --thing-name MyGreengrassCore
```

Die Antwort sieht dem folgenden Beispiel ähnlich, wenn die Anfrage erfolgreich ist.

```
{
  "thingName": "MyGreengrassCore",
  "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "thingId": "8cb4b6cd-268e-495d-b5b9-1713d71dbf42"
}
```

2. (Optional) Fügen Sie das AWS IoT Ding einer neuen oder vorhandenen Dinggruppe hinzu. Sie verwenden Dinggruppen, um Flotten von Greengrass-Kerngeräten zu verwalten. Wenn Sie Softwarekomponenten auf Ihren Geräten bereitstellen, können Sie einzelne Geräte oder Gerätegruppen gezielt ansprechen. Sie können ein Gerät zu einer Dinggruppe mit einer aktiven Greengrass-Bereitstellung hinzufügen, um die Softwarekomponenten dieser Dinggruppe auf dem Gerät bereitzustellen. Gehen Sie wie folgt vor:

   a. (Optional) Erstellen Sie eine AWS IoT Dinggruppe.

      - *MyGreengrassCoreGroup*Ersetzen Sie durch den Namen der zu erstellenden Dinggruppe.

        > **Note**
        >
        > Der Name der Dinggruppe darf keine Doppelpunkte (`:`) enthalten.

      ```
      aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
      ```

      Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anfrage erfolgreich ist.

      ```
      {
        "thingGroupName": "MyGreengrassCoreGroup",
        "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
      MyGreengrassCoreGroup",
        "thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"
      }
      ```

b. Fügt das AWS IoT Ding einer Dinggruppe hinzu.

- Ersetze *MyGreengrassCore* durch den Namen deines AWS IoT Dings.
- Ersetze es *MyGreengrassCoreGroup*durch den Namen der Dinggruppe.

```
aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-
name MyGreengrassCoreGroup
```

Der Befehl hat keine Ausgabe, wenn die Anfrage erfolgreich ist.

Erstellen Sie das Ding-Zertifikat

Wenn Sie ein Gerät als AWS IoT Ding registrieren, kann dieses Gerät ein digitales Zertifikat zur Authentifizierung verwenden. AWS Dieses Zertifikat ermöglicht dem Gerät die Kommunikation mit AWS IoT und AWS IoT Greengrass.

In diesem Abschnitt erstellen und laden Sie Zertifikate herunter, mit denen Ihr Gerät eine Verbindung herstellen kann AWS.

Wenn Sie die AWS IoT Greengrass Core-Software so konfigurieren möchten, dass sie ein Hardware-Sicherheitsmodul (HSM) verwendet, um den privaten Schlüssel und das Zertifikat sicher zu speichern, gehen Sie wie folgt vor, um das Zertifikat aus einem privaten Schlüssel in einem HSM zu erstellen. Folgen Sie andernfalls den Schritten zum Erstellen des Zertifikats und des privaten Schlüssels im AWS IoT Dienst. Die Hardware-Sicherheitsfunktion ist nur auf Linux-Geräten verfügbar. Weitere Informationen zur Hardwaresicherheit und zu den Voraussetzungen für ihre Verwendung finden Sie unterIntegration von Hardware-Sicherheit.

Erstellen Sie das Zertifikat und den privaten Schlüssel im AWS IoT Dienst

Um das Ding-Zertifikat zu erstellen

1. Erstellen Sie einen Ordner, in den Sie die Zertifikate für das AWS IoT Ding herunterladen.

```
mkdir greengrass-v2-certs
```

2. Erstellen Sie die Zertifikate für das AWS IoT Ding und laden Sie sie herunter.

```
aws iot create-keys-and-certificate --set-as-active --certificate-pem-outfile
 greengrass-v2-certs/device.pem.crt --public-key-outfile greengrass-v2-certs/
public.pem.key --private-key-outfile greengrass-v2-certs/private.pem.key
```

Die Antwort ähnelt dem folgenden Beispiel, wenn die Anfrage erfolgreich ist.

```
{
  "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificateId":
 "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificatePem": "-----BEGIN CERTIFICATE-----
MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w
 0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
 WF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIw
 EAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5
 jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
 MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBb
 WF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMx
 HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wgZ8wDQYJKoZIhvcNAQE
 BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
 k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
 ITxOUSQv7c7ugFFDzQGBzZswY6786m86gpEIbb3OhjZnzcvQAaRHhdlQWIMm2nr
 AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
 KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJIlJ00zbhNYS5f6Guo
 EDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
 3rrszlaEXAMPLE=
-----END CERTIFICATE-----",
  "keyPair": {
    "PublicKey": "-----BEGIN PUBLIC KEY-----\
MIIBIjANBgkqhkEXAMPLEQEFAAOCAQ8AMIIBCgKCAQEAEXAMPLE1nnyJwKSMHw4h\
MMEXAMPLEuuN/dMAS3fyce8DW/4+EXAMPLEyjmoF/YVF/gHr99VEEXAMPLE5VF13\
59VK7cEXAMPLE67GK+y+jikqXOgHh/xJTwo
+sGpWEXAMPLEDz18xOd2ka4tCzuWEXAMPLEahJbYkCPUBSU8opVkR7qkEXAMPLE1DR6sx2HocliOOLtu6Fkw91swQWE
\\GB3ZPrNh0PzQYvjUStZeccyNCx2EXAMPLEvp9mQOUXP6plfgxwKRX2fEXAMPLEDa\
hJLXkX3rHU2xbxJSq7D+XEXAMPLEcw+LyFhI5mgFRl88eGdsAEXAMPLElnI9EesG\
FQIDAQAB\
-----END PUBLIC KEY-----\
",
    "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\
key omitted for security reasons\
-----END RSA PRIVATE KEY-----\
```

```
"
  }
}
```

Speichern Sie den Amazon-Ressourcennamen (ARN) des Zertifikats, um das Zertifikat später zu konfigurieren.

Erstellen Sie das Zertifikat aus einem privaten Schlüssel in einem HSM

> **Note**
>
> Diese Funktion ist für Version 2.5.3 und höher der Greengrass Nucleus-Komponente verfügbar. AWS IoT Greengrass unterstützt diese Funktion derzeit nicht auf Windows Core-Geräten.

Um das Ding-Zertifikat zu erstellen

1. Initialisieren Sie auf dem Core-Gerät ein PKCS #11 -Token im HSM und generieren Sie einen privaten Schlüssel. Der private Schlüssel muss ein RSA-Schlüssel mit einer RSA-2048-Schlüsselgröße (oder größer) oder ein ECC-Schlüssel sein.

   > **Note**
   >
   > Um ein Hardware-Sicherheitsmodul mit ECC-Schlüsseln zu verwenden, müssen Sie Greengrass Nucleus v2.5.6 oder höher verwenden.
   > Um ein Hardware-Sicherheitsmodul und einen Secret Manager zu verwenden, müssen Sie ein Hardware-Sicherheitsmodul mit RSA-Schlüsseln verwenden.

   In der Dokumentation zu Ihrem HSM erfahren Sie, wie Sie das Token initialisieren und den privaten Schlüssel generieren. Wenn Ihr HSM Objekt-IDs unterstützt, geben Sie bei der Generierung des privaten Schlüssels eine Objekt-ID an. Speichern Sie die Slot-ID, die Benutzer-PIN, die Objektbezeichnung und die Objekt-ID (falls Ihr HSM eine verwendet), die Sie angeben, wenn Sie das Token initialisieren und den privaten Schlüssel generieren. Sie verwenden diese Werte später, wenn Sie das Ding-Zertifikat in das HSM importieren und die Core-Software konfigurieren. AWS IoT Greengrass

2. Erstellen Sie eine Zertifikatsignieranforderung (CSR) aus dem privaten Schlüssel. AWS IoT verwendet diese CSR, um ein Ding-Zertifikat für den privaten Schlüssel zu erstellen, den Sie im HSM generiert haben. Informationen zum Erstellen einer CSR aus dem privaten Schlüssel finden Sie in der Dokumentation zu Ihrem HSM. Die CSR ist eine Datei, wie z. B. `iotdevicekey.csr`

3. Kopieren Sie die CSR vom Gerät auf Ihren Entwicklungscomputer. Wenn SSH und SCP auf dem Entwicklungscomputer und dem Gerät aktiviert sind, können Sie den `scp` Befehl auf Ihrem Entwicklungscomputer verwenden, um die CSR zu übertragen. Ersetzen Sie *`device-ip-address`* durch die IP-Adresse Ihres Geräts und ersetzen Sie *`~/iotdevicekey.csr`* durch den Pfad zur CSR-Datei auf dem Gerät.

   ```
   scp device-ip-address:~/iotdevicekey.csr iotdevicekey.csr
   ```

4. Erstellen Sie auf Ihrem Entwicklungscomputer einen Ordner, in den Sie das Zertifikat für das Ding herunterladen. AWS IoT

   ```
   mkdir greengrass-v2-certs
   ```

5. Verwenden Sie die CSR-Datei, um das Zertifikat für das AWS IoT Ding zu erstellen und auf Ihren Entwicklungscomputer herunterzuladen.

   ```
   aws iot create-certificate-from-csr --set-as-active --certificate-signing-
   request=file://iotdevicekey.csr --certificate-pem-outfile greengrass-v2-certs/
   device.pem.crt
   ```

   Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anfrage erfolgreich ist.

   ```
   {
     "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
   aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
     "certificateId":
    "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
     "certificatePem": "-----BEGIN CERTIFICATE-----
   MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w
    0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
    WF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIw
    EAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5
    jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
    MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBb
    WF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMx
    HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wgZ8wDQYJKoZIhvcNAQE
   ```

```
 BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
 k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
 ITxOUSQv7c7ugFFDzQGBzZswY6786m86gpEIbb3OhjZnzcvQAaRHhdlQWIMm2nr
 AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
 KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJIlJ00zbhNYS5f6Guo
 EDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
 3rrszlaEXAMPLE=
-----END CERTIFICATE-----"
}
```

Speichern Sie den ARN des Zertifikats, um das Zertifikat später zu konfigurieren.

Konfigurieren Sie das Ding-Zertifikat

Hängen Sie das Ding-Zertifikat an das AWS IoT Ding an, das Sie zuvor erstellt haben, und fügen Sie dem Zertifikat eine AWS IoT Richtlinie hinzu, um die AWS IoT Berechtigungen für das Kerngerät zu definieren.

Um das Zertifikat des Dings zu konfigurieren

1. Hängen Sie das Zertifikat an das AWS IoT Ding an.

   - Ersetze *MyGreengrassCore* durch den Namen deines AWS IoT Dings.
   - Ersetzen Sie das Zertifikat Amazon Resource Name (ARN) durch den ARN des Zertifikats, das Sie im vorherigen Schritt erstellt haben.

   ```
   aws iot attach-thing-principal --thing-name MyGreengrassCore
    --principal arn:aws:iot:us-west-2:123456789012:cert/
   aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
   ```

   Der Befehl hat keine Ausgabe, wenn die Anfrage erfolgreich ist.

2. Erstellen Sie eine AWS IoT Richtlinie, die die AWS IoT Berechtigungen für Ihr Greengrass-Core-Gerät definiert, und fügen Sie sie hinzu. Die folgende Richtlinie ermöglicht den Zugriff auf alle MQTT-Themen und Greengrass-Operationen, sodass Ihr Gerät mit benutzerdefinierten Anwendungen und future Änderungen, die neue Greengrass-Operationen erfordern, funktioniert. Sie können diese Richtlinie je nach Anwendungsfall einschränken. Weitere Informationen finden Sie unter Minimale AWS IoT Richtlinie für -AWS IoT Greengrass V2Core-Geräte.

Wenn Sie bereits ein Greengrass-Core-Gerät eingerichtet haben, können Sie dessen AWS IoT Richtlinie anhängen, anstatt eine neue zu erstellen.

Gehen Sie wie folgt vor:

a. Erstellen Sie eine Datei, die das AWS IoT Richtliniendokument enthält, das für Greengrass-Core-Geräte erforderlich ist.

   Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um die Datei mit GNU Nano zu erstellen.

   ```
   nano greengrass-v2-iot-policy.json
   ```

   Kopieren Sie den folgenden JSON-Code in die Datei.

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": [
           "iot:Publish",
           "iot:Subscribe",
           "iot:Receive",
           "iot:Connect",
           "greengrass:*"
         ],
         "Resource": [
           "*"
         ]
       }
     ]
   }
   ```

b. Erstellen Sie eine AWS IoT Richtlinie aus dem Richtliniendokument.

   - Ersetzen Sie *GreengrassV2IoT* durch den Namen der ThingPolicy zu erstellenden Richtlinie.

```
aws iot create-policy --policy-name GreengrassV2IoTThingPolicy --policy-
document file://greengrass-v2-iot-policy.json
```

Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anfrage erfolgreich ist.

```
{
  "policyName": "GreengrassV2IoTThingPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
  "policyDocument": "{
    \\"Version\\": \\"2012-10-17\\",
    \\"Statement\\": [
      {
        \\"Effect\\": \\"Allow\\",
        \\"Action\\": [
          \\"iot:Publish\\",
          \\"iot:Subscribe\\",
          \\"iot:Receive\\",
          \\"iot:Connect\\",
          \\"greengrass:*\\"
        ],
        \\"Resource\\": [
          \\"*\\"
        ]
      }
    ]
  }",
  "policyVersionId": "1"
}
```

c. Hängen Sie die AWS IoT Richtlinie an das Zertifikat der AWS IoT Sache an.

- Ersetzen Sie *GreengrassV2IoT* durch den Namen der ThingPolicy Richtlinie, die angehängt werden soll.
- Ersetzen Sie den Ziel-ARN durch den ARN des Zertifikats für Ihr AWS IoT Ding.

```
aws iot attach-policy --policy-name GreengrassV2IoTThingPolicy
 --target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

Der Befehl hat keine Ausgabe, wenn die Anfrage erfolgreich ist.

Erstellen Sie eine Token-Exchange-Rolle

Greengrass-Core-Geräte verwenden eine IAM-Servicerolle, die so genannte Token-Exchange-Rolle, um Aufrufe von Diensten zu autorisieren. AWS Das Gerät verwendet den AWS IoT Anmeldeinformationsanbieter, um temporäre AWS Anmeldeinformationen für diese Rolle abzurufen. Dadurch kann das Gerät mit Amazon Logs interagieren AWS IoT, Protokolle an Amazon CloudWatch Logs senden und benutzerdefinierte Komponentenartefakte von Amazon S3 herunterladen. Weitere Informationen finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

Sie verwenden einen AWS IoT Rollenalias, um die Token-Austauschrolle für Greengrass-Core-Geräte zu konfigurieren. Rollenaliase ermöglichen es Ihnen, die Token-Austauschrolle für ein Gerät zu ändern, aber die Gerätekonfiguration beizubehalten. Weitere Informationen finden Sie im AWS IoT Core Entwicklerhandbuch unter Autorisieren von direkten Aufrufen von AWS Diensten.

In diesem Abschnitt erstellen Sie eine Token-Exchange-IAM-Rolle und einen AWS IoT Rollenalias, der auf die Rolle verweist. Wenn Sie bereits ein Greengrass-Core-Gerät eingerichtet haben, können Sie dessen Token-Austauschrolle und seinen Rollenalias verwenden, anstatt neue zu erstellen. Anschließend konfigurieren Sie das Gerät so, dass es AWS IoT diese Rolle und diesen Alias verwendet.

Um eine Token-Exchange-IAM-Rolle zu erstellen

1. Erstellen Sie eine IAM-Rolle, die Ihr Gerät als Token-Austauschrolle verwenden kann. Gehen Sie wie folgt vor:

   a. Erstellen Sie eine Datei, die das Dokument mit der Vertrauensrichtlinie enthält, das für die Token-Austauschrolle erforderlich ist.

      Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um die Datei mit GNU Nano zu erstellen.

      ```
      nano device-role-trust-policy.json
      ```

      Kopieren Sie den folgenden JSON-Code in die Datei.

      ```
      {
      ```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. Erstellen Sie die Token-Austauschrolle mit dem Dokument zur Vertrauensrichtlinie.

- Ersetzen Sie die *TokenExchangeGreengrassV2-Rolle* durch den Namen der zu erstellenden IAM-Rolle.

```
aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-
policy-document file://device-role-trust-policy.json
```

Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anfrage erfolgreich ist.

```
{
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassV2TokenExchangeRole",
    "RoleId": "AROAZ2YMUHYHK5OKM77FB",
    "Arn": "arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole",
    "CreateDate": "2021-02-06T00:13:29+00:00",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "credentials.iot.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
```

```
  }
```

c. Erstellen Sie eine Datei, die das Dokument mit der Zugriffsrichtlinie enthält, das für die Token-Austauschrolle erforderlich ist.

   Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um die Datei mit GNU Nano zu erstellen.

```
nano device-role-access-policy.json
```

   Kopieren Sie den folgenden JSON-Code in die Datei.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

> **Note**
>
> Diese Zugriffsrichtlinie erlaubt keinen Zugriff auf Komponentenartefakte in S3-Buckets. Um benutzerdefinierte Komponenten bereitzustellen, die Artefakte in Amazon S3 definieren, müssen Sie der Rolle Berechtigungen hinzufügen, damit Ihr Kerngerät Komponentenartefakte abrufen kann. Weitere Informationen finden Sie unter Erlauben Sie den Zugriff auf S3-Buckets für Komponentenartefakte.
> Wenn Sie noch keinen S3-Bucket für Komponentenartefakte haben, können Sie diese Berechtigungen später hinzufügen, nachdem Sie einen Bucket erstellt haben.

d. Erstellen Sie die IAM-Richtlinie anhand des Richtliniendokuments.

- Ersetzen Sie *GreengrassV2 TokenExchange RoleAccess* durch den Namen der zu erstellenden IAM-Richtlinie.

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --
policy-document file://device-role-access-policy.json
```

Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anfrage erfolgreich ist.

```
{
  "Policy": {
    "PolicyName": "GreengrassV2TokenExchangeRoleAccess",
    "PolicyId": "ANPAZ2YMUHYHACI7C5Z66",
    "Arn": "arn:aws:iam::123456789012:policy/
GreengrassV2TokenExchangeRoleAccess",
    "Path": "/",
    "DefaultVersionId": "v1",
    "AttachmentCount": 0,
    "PermissionsBoundaryUsageCount": 0,
    "IsAttachable": true,
    "CreateDate": "2021-02-06T00:37:17+00:00",
    "UpdateDate": "2021-02-06T00:37:17+00:00"
  }
}
```

e. Hängen Sie die IAM-Richtlinie an die Token-Exchange-Rolle an.

- Ersetzen Sie die *TokenExchangeGreengrassV2-Rolle* durch den Namen der IAM-Rolle.
- Ersetzen Sie den Richtlinien-ARN durch den ARN der IAM-Richtlinie, die Sie im vorherigen Schritt erstellt haben.

```
aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-
arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess
```

Der Befehl hat keine Ausgabe, wenn die Anfrage erfolgreich ist.

2. Erstellen Sie einen AWS IoT Rollenalias, der auf die Token-Exchange-Rolle verweist.

- *GreengrassCoreTokenExchangeRoleAlias*Ersetzen Sie ihn durch den Namen des Rollenalias, den Sie erstellen möchten.
- Ersetzen Sie den Rollen-ARN durch den ARN der IAM-Rolle, die Sie im vorherigen Schritt erstellt haben.

```
aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-
arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole
```

Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anfrage erfolgreich ist.

```
{
  "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
}
```

> **Note**
>
> Um einen Rollenalias zu erstellen, benötigen Sie die Berechtigung, die Token-Exchange-IAM-Rolle zu übergeben. AWS IoT Wenn Sie beim Versuch, einen Rollenalias zu erstellen, eine Fehlermeldung erhalten, überprüfen Sie, ob Ihr AWS Benutzer über diese Berechtigung verfügt. Weitere Informationen finden Sie im Benutzerhandbuch unter Erteilen von Benutzerberechtigungen zur Übergabe einer Rolle an einen AWS Dienst.AWS Identity and Access Management

3. Erstellen und fügen Sie eine AWS IoT Richtlinie hinzu, die es Ihrem Greengrass-Core-Gerät ermöglicht, den Rollenalias zu verwenden, um die Token-Austauschrolle zu übernehmen. Wenn Sie bereits ein Greengrass-Core-Gerät eingerichtet haben, können Sie dessen AWS IoT Rollenalias-Richtlinie anhängen, anstatt eine neue zu erstellen. Gehen Sie wie folgt vor:

   a. (Optional) Erstellen Sie eine Datei, die das AWS IoT Richtliniendokument enthält, das für den Rollenalias erforderlich ist.

      Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um die Datei mit GNU Nano zu erstellen.

```
nano greengrass-v2-iot-role-alias-policy.json
```

Kopieren Sie den folgenden JSON-Code in die Datei.

- Ersetzen Sie den Ressourcen-ARN durch den ARN Ihres Rollenalias.

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:AssumeRoleWithCertificate",
      "Resource": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
    }
  ]
}
```

b. Erstellen Sie eine AWS IoT Richtlinie aus dem Richtliniendokument.

- Ersetzen Sie *GreengrassCoreTokenExchangeRoleAliasPolicy* durch den Namen der zu erstellenden AWS IoT Richtlinie.

```
aws iot create-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
 --policy-document file://greengrass-v2-iot-role-alias-policy.json
```

Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anfrage erfolgreich ist.

```
{
  "policyName": "GreengrassCoreTokenExchangeRoleAliasPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassCoreTokenExchangeRoleAliasPolicy",
  "policyDocument": "{
    \\"Version\\":\\"2012-10-17\\",
    \\"Statement\\": [
      {
        \\"Effect\\": \\"Allow\\",
        \\"Action\\": \\"iot:AssumeRoleWithCertificate\\",
```

```
        \\"Resource\\": \\"arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias\\"
      }
    ]
  }",
  "policyVersionId": "1"
}
```

c. Hängen Sie die AWS IoT Richtlinie an das Zertifikat der AWS IoT Sache an.

- Ersetzen Sie *GreengrassCoreTokenExchangeRoleAliasPolicy* durch den Namen der AWS IoT Rollen-Alias-Richtlinie.
- Ersetzen Sie den Ziel-ARN durch den ARN des Zertifikats für Ihr AWS IoT Ding.

```
aws iot attach-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
 --target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

Der Befehl hat keine Ausgabe, wenn die Anfrage erfolgreich ist.

Laden Sie Zertifikate auf das Gerät herunter

Zuvor haben Sie das Zertifikat Ihres Geräts auf Ihren Entwicklungscomputer heruntergeladen. In diesem Abschnitt kopieren Sie das Zertifikat auf Ihr Kerngerät, um das Gerät mit den Zertifikaten einzurichten, mit denen es eine Verbindung herstellt AWS IoT. Sie laden auch das Zertifikat der Amazon Root Certificate Authority (CA) herunter. Wenn Sie ein HSM verwenden, importieren Sie in diesem Abschnitt auch die Zertifikatsdatei in das HSM.

- Wenn Sie das Ding-Zertifikat und den privaten Schlüssel zuvor im AWS IoT Dienst erstellt haben, folgen Sie den Schritten, um die Zertifikate mit dem privaten Schlüssel und den Zertifikatsdateien herunterzuladen.
- Wenn Sie das Ding-Zertifikat zuvor aus einem privaten Schlüssel in einem Hardware-Sicherheitsmodul (HSM) erstellt haben, gehen Sie wie folgt vor, um die Zertifikate mit dem privaten Schlüssel und dem Zertifikat in ein HSM herunterzuladen.

Laden Sie Zertifikate mit privatem Schlüssel und Zertifikatsdateien herunter

Um Zertifikate auf das Gerät herunterzuladen

1. Kopieren Sie das AWS IoT Ding-Zertifikat von Ihrem Entwicklungscomputer auf das Gerät. Wenn SSH und SCP auf dem Entwicklungscomputer und dem Gerät aktiviert sind, können Sie den `scp` Befehl auf Ihrem Entwicklungscomputer verwenden, um das Zertifikat zu übertragen. Ersetzen Sie die *Geräte-IP-Adresse* durch die IP-Adresse Ihres Geräts.

   ```
   scp -r greengrass-v2-certs/ device-ip-address:~
   ```

2. Erstellen Sie den Greengrass-Stammordner auf dem Gerät. Sie werden später die AWS IoT Greengrass Core-Software in diesem Ordner installieren.

   Linux or Unix

   - Ersetzen Sie es */greengrass/v2* durch den Ordner, den Sie verwenden möchten.

   ```
   sudo mkdir -p /greengrass/v2
   ```

   Windows Command Prompt

   - Ersetzen Sie *C:\greengrass\v2* durch den zu verwendenden Ordner.

   ```
   mkdir C:\greengrass\v2
   ```

   PowerShell

   - Ersetzen Sie *C:\greengrass\v2* durch den Ordner, den Sie verwenden möchten.

   ```
   mkdir C:\greengrass\v2
   ```

3. (Nur Linux) Legen Sie die Berechtigungen des übergeordneten Elements des Greengrass-Stammordners fest.

   - Ersetzen Sie */greengrass* durch das übergeordnete Objekt des Stammordners.

```
sudo chmod 755 /greengrass
```

4. Kopieren Sie die AWS IoT Ding-Zertifikate in den Greengrass-Stammordner.

   Linux or Unix

   - */greengrass/v2*Ersetzen Sie es durch den Greengrass-Stammordner.

   ```
   sudo cp -R ~/greengrass-v2-certs/* /greengrass/v2
   ```

   Windows Command Prompt

   - Ersetzen Sie *C:\greengrass\v2* durch den Ordner, den Sie verwenden möchten.

   ```
   robocopy %USERPROFILE%\greengrass-v2-certs C:\greengrass\v2 /E
   ```

   PowerShell

   - Ersetzen Sie *C:\greengrass\v2* durch den Ordner, den Sie verwenden möchten.

   ```
   cp -Path ~\greengrass-v2-certs\* -Destination C:\greengrass\v2
   ```

5. Laden Sie das Zertifikat der Amazon Root Certificate Authority (CA) herunter. AWS IoT Zertifikate sind standardmäßig mit dem Root-CA-Zertifikat von Amazon verknüpft.

   Linux or Unix

   ```
   sudo curl -o /greengrass/v2/AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   ```

   Windows Command Prompt (CMD)

   ```
   curl -o C:\greengrass\v2\\AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   ```

PowerShell

```
iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile C:
\greengrass\v2\\AmazonRootCA1.pem
```

Laden Sie Zertifikate mit dem privaten Schlüssel und dem Zertifikat in ein HSM herunter

> **Note**
>
> Diese Funktion ist für Version 2.5.3 und höher der Greengrass Nucleus-Komponente verfügbar. AWS IoT Greengrass unterstützt diese Funktion derzeit nicht auf Windows Core-Geräten.

Um Zertifikate auf das Gerät herunterzuladen

1. Kopieren Sie das AWS IoT Ding-Zertifikat von Ihrem Entwicklungscomputer auf das Gerät. Wenn SSH und SCP auf dem Entwicklungscomputer und dem Gerät aktiviert sind, können Sie den `scp` Befehl auf Ihrem Entwicklungscomputer verwenden, um das Zertifikat zu übertragen. Ersetzen Sie die *Geräte-IP-Adresse* durch die IP-Adresse Ihres Geräts.

   ```
   scp -r greengrass-v2-certs/ device-ip-address:~
   ```

2. Erstellen Sie den Greengrass-Stammordner auf dem Gerät. Sie werden später die AWS IoT Greengrass Core-Software in diesem Ordner installieren.

   Linux or Unix

   - Ersetzen Sie es */greengrass/v2* durch den Ordner, den Sie verwenden möchten.

   ```
   sudo mkdir -p /greengrass/v2
   ```

   Windows Command Prompt

   - Ersetzen Sie *C:\greengrass\v2* durch den zu verwendenden Ordner.

```
mkdir C:\greengrass\v2
```

PowerShell

- Ersetzen Sie *C:\greengrass\v2* durch den Ordner, den Sie verwenden möchten.

```
mkdir C:\greengrass\v2
```

3. (Nur Linux) Legen Sie die Berechtigungen des übergeordneten Elements des Greengrass-Stammordners fest.

   - Ersetzen Sie */greengrass* durch das übergeordnete Objekt des Stammordners.

   ```
   sudo chmod 755 /greengrass
   ```

4. Importieren Sie die Ding-Zertifikatsdatei,`~/greengrass-v2-certs/device.pem.crt`, in das HSM. In der Dokumentation zu Ihrem HSM erfahren Sie, wie Sie Zertifikate in Ihr HSM importieren können. Importieren Sie das Zertifikat mit demselben Token, derselben Slot-ID, derselben Benutzer-PIN, derselben Objektbezeichnung und Objekt-ID (falls Ihr HSM eine verwendet), mit denen Sie zuvor den privaten Schlüssel im HSM generiert haben.

   > **Note**
   >
   > Wenn Sie den privaten Schlüssel zuvor ohne Objekt-ID generiert haben und das Zertifikat eine Objekt-ID hat, setzen Sie die Objekt-ID des privaten Schlüssels auf denselben Wert wie das Zertifikat. In der Dokumentation zu Ihrem HSM erfahren Sie, wie Sie die Objekt-ID für das Objekt mit dem privaten Schlüssel festlegen.

5. (Optional) Löschen Sie die Ding-Zertifikatsdatei, sodass sie nur im HSM existiert.

   ```
   rm ~/greengrass-v2-certs/device.pem.crt
   ```

6. Laden Sie das Zertifikat der Amazon Root Certificate Authority (CA) herunter. AWS IoT Zertifikate sind standardmäßig mit dem Root-CA-Zertifikat von Amazon verknüpft.

Linux or Unix

```
sudo curl -o /greengrass/v2/AmazonRootCA1.pem https://www.amazontrust.com/
repository/AmazonRootCA1.pem
```

Windows Command Prompt (CMD)

```
curl -o C:\greengrass\v2\\AmazonRootCA1.pem https://www.amazontrust.com/
repository/AmazonRootCA1.pem
```

PowerShell

```
iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile C:
\greengrass\v2\\AmazonRootCA1.pem
```

Richten Sie die Geräteumgebung ein

Folgen Sie den Schritten in diesem Abschnitt, um ein Linux- oder Windows-Gerät einzurichten, das als Ihr AWS IoT Greengrass Kerngerät verwendet werden soll.

Richten Sie ein Linux-Gerät ein

Um ein Linux-Gerät einzurichten für AWS IoT Greengrass V2

1. Installieren Sie die Java-Runtime, die für die Ausführung der AWS IoT Greengrass Core-Software erforderlich ist. Wir empfehlen, dass Sie die Langzeit-Support-Versionen von Amazon Corretto oder OpenJDK verwenden. Version 8 oder höher ist erforderlich. Die folgenden Befehle zeigen Ihnen, wie Sie OpenJDK auf Ihrem Gerät installieren.

   - Für Debian- oder Ubuntu-basierte Distributionen:

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

   - Für Red Hat-basierte Distributionen:

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

   - Für Amazon Linux 2:

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

- Für Amazon Linux 2023:

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

Führen Sie nach Abschluss der Installation den folgenden Befehl aus, um zu überprüfen, ob Java auf Ihrem Linux-Gerät ausgeführt wird.

```
java -version
```

Der Befehl druckt die Version von Java, die auf dem Gerät ausgeführt wird. Bei einer Debian-basierten Distribution könnte die Ausgabe beispielsweise dem folgenden Beispiel ähneln.

```
openjdk version "11.0.9.1" 2020-11-04
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
```

2. (Optional) Erstellen Sie den Standardsystembenutzer und die Standardgruppe, die Komponenten auf dem Gerät ausführen. Sie können auch festlegen, dass der AWS IoT Greengrass Core-Software-Installer diesen Benutzer und diese Gruppe während der Installation mit dem `--component-default-user` Installer-Argument erstellt. Weitere Informationen finden Sie unter Installer-Argumente.

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

3. Stellen Sie sicher, dass der Benutzer, der die AWS IoT Greengrass Core-Software ausführt (normalerweise`root`), über die erforderlichen Rechte verfügt, die Software `sudo` mit jedem beliebigen Benutzer und jeder Gruppe auszuführen.

   a. Führen Sie den folgenden Befehl aus, um die `/etc/sudoers` Datei zu öffnen.

   ```
   sudo visudo
   ```

   b. Stellen Sie sicher, dass die Berechtigung für den Benutzer wie im folgenden Beispiel aussieht.

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

4. (Optional) Um containerisierte Lambda-Funktionen auszuführen, müssen Sie cgroups v1 aktivieren und Sie müssen die Speicher - und Geräte-Cgroups aktivieren und mounten. Wenn Sie nicht vorhaben, containerisierte Lambda-Funktionen auszuführen, können Sie diesen Schritt überspringen.

   Um diese Cgroups-Optionen zu aktivieren, starten Sie das Gerät mit den folgenden Linux-Kernelparametern.

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

   Informationen zum Anzeigen und Einstellen der Kernel-Parameter für Ihr Gerät finden Sie in der Dokumentation zu Ihrem Betriebssystem und Bootloader. Folgen Sie den Anweisungen, um die Kernel-Parameter dauerhaft einzustellen.
5. Installieren Sie alle anderen erforderlichen Abhängigkeiten auf Ihrem Gerät, wie in der Liste der Anforderungen unter angegebenAnforderungen an Speichergeräte.

Richten Sie ein Windows-Gerät ein

> **Note**
>
> Diese Funktion ist für Version 2.5.0 und höher der Greengrass Nucleus-Komponente verfügbar.

Um ein Windows-Gerät einzurichten für AWS IoT Greengrass V2

1. Installieren Sie die Java-Runtime, die für die Ausführung der AWS IoT Greengrass Core-Software erforderlich ist. Wir empfehlen, dass Sie die Langzeit-Support-Versionen von Amazon Corretto oder OpenJDK verwenden. Version 8 oder höher ist erforderlich.
2. Prüfen Sie, ob Java in der Systemvariablen PATH verfügbar ist, und fügen Sie es hinzu, falls nicht. Auf dem LocalSystem Konto wird die AWS IoT Greengrass Core-Software ausgeführt, sodass Sie der Systemvariablen PATH statt der Benutzervariablen PATH für Ihren Benutzer Java hinzufügen müssen. Gehen Sie wie folgt vor:

   a. Drücken Sie die Windows-Taste, um das Startmenü zu öffnen.

b. Geben Sie **`environment variables`** ein, um im Startmenü nach den Systemoptionen zu suchen.

c. Wählen Sie in den Suchergebnissen des Startmenüs die Option Systemumgebungsvariablen bearbeiten aus, um das Fenster mit den Systemeigenschaften zu öffnen.

d. Wählen Sie Umgebungsvariablen... um das Fenster Umgebungsvariablen zu öffnen.

e. Wählen Sie unter Systemvariablen die Option Pfad und dann Bearbeiten aus. Im Fenster Umgebungsvariable bearbeiten können Sie jeden Pfad in einer separaten Zeile anzeigen.

f. Überprüfen Sie, ob der Pfad zum `bin` Ordner der Java-Installation vorhanden ist. Der Pfad könnte dem folgenden Beispiel ähneln.

```
C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
```

g. Wenn der `bin` Ordner der Java-Installation in Path fehlt, wählen Sie Neu, um ihn hinzuzufügen, und klicken Sie dann auf OK.

3. Öffnen Sie die Windows-Eingabeaufforderung (`cmd.exe`) als Administrator.

4. Erstellen Sie den Standardbenutzer für das LocalSystem Konto auf dem Windows-Gerät. Ersetzen Sie *`das Passwort`* durch ein sicheres Passwort.

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

> **Tip**
>
> Abhängig von Ihrer Windows-Konfiguration ist das Benutzerkennwort möglicherweise so eingestellt, dass es an einem Datum in der future abläuft. Um sicherzustellen, dass Ihre Greengrass-Anwendungen weiterhin funktionieren, verfolgen Sie, wann das Passwort abläuft, und aktualisieren Sie es, bevor es abläuft. Sie können das Benutzerkennwort auch so einrichten, dass es niemals abläuft.
>
> - Führen Sie den folgenden Befehl aus, um zu überprüfen, wann ein Benutzer und sein Passwort ablaufen.
>
> ```
> net user ggc_user | findstr /C:expires
> ```
>
> - Führen Sie den folgenden Befehl aus, um das Passwort eines Benutzers so einzustellen, dass es nie abläuft.

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

- Wenn Sie Windows 10 oder höher verwenden und der wmicBefehl veraltet ist, führen Sie den folgenden PowerShell Befehl aus.

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

5. Laden Sie das PsExecProgramm von Microsoft herunter und installieren Sie es auf dem Gerät.
6. Verwenden Sie das PsExec Hilfsprogramm, um den Benutzernamen und das Passwort für den Standardbenutzer in der Credential Manager-Instanz für das LocalSystem Konto zu speichern. Ersetzen Sie *das Passwort* durch das Passwort des Benutzers, das Sie zuvor festgelegt haben.

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

Wenn das PsExec License Agreementgeöffnet wird, stimmen Sie Acceptder Lizenz zu und führen Sie den Befehl aus.

> **Note**
>
> Auf Windows-Geräten wird auf dem LocalSystem Konto der Greengrass-Nucleus ausgeführt, und Sie müssen das PsExec Hilfsprogramm verwenden, um die Standardbenutzerinformationen im LocalSystem Konto zu speichern. Wenn Sie die Credential Manager-Anwendung verwenden, werden diese Informationen nicht im Konto, sondern im Windows-Konto des aktuell angemeldeten Benutzers gespeichert. LocalSystem

Laden Sie die AWS IoT Greengrass Core-Software herunter

Sie können die neueste Version der AWS IoT Greengrass Core-Software von der folgenden Adresse herunterladen:

- https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip

> **Note**
>
> Sie können eine bestimmte Version der AWS IoT Greengrass Core-Software von der folgenden Adresse herunterladen. Ersetzen Sie die *Version* durch die Version, die Sie herunterladen möchten.
>
> ```
> https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
> ```

Um die AWS IoT Greengrass Core-Software herunterzuladen

1. Laden Sie die Core-Software auf Ihrem AWS IoT Greengrass Core-Gerät in eine Datei mit dem Namen heruntergreengrass-nucleus-latest.zip.

   Linux or Unix

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip > greengrass-nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip > greengrass-nucleus-latest.zip
   ```

   PowerShell

   ```
   iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip -OutFile greengrass-nucleus-latest.zip
   ```

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

2. (Optional) Um die Greengrass Nucleus-Softwaresignatur zu überprüfen

   > **Note**
   >
   > Diese Funktion ist mit Greengrass Nucleus Version 2.9.5 und höher verfügbar.

a. Verwenden Sie den folgenden Befehl, um die Signatur Ihres Greengrass-Kernartefakts zu überprüfen:

   Linux or Unix

   ```
   jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   Der Dateiname kann je nach installierter JDK-Version anders aussehen. Ersetzen Sie es *jdk17.0.6_10*durch die JDK-Version, die Sie installiert haben.

   ```
   "C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -
   verify -certs -verbose greengrass-nucleus-latest.zip
   ```

   PowerShell

   Der Dateiname sieht je nach installierter JDK-Version möglicherweise anders aus. Ersetzen Sie es *jdk17.0.6_10*durch die JDK-Version, die Sie installiert haben.

   ```
   'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -
   verify -certs -verbose greengrass-nucleus-latest.zip
   ```

b. Der `jarsigner` Aufruf liefert eine Ausgabe, die die Ergebnisse der Überprüfung angibt.

   i. Wenn die Greengrass Nucleus-Zip-Datei signiert ist, enthält die Ausgabe die folgende Anweisung:

      ```
      jar verified.
      ```

   ii. Wenn die Greengrass Nucleus-Zip-Datei nicht signiert ist, enthält die Ausgabe die folgende Anweisung:

      ```
      jar is unsigned.
      ```

c. Wenn Sie die `-certs` Option Jarsigner zusammen mit den `-verbose` Optionen `-verify` und angegeben haben, enthält die Ausgabe auch detaillierte Informationen zum Unterzeichnerzertifikat.

3. Entpacken Sie die AWS IoT Greengrass Core-Software in einen Ordner auf Ihrem Gerät. *GreengrassInstaller*Ersetzen Sie es durch den Ordner, den Sie verwenden möchten.

   Linux or Unix

   ```
   unzip greengrass-nucleus-latest.zip -d GreengrassInstaller && rm greengrass-
   nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir GreengrassInstaller && tar -xf greengrass-nucleus-latest.zip -
   C GreengrassInstaller && del greengrass-nucleus-latest.zip
   ```

   PowerShell

   ```
   Expand-Archive -Path greengrass-nucleus-latest.zip -DestinationPath .\
   \GreengrassInstaller
   rm greengrass-nucleus-latest.zip
   ```

4. (Optional) Führen Sie den folgenden Befehl aus, um die Version der AWS IoT Greengrass Core-Software zu sehen.

   ```
   java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
   ```

> **Important**
>
> Wenn Sie eine Version von Greengrass Nucleus vor v2.4.0 installieren, entfernen Sie diesen Ordner nicht, nachdem Sie die AWS IoT Greengrass Core-Software installiert haben. Die AWS IoT Greengrass Core-Software verwendet die Dateien in diesem Ordner zur Ausführung.
> Wenn Sie die neueste Version der Software heruntergeladen haben, installieren Sie v2.4.0 oder höher, und Sie können diesen Ordner entfernen, nachdem Sie die AWS IoT Greengrass Core-Software installiert haben.

Installieren Sie die Core-Software AWS IoT Greengrass

Führen Sie das Installationsprogramm mit Argumenten aus, die die folgenden Aktionen angeben:

- Installieren Sie die Installation aus einer Teilkonfigurationsdatei, in der angegeben ist, dass die AWS Ressourcen und Zertifikate verwendet werden sollen, die Sie zuvor erstellt haben. Die AWS IoT Greengrass Core-Software verwendet eine Konfigurationsdatei, die die Konfiguration jeder Greengrass-Komponente auf dem Gerät spezifiziert. Das Installationsprogramm erstellt aus der von Ihnen bereitgestellten Teilkonfigurationsdatei eine vollständige Konfigurationsdatei.
- Geben Sie an, dass der `ggc_user` Systembenutzer Softwarekomponenten auf dem Kerngerät ausführen soll. Auf Linux-Geräten gibt dieser Befehl auch an, dass die `ggc_group` Systemgruppe verwendet werden soll, und das Installationsprogramm erstellt den Systembenutzer und die Systemgruppe für Sie.
- Richten Sie die AWS IoT Greengrass Core-Software als Systemdienst ein, der beim Booten ausgeführt wird. Auf Linux-Geräten erfordert dies das Systemd-Init-System.

> **Important**
>
> Auf Windows Core-Geräten müssen Sie die AWS IoT Greengrass Core-Software als Systemdienst einrichten.

Weitere Hinweise zu den Argumenten, die Sie angeben können, finden Sie unterInstaller-Argumente.

> **Note**
>
> Wenn Sie AWS IoT Greengrass auf einem Gerät mit begrenztem Arbeitsspeicher arbeiten, können Sie die Speichermenge steuern, die die AWS IoT Greengrass Core-Software verwendet. Um die Speicherzuweisung zu steuern, können Sie im `jvmOptions` Konfigurationsparameter Ihrer Nucleus-Komponente die Optionen für die JVM-Heap-Größe festlegen. Weitere Informationen finden Sie unter Steuern Sie die Speicherzuweisung mit JVM-Optionen.

- Wenn Sie das Ding-Zertifikat und den privaten Schlüssel zuvor im AWS IoT Dienst erstellt haben, folgen Sie den Schritten, um die AWS IoT Greengrass Core-Software mit privaten Schlüssel- und Zertifikatsdateien zu installieren.
- Wenn Sie das Ding-Zertifikat zuvor mit einem privaten Schlüssel in einem Hardware-Sicherheitsmodul (HSM) erstellt haben, folgen Sie den Schritten, um die AWS IoT Greengrass Core-Software mit dem privaten Schlüssel und dem Zertifikat in einem HSM zu installieren.

Installieren Sie die AWS IoT Greengrass Core-Software mit privaten Schlüssel- und Zertifikatsdateien

Um die AWS IoT Greengrass Core-Software zu installieren

1. Überprüfen Sie die Version der AWS IoT Greengrass Core-Software.

   - *GreengrassInstaller*Ersetzen Sie durch den Pfad zu dem Ordner, der die Software enthält.

   ```
   java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
   ```

2. Verwenden Sie einen Texteditor, um eine Konfigurationsdatei mit dem Namen `config.yaml` zu erstellen, die dem Installationsprogramm zur Verfügung gestellt werden soll.

   Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um die Datei mit GNU Nano zu erstellen.

   ```
   nano GreengrassInstaller/config.yaml
   ```

   Kopieren Sie den folgenden YAML-Inhalt in die Datei. Diese Teilkonfigurationsdatei spezifiziert Systemparameter und Greengrass-Nukleus-Parameter.

   ```
   ---
   system:
     certificateFilePath: "/greengrass/v2/device.pem.crt"
     privateKeyPath: "/greengrass/v2/private.pem.key"
     rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
     rootpath: "/greengrass/v2"
     thingName: "MyGreengrassCore"
   services:
     aws.greengrass.Nucleus:
       componentType: "NUCLEUS"
       version: "2.12.6"
       configuration:
         awsRegion: "us-west-2"
         iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
         iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
         iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
   west-2.amazonaws.com"
   ```

Führen Sie dann die folgenden Schritte aus:

- Ersetzen Sie jede Instanz von */greengrass/v2* durch den Greengrass-Stammordner.
- Ersetzen Sie *MyGreengrassCore* durch den Namen der AWS IoT Sache.
- Ersetzen Sie *2.12.6* durch die Version der AWS IoT Greengrass Core-Software.
- Ersetzen Sie *us-west-2* durch den Ort, AWS-Region an dem Sie die Ressourcen erstellt haben.
- Ersetzen Sie es *GreengrassCoreTokenExchangeRoleAlias*durch den Namen des Alias der Token-Exchange-Rolle.
- Ersetzen Sie den `iotDataEndpoint` durch Ihren AWS IoT Datenendpunkt.
- Ersetzen Sie den Endpunkt `iotCredEndpoint` durch Ihren AWS IoT Anmeldeinformationen-Endpunkt.

> **Note**
>
> In dieser Konfigurationsdatei können Sie andere Nucleus-Konfigurationsoptionen wie die zu verwendenden Ports und den Netzwerk-Proxy anpassen, wie im folgenden Beispiel gezeigt. Weitere Informationen finden Sie unter Greengrass Nucleus-Konfiguration.

```
---
system:
  certificateFilePath: "/greengrass/v2/device.pem.crt"
  privateKeyPath: "/greengrass/v2/private.pem.key"
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
  rootpath: "/greengrass/v2"
  thingName: "MyGreengrassCore"
services:
  aws.greengrass.Nucleus:
    componentType: "NUCLEUS"
    version: "2.12.6"
    configuration:
      awsRegion: "us-west-2"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      mqtt:
        port: 443
```

```
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "https://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
```

3. Führen Sie das Installationsprogramm aus und geben Sie `--init-config` an, dass Sie die Konfigurationsdatei bereitstellen möchten.

   - Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Greengrass-Stammordner.
   - Ersetzen Sie jede Instanz von *GreengrassInstaller*durch den Ordner, in den Sie das Installationsprogramm entpackt haben.

   Linux or Unix

   ```
   sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
     -jar ./GreengrassInstaller/lib/Greengrass.jar \
     --init-config ./GreengrassInstaller/config.yaml \
     --component-default-user ggc_user:ggc_group \
     --setup-system-service true
   ```

   Windows Command Prompt (CMD)

   ```
   java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
     -jar ./GreengrassInstaller/lib/Greengrass.jar ^
     --init-config ./GreengrassInstaller/config.yaml ^
     --component-default-user ggc_user ^
     --setup-system-service true
   ```

   PowerShell

   ```
   java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
     -jar ./GreengrassInstaller/lib/Greengrass.jar `
     --init-config ./GreengrassInstaller/config.yaml `
     --component-default-user ggc_user `
     --setup-system-service true
   ```

> **Important**
>
> Auf Windows Core-Geräten müssen Sie angeben, `--setup-system-service true` dass die AWS IoT Greengrass Core-Software als Systemdienst eingerichtet werden soll.

Wenn Sie dies angeben`--setup-system-service true`, gibt das Installationsprogramm aus, `Successfully set up Nucleus as a system service` ob es die Software als Systemdienst eingerichtet und ausgeführt hat. Andernfalls gibt das Installationsprogramm keine Meldung aus, wenn es die Software erfolgreich installiert hat.

> **Note**
>
> Sie können das `deploy-dev-tools` Argument nicht verwenden, um lokale Entwicklungstools bereitzustellen, wenn Sie das Installationsprogramm ohne das `--provision true` Argument ausführen. Informationen zur direkten Bereitstellung der Greengrass-CLI auf Ihrem Gerät finden Sie unterGreengrass-Befehlszeilenschnittstelle.

4. Überprüfen Sie die Installation, indem Sie sich die Dateien im Stammordner ansehen.

   Linux or Unix

   ```
   ls /greengrass/v2
   ```

   Windows Command Prompt (CMD)

   ```
   dir C:\greengrass\v2
   ```

   PowerShell

   ```
   ls C:\greengrass\v2
   ```

   Wenn die Installation erfolgreich war, enthält der Stammordner mehrere Ordner, z. B. `configpackages`, und`logs`.

Installieren Sie die AWS IoT Greengrass Core-Software mit dem privaten Schlüssel und dem Zertifikat in einem HSM

> **Note**
>
> Diese Funktion ist für Version 2.5.3 und höher der Greengrass Nucleus-Komponente verfügbar. AWS IoT Greengrass unterstützt diese Funktion derzeit nicht auf Windows Core-Geräten.

Um die AWS IoT Greengrass Core-Software zu installieren

1. Überprüfen Sie die Version der AWS IoT Greengrass Core-Software.

   - *GreengrassInstaller*Ersetzen Sie durch den Pfad zu dem Ordner, der die Software enthält.

   ```
   java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
   ```

2. Damit die AWS IoT Greengrass Core-Software den privaten Schlüssel und das Zertifikat im HSM verwenden kann, installieren Sie bei der Installation der AWS IoT Greengrass Core-Software die PKCS #11 -Anbieterkomponente. Die PKCS #11 -Provider-Komponente ist ein Plugin, das Sie während der Installation konfigurieren können. Sie können die neueste Version der PKCS #11 -Provider-Komponente von der folgenden Adresse herunterladen:

   - https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/aws.greengrass.crypto.Pkcs11Provider-latest.jar

   Laden Sie das PKCS #11 -Anbieter-Plugin in eine Datei mit dem Namen `aws.greengrass.crypto.Pkcs11Provider.jar` herunter. *GreengrassInstaller*Ersetzen Sie es durch den Ordner, den Sie verwenden möchten.

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/
   aws.greengrass.crypto.Pkcs11Provider-latest.jar > GreengrassInstaller/
   aws.greengrass.crypto.Pkcs11Provider.jar
   ```

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

3. Verwenden Sie einen Texteditor, um eine Konfigurationsdatei mit dem Namen `config.yaml` zu erstellen, die dem Installationsprogramm zur Verfügung gestellt werden soll.

   Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um die Datei mit GNU Nano zu erstellen.

   ```
   nano GreengrassInstaller/config.yaml
   ```

   Kopieren Sie den folgenden YAML-Inhalt in die Datei. Diese Teilkonfigurationsdatei spezifiziert Systemparameter, Greengrass-Nukleus-Parameter und PKCS #11 -Anbieterparameter.

   ```
   ---
   system:
     certificateFilePath: "pkcs11:object=iotdevicekey;type=cert"
     privateKeyPath: "pkcs11:object=iotdevicekey;type=private"
     rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
     rootpath: "/greengrass/v2"
     thingName: "MyGreengrassCore"
   services:
     aws.greengrass.Nucleus:
       componentType: "NUCLEUS"
       version: "2.12.6"
       configuration:
         awsRegion: "us-west-2"
         iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
         iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
         iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
   west-2.amazonaws.com"
     aws.greengrass.crypto.Pkcs11Provider:
       configuration:
         name: "softhsm_pkcs11"
         library: "/usr/local/Cellar/softhsm/2.6.1/lib/softhsm/libsofthsm2.so"
         slot: 1
         userPin: "1234"
   ```

   Führen Sie dann die folgenden Schritte aus:

   - Ersetzen Sie jede Instanz von *iotdevicekey* in den PKCS #11 -URIs durch die Objektbezeichnung, in der Sie den privaten Schlüssel erstellt und das Zertifikat importiert haben.
   - Ersetzen Sie jede Instanz von */greengrass/v2* durch den Greengrass-Stammordner.

- Ersetzen Sie *MyGreengrassCore* durch den Namen der AWS IoT Sache.
- Ersetzen Sie *2.12.6* durch die Version der AWS IoT Greengrass Core-Software.
- Ersetzen Sie *us-west-2* durch den Ort, AWS-Region an dem Sie die Ressourcen erstellt haben.
- Ersetzen Sie es *GreengrassCoreTokenExchangeRoleAlias*durch den Namen des Alias der Token-Exchange-Rolle.
- Ersetzen Sie den `iotDataEndpoint` durch Ihren AWS IoT Datenendpunkt.
- Ersetzen Sie den Endpunkt `iotCredEndpoint` durch Ihren AWS IoT Anmeldeinformationen-Endpunkt.
- Ersetzen Sie die Konfigurationsparameter für die `aws.greengrass.crypto.Pkcs11Provider` Komponente durch die Werte für die HSM-Konfiguration auf dem Kerngerät.

> **Note**
>
> In dieser Konfigurationsdatei können Sie andere Nucleus-Konfigurationsoptionen wie die zu verwendenden Ports und den Netzwerk-Proxy anpassen, wie im folgenden Beispiel gezeigt. Weitere Informationen finden Sie unter Greengrass Nucleus-Konfiguration.

```
---
system:
  certificateFilePath: "pkcs11:object=iotdevicekey;type=cert"
  privateKeyPath: "pkcs11:object=iotdevicekey;type=private"
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
  rootpath: "/greengrass/v2"
  thingName: "MyGreengrassCore"
services:
  aws.greengrass.Nucleus:
    componentType: "NUCLEUS"
    version: "2.12.6"
    configuration:
      awsRegion: "us-west-2"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
      mqtt:
        port: 443
```

```
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "https://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
  aws.greengrass.crypto.Pkcs11Provider:
    configuration:
      name: "softhsm_pkcs11"
      library: "/usr/local/Cellar/softhsm/2.6.1/lib/softhsm/libsofthsm2.so"
      slot: 1
      userPin: "1234"
```

4. Führen Sie das Installationsprogramm aus und geben Sie `--init-config` an, dass Sie die Konfigurationsdatei bereitstellen möchten.

   - */greengrass/v2*Ersetzen Sie es durch den Greengrass-Stammordner.
   - Ersetzen Sie jede Instanz von *GreengrassInstaller*durch den Ordner, in den Sie das Installationsprogramm entpackt haben.

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
  -jar ./GreengrassInstaller/lib/Greengrass.jar \
  --trusted-plugin ./GreengrassInstaller/aws.greengrass.crypto.Pkcs11Provider.jar \
  --init-config ./GreengrassInstaller/config.yaml \
  --component-default-user ggc_user:ggc_group \
  --setup-system-service true
```

> **Important**
>
> Auf Windows Core-Geräten müssen Sie angeben, `--setup-system-service true` dass die AWS IoT Greengrass Core-Software als Systemdienst eingerichtet werden soll.

Wenn Sie dies angeben`--setup-system-service true`, gibt das Installationsprogramm aus, `Successfully set up Nucleus as a system service` ob es die Software als Systemdienst eingerichtet und ausgeführt hat. Andernfalls gibt das Installationsprogramm keine Meldung aus, wenn es die Software erfolgreich installiert hat.

> **Note**
>
> Sie können das `deploy-dev-tools` Argument nicht verwenden, um lokale Entwicklungstools bereitzustellen, wenn Sie das Installationsprogramm ohne das `--provision true` Argument ausführen. Informationen zur direkten Bereitstellung der Greengrass-CLI auf Ihrem Gerät finden Sie unterGreengrass-Befehlszeilenschnittstelle.

5. Überprüfen Sie die Installation, indem Sie sich die Dateien im Stammordner ansehen.

   Linux or Unix

   ```
   ls /greengrass/v2
   ```

   Windows Command Prompt (CMD)

   ```
   dir C:\greengrass\v2
   ```

   PowerShell

   ```
   ls C:\greengrass\v2
   ```

   Wenn die Installation erfolgreich war, enthält der Stammordner mehrere Ordner, z. B. `configpackages`, und`logs`.

Wenn Sie die AWS IoT Greengrass Core-Software als Systemdienst installiert haben, führt das Installationsprogramm die Software für Sie aus. Andernfalls müssen Sie die Software manuell ausführen. Weitere Informationen finden Sie unter Ausführen der AWS IoT Greengrass -Core-Software.

Weitere Informationen zur Konfiguration und Verwendung der Software finden Sie unter: AWS IoT Greengrass

- Konfigurieren Sie die AWS IoT Greengrass Core-Software
- Entwickeln von AWS IoT Greengrass Komponenten
- Bereitstellen von AWS IoT Greengrass Komponenten auf Geräten
- Greengrass-Befehlszeilenschnittstelle

Installieren Sie die AWS IoT Greengrass Core-Software mit AWS IoT Flottenbereitstellung

Diese Funktion ist für Version 2.4.0 und höher der Greengrass Nucleus-Komponente verfügbar.

Mit AWS IoT Fleet Provisioning können Sie konfigurieren, AWS IoT dass X.509-Gerätezertifikate und private Schlüssel generiert und sicher an Ihre Geräte gesendet werden, wenn diese zum ersten Mal eine Verbindung herstellen. AWS IoT AWS IoT stellt Client-Zertifikate bereit, die von der Amazon Root Certificate Authority (CA) signiert wurden. Sie können auch so konfigurieren AWS IoT , dass Dinggruppen, Dingtypen und Berechtigungen für Greengrass-Kerngeräte angegeben werden, die Sie mit Fleet Provisioning bereitstellen. Sie definieren eine Bereitstellungsvorlage, um zu definieren, wie jedes Gerät bereitgestellt wird AWS IoT . In der Bereitstellungsvorlage werden die Ressourcen für Dinge, Richtlinien und Zertifikate angegeben, die bei der Bereitstellung für ein Gerät erstellt werden sollen. Weitere Informationen finden Sie unter Bereitstellungsvorlagen im Entwicklerhandbuch.AWS IoT Core

AWS IoT Greengrass bietet ein AWS IoT Flottenbereitstellungs-Plugin, mit dem Sie die AWS IoT Greengrass Core-Software mithilfe von AWS Ressourcen installieren können, die durch AWS IoT Fleet Provisioning erstellt wurden. Das Fleet Provisioning Plugin verwendet die Bereitstellung nach Anspruch. Geräte verwenden ein Provisioning-Anspruchszertifikat und einen privaten Schlüssel, um ein eindeutiges X.509-Gerätezertifikat und einen privaten Schlüssel zu erhalten, die sie für den regulären Betrieb verwenden können. Sie können das Antragszertifikat und den privaten Schlüssel während der Herstellung in jedes Gerät einbetten, sodass Ihre Kunden die Geräte später aktivieren können, wenn jedes Gerät online ist. Sie können dasselbe Antragszertifikat und denselben privaten Schlüssel für mehrere Geräte verwenden. Weitere Informationen finden Sie unter Provisioning by Claim im AWS IoT Core Developer Guide.

> **Note**
>
> Das Fleet Provisioning-Plugin unterstützt derzeit nicht das Speichern von privaten Schlüssel- und Zertifikatsdateien in einem Hardware-Sicherheitsmodul (HSM). Um ein HSM zu verwenden, installieren Sie die AWS IoT Greengrass Core-Software mit manueller Bereitstellung.

Um die AWS IoT Greengrass Core-Software mit AWS IoT Flottenbereitstellung zu installieren, müssen Sie Ressourcen in Ihrem einrichten, die für AWS-Konto die Bereitstellung von Greengrass-Core-Geräten AWS IoT verwendet werden. Zu diesen Ressourcen gehören eine

Bereitstellungsvorlage, Antragszertifikate und eine IAM-Rolle für den Tokenaustausch. Nachdem Sie diese Ressourcen erstellt haben, können Sie sie wiederverwenden, um mehrere Kerngeräte in einer Flotte bereitzustellen. Weitere Informationen finden Sie unter Einrichten der AWS IoT Flottenbereitstellung für Greengrass-Core-Geräte.

> **Important**
>
> Bevor Sie die AWS IoT Greengrass Core-Software herunterladen, überprüfen Sie, ob Ihr Kerngerät die Anforderungen für die Installation und Ausführung der AWS IoT Greengrass Core-Software v2.0 erfüllt.

Themen

Voraussetzungen

Um die AWS IoT Greengrass Core-Software mit AWS IoT Flottenbereitstellung zu installieren, müssen Sie zunächst die AWS IoT Flottenbereitstellung für Greengrass-Kerngeräte einrichten. Nachdem Sie diese Schritte einmal abgeschlossen haben, können Sie Fleet Provisioning verwenden, um die AWS IoT Greengrass Core-Software auf einer beliebigen Anzahl von Geräten zu installieren.

Endpunkte abrufen AWS IoT

Holen Sie sich die AWS IoT Endpunkte für Sie und speichern Sie sie AWS-Konto, um sie später zu verwenden. Ihr Gerät verwendet diese Endpunkte, um eine Verbindung herzustellen. AWS IoT Gehen Sie wie folgt vor:

1. Holen Sie sich den AWS IoT Datenendpunkt für Ihren AWS-Konto.

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

   Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anfrage erfolgreich ist.

   ```
   {
     "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
   }
   ```

2. Rufen Sie den Endpunkt der AWS IoT Anmeldeinformationen für Ihren AWS-Konto ab.

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

   Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anfrage erfolgreich ist.

   ```
   {
     "endpointAddress": "device-credentials-prefix.credentials.iot.us-
   west-2.amazonaws.com"
   }
   ```

Laden Sie Zertifikate auf das Gerät herunter

Das Gerät verwendet ein Anspruchszertifikat und einen privaten Schlüssel, um seine Anfrage zur Bereitstellung von AWS Ressourcen zu authentifizieren und ein X.509-Gerätezertifikat zu erwerben. Sie können das Antragszertifikat und den privaten Schlüssel bei der Herstellung in das Gerät einbetten oder das Zertifikat und den Schlüssel bei der Installation auf das Gerät kopieren. In diesem Abschnitt kopieren Sie das Antragszertifikat und den privaten Schlüssel auf das Gerät. Sie laden auch das Zertifikat der Amazon Root Certificate Authority (CA) auf das Gerät herunter.

> **Important**
>
> Private Schlüssel für Provisioning Claim sollten jederzeit gesichert sein, auch auf Greengrass-Core-Geräten. Wir empfehlen Ihnen, anhand von CloudWatch Kennzahlen und Protokollen von Amazon nach Hinweisen auf Missbrauch zu suchen, wie z. B. die unbefugte Verwendung des Antragszertifikats zur Bereitstellung von Geräten. Wenn Sie einen Missbrauch feststellen, deaktivieren Sie das Provisioning Claim Certificate, sodass

es nicht für die Bereitstellung von Geräten verwendet werden kann. Weitere Informationen finden Sie unter Überwachung AWS IoT im AWS IoT Core Entwicklerhandbuch.
Damit Sie die Anzahl der Geräte und die Geräte, die sich bei Ihnen registrieren, besser verwalten können, können Sie AWS-Konto bei der Erstellung einer Flottenbereitstellungsvorlage einen Pre-Provisioning-Hook angeben. Ein Pre-Provisioning-Hook ist eine AWS Lambda Funktion, die Vorlagenparameter validiert, die Geräte bei der Registrierung angeben. Sie können beispielsweise einen Pre-Provisioning-Hook erstellen, der eine Geräte-ID mit einer Datenbank vergleicht, um sicherzustellen, dass das Gerät über eine Bereitstellungsberechtigung verfügt. Weitere Informationen finden Sie unter Pre-Provisioning Hooks im Developer Guide.AWS IoT Core

So laden Sie Anspruchszertifikate auf das Gerät herunter

1. Kopieren Sie das Antragszertifikat und den privaten Schlüssel auf das Gerät. Wenn SSH und SCP auf dem Entwicklungscomputer und dem Gerät aktiviert sind, können Sie den `scp` Befehl auf Ihrem Entwicklungscomputer verwenden, um das Anspruchszertifikat und den privaten Schlüssel zu übertragen. Mit dem folgenden Beispielbefehl werden diese Dateien in einem Ordner mit dem Namen `claim-certs` auf Ihrem Entwicklungscomputer auf das Gerät übertragen. Ersetzen Sie die *`Geräte-IP-Adresse`* durch die IP-Adresse Ihres Geräts.

   ```
   scp -r claim-certs/ device-ip-address:~
   ```

2. Erstellen Sie den Greengrass-Stammordner auf dem Gerät. Sie werden später die AWS IoT Greengrass Core-Software in diesem Ordner installieren.

   Linux or Unix

   - Ersetzen Sie es */greengrass/v2* durch den Ordner, den Sie verwenden möchten.

   ```
   sudo mkdir -p /greengrass/v2
   ```

   Windows Command Prompt

   - Ersetzen Sie *C:\greengrass\v2* durch den zu verwendenden Ordner.

   ```
   mkdir C:\greengrass\v2
   ```

PowerShell

- Ersetzen Sie *C:\greengrass\v2* durch den Ordner, den Sie verwenden möchten.

```
mkdir C:\greengrass\v2
```

3. (Nur Linux) Legen Sie die Berechtigungen des übergeordneten Elements des Greengrass-Stammordners fest.

   - Ersetzen Sie */greengrass* durch das übergeordnete Objekt des Stammordners.

   ```
   sudo chmod 755 /greengrass
   ```

4. Verschieben Sie die Anspruchszertifikate in den Greengrass-Stammordner.

   - Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Greengrass-Stammordner.

   Linux or Unix

   ```
   sudo mv ~/claim-certs /greengrass/v2
   ```

   Windows Command Prompt (CMD)

   ```
   move %USERPROFILE%\claim-certs C:\greengrass\v2
   ```

   PowerShell

   ```
   mv -Path ~\claim-certs -Destination C:\greengrass\v2
   ```

5. Laden Sie das Zertifikat der Amazon Root Certificate Authority (CA) herunter. AWS IoT Zertifikate sind standardmäßig mit dem Root-CA-Zertifikat von Amazon verknüpft.

   Linux or Unix

   ```
   sudo curl -o /greengrass/v2/AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   ```

Windows Command Prompt (CMD)

```
curl -o C:\greengrass\v2\\AmazonRootCA1.pem https://www.amazontrust.com/
repository/AmazonRootCA1.pem
```

PowerShell

```
iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile C:
\greengrass\v2\\AmazonRootCA1.pem
```

Richten Sie die Geräteumgebung ein

Folgen Sie den Schritten in diesem Abschnitt, um ein Linux- oder Windows-Gerät einzurichten, das als Ihr AWS IoT Greengrass Kerngerät verwendet werden soll.

Richten Sie ein Linux-Gerät ein

Um ein Linux-Gerät einzurichten für AWS IoT Greengrass V2

1. Installieren Sie die Java-Runtime, die für die Ausführung der AWS IoT Greengrass Core-Software erforderlich ist. Wir empfehlen, dass Sie die Langzeit-Support-Versionen von Amazon Corretto oder OpenJDK verwenden. Version 8 oder höher ist erforderlich. Die folgenden Befehle zeigen Ihnen, wie Sie OpenJDK auf Ihrem Gerät installieren.

   - Für Debian- oder Ubuntu-basierte Distributionen:

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

   - Für Red Hat-basierte Distributionen:

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

   - Für Amazon Linux 2:

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

   - Für Amazon Linux 2023:

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

Führen Sie nach Abschluss der Installation den folgenden Befehl aus, um zu überprüfen, ob Java auf Ihrem Linux-Gerät ausgeführt wird.

```
java -version
```

Der Befehl druckt die Version von Java, die auf dem Gerät ausgeführt wird. Bei einer Debian-basierten Distribution könnte die Ausgabe beispielsweise dem folgenden Beispiel ähneln.

```
openjdk version "11.0.9.1" 2020-11-04
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
```

2. (Optional) Erstellen Sie den Standardsystembenutzer und die Standardgruppe, die Komponenten auf dem Gerät ausführen. Sie können auch festlegen, dass der AWS IoT Greengrass Core-Software-Installer diesen Benutzer und diese Gruppe während der Installation mit dem `--component-default-user` Installer-Argument erstellt. Weitere Informationen finden Sie unter Installer-Argumente.

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

3. Stellen Sie sicher, dass der Benutzer, der die AWS IoT Greengrass Core-Software ausführt (in der Regel`root`), über die erforderlichen Rechte verfügt, die Software `sudo` mit jedem beliebigen Benutzer und jeder Gruppe auszuführen.

   a. Führen Sie den folgenden Befehl aus, um die `/etc/sudoers` Datei zu öffnen.

      ```
      sudo visudo
      ```

   b. Stellen Sie sicher, dass die Berechtigung für den Benutzer wie im folgenden Beispiel aussieht.

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

4. (Optional) Um containerisierte Lambda-Funktionen auszuführen, müssen Sie cgroups v1 aktivieren und Sie müssen die Speicher - und Geräte-Cgroups aktivieren und mounten. Wenn Sie nicht vorhaben, containerisierte Lambda-Funktionen auszuführen, können Sie diesen Schritt überspringen.

Um diese Cgroups-Optionen zu aktivieren, starten Sie das Gerät mit den folgenden Linux-Kernelparametern.

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

Informationen zum Anzeigen und Einstellen der Kernel-Parameter für Ihr Gerät finden Sie in der Dokumentation zu Ihrem Betriebssystem und Bootloader. Folgen Sie den Anweisungen, um die Kernel-Parameter dauerhaft einzustellen.

5. Installieren Sie alle anderen erforderlichen Abhängigkeiten auf Ihrem Gerät, wie in der Liste der Anforderungen unter angegebenAnforderungen an Speichergeräte.

Richten Sie ein Windows-Gerät ein

> **Note**
>
> Diese Funktion ist für Version 2.5.0 und höher der Greengrass Nucleus-Komponente verfügbar.

Um ein Windows-Gerät einzurichten für AWS IoT Greengrass V2

1. Installieren Sie die Java-Runtime, die für die Ausführung der AWS IoT Greengrass Core-Software erforderlich ist. Wir empfehlen, dass Sie die Langzeit-Support-Versionen von Amazon Corretto oder OpenJDK verwenden. Version 8 oder höher ist erforderlich.
2. Prüfen Sie, ob Java in der Systemvariablen PATH verfügbar ist, und fügen Sie es hinzu, falls nicht. Auf dem LocalSystem Konto wird die AWS IoT Greengrass Core-Software ausgeführt, sodass Sie der Systemvariablen PATH statt der Benutzervariablen PATH für Ihren Benutzer Java hinzufügen müssen. Gehen Sie wie folgt vor:

   a. Drücken Sie die Windows-Taste, um das Startmenü zu öffnen.
   b. Geben Sie **`environment variables`** ein, um im Startmenü nach den Systemoptionen zu suchen.
   c. Wählen Sie in den Suchergebnissen des Startmenüs die Option Systemumgebungsvariablen bearbeiten aus, um das Fenster mit den Systemeigenschaften zu öffnen.
   d. Wählen Sie Umgebungsvariablen... um das Fenster mit den Umgebungsvariablen zu öffnen.

e. Wählen Sie unter Systemvariablen die Option Pfad und dann Bearbeiten aus. Im Fenster Umgebungsvariable bearbeiten können Sie jeden Pfad in einer separaten Zeile anzeigen.

f. Überprüfen Sie, ob der Pfad zum `bin` Ordner der Java-Installation vorhanden ist. Der Pfad könnte dem folgenden Beispiel ähneln.

```
C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
```

g. Wenn der `bin` Ordner der Java-Installation in Path fehlt, wählen Sie Neu, um ihn hinzuzufügen, und klicken Sie dann auf OK.

3. Öffnen Sie die Windows-Eingabeaufforderung (`cmd.exe`) als Administrator.

4. Erstellen Sie den Standardbenutzer für das LocalSystem Konto auf dem Windows-Gerät. Ersetzen Sie *das Passwort* durch ein sicheres Passwort.

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

> **Tip**
>
> Abhängig von Ihrer Windows-Konfiguration ist das Benutzerkennwort möglicherweise so eingestellt, dass es an einem Datum in der future abläuft. Um sicherzustellen, dass Ihre Greengrass-Anwendungen weiterhin funktionieren, verfolgen Sie, wann das Passwort abläuft, und aktualisieren Sie es, bevor es abläuft. Sie können das Benutzerkennwort auch so einrichten, dass es niemals abläuft.
>
> - Führen Sie den folgenden Befehl aus, um zu überprüfen, wann ein Benutzer und sein Passwort ablaufen.
>
> ```
> net user ggc_user | findstr /C:expires
> ```
>
> - Führen Sie den folgenden Befehl aus, um das Passwort eines Benutzers so einzustellen, dass es nie abläuft.
>
> ```
> wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
> ```
>
> - Wenn Sie Windows 10 oder höher verwenden und der `wmic` Befehl veraltet ist, führen Sie den folgenden PowerShell Befehl aus.

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

5. Laden Sie das PsExecProgramm von Microsoft herunter und installieren Sie es auf dem Gerät.
6. Verwenden Sie das PsExec Hilfsprogramm, um den Benutzernamen und das Passwort für den Standardbenutzer in der Credential Manager-Instanz für das LocalSystem Konto zu speichern. Ersetzen Sie *das Passwort* durch das Passwort des Benutzers, das Sie zuvor festgelegt haben.

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

   Wenn das PsExec License Agreementgeöffnet wird, stimmen Sie Acceptder Lizenz zu und führen Sie den Befehl aus.

   > **Note**
   >
   > Auf Windows-Geräten wird auf dem LocalSystem Konto der Greengrass-Nucleus ausgeführt, und Sie müssen das PsExec Hilfsprogramm verwenden, um die Standardbenutzerinformationen im LocalSystem Konto zu speichern. Wenn Sie die Credential Manager-Anwendung verwenden, werden diese Informationen nicht im Konto, sondern im Windows-Konto des aktuell angemeldeten Benutzers gespeichert. LocalSystem

Laden Sie die AWS IoT Greengrass Core-Software herunter

Sie können die neueste Version der AWS IoT Greengrass Core-Software von der folgenden Adresse herunterladen:

- https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip

> **Note**
>
> Sie können eine bestimmte Version der AWS IoT Greengrass Core-Software von der folgenden Adresse herunterladen. Ersetzen Sie die *Version* durch die Version, die Sie herunterladen möchten.

```
https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
```

Um die AWS IoT Greengrass Core-Software herunterzuladen

1. Laden Sie die Core-Software auf Ihrem AWS IoT Greengrass Core-Gerät in eine Datei mit dem Namen herunter`greengrass-nucleus-latest.zip`.

   Linux or Unix

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip > greengrass-nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip > greengrass-nucleus-latest.zip
   ```

   PowerShell

   ```
   iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip -OutFile greengrass-nucleus-latest.zip
   ```

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

2. (Optional) Um die Greengrass Nucleus-Softwaresignatur zu überprüfen

   > **Note**
   >
   > Diese Funktion ist mit Greengrass Nucleus Version 2.9.5 und höher verfügbar.

   a. Verwenden Sie den folgenden Befehl, um die Signatur Ihres Greengrass-Kernartefakts zu überprüfen:

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

Der Dateiname kann je nach installierter JDK-Version anders aussehen. Ersetzen Sie es *jdk17.0.6_10* durch die JDK-Version, die Sie installiert haben.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

Der Dateiname sieht je nach installierter JDK-Version möglicherweise anders aus. Ersetzen Sie es *jdk17.0.6_10* durch die JDK-Version, die Sie installiert haben.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -
verify -certs -verbose greengrass-nucleus-latest.zip
```

b. Der `jarsigner` Aufruf liefert eine Ausgabe, die die Ergebnisse der Überprüfung angibt.

   i. Wenn die Greengrass Nucleus-Zip-Datei signiert ist, enthält die Ausgabe die folgende Anweisung:

   ```
   jar verified.
   ```

   ii. Wenn die Greengrass Nucleus-Zip-Datei nicht signiert ist, enthält die Ausgabe die folgende Anweisung:

   ```
   jar is unsigned.
   ```

c. Wenn Sie die `-certs` Option Jarsigner zusammen mit den `-verbose` Optionen `-verify` und angegeben haben, enthält die Ausgabe auch detaillierte Informationen zum Unterzeichnerzertifikat.

3. Entpacken Sie die AWS IoT Greengrass Core-Software in einen Ordner auf Ihrem Gerät. *GreengrassInstaller* Ersetzen Sie es durch den Ordner, den Sie verwenden möchten.

Linux or Unix

```
unzip greengrass-nucleus-latest.zip -d GreengrassInstaller && rm greengrass-
nucleus-latest.zip
```

Windows Command Prompt (CMD)

```
mkdir GreengrassInstaller && tar -xf greengrass-nucleus-latest.zip -
C GreengrassInstaller && del greengrass-nucleus-latest.zip
```

PowerShell

```
Expand-Archive -Path greengrass-nucleus-latest.zip -DestinationPath .\
\GreengrassInstaller
rm greengrass-nucleus-latest.zip
```

4. (Optional) Führen Sie den folgenden Befehl aus, um die Version der AWS IoT Greengrass Core-Software zu sehen.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

> **Important**
>
> Wenn Sie eine Version von Greengrass Nucleus vor v2.4.0 installieren, entfernen Sie diesen Ordner nicht, nachdem Sie die AWS IoT Greengrass Core-Software installiert haben. Die AWS IoT Greengrass Core-Software verwendet die Dateien in diesem Ordner zur Ausführung.
> Wenn Sie die neueste Version der Software heruntergeladen haben, installieren Sie v2.4.0 oder höher, und Sie können diesen Ordner entfernen, nachdem Sie die AWS IoT Greengrass Core-Software installiert haben.

Laden Sie das AWS IoT Fleet Provisioning Plugin herunter

Sie können die neueste Version des AWS IoT Fleet Provisioning-Plug-ins von der folgenden Adresse herunterladen:

- https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass - /fleetprovisioningbyclaim-latest.jar FleetProvisioning ByClaim

> **Note**
>
> Sie können eine bestimmte Version des AWS IoT Fleet Provisioning-Plug-ins von der folgenden Adresse herunterladen. Ersetzen Sie die *Version* durch die Version, die heruntergeladen werden soll. Weitere Informationen zu den einzelnen Versionen des Fleet Provisioning-Plug-ins finden Sie unterAWS IoT Changelog des Plug-ins für die Flottenbereitstellung.
>
> ```
> https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-
> FleetProvisioningByClaim/fleetprovisioningbyclaim-version.jar
> ```

Das Fleet Provisioning Plugin ist Open Source. Den Quellcode finden Sie im AWS IoT Fleet Provisioning Plugin unter. GitHub

Um das AWS IoT Fleet Provisioning Plugin herunterzuladen

- Laden Sie auf Ihrem Gerät das AWS IoT Fleet Provisioning-Plugin in eine Datei mit dem Namen herunter. `aws.greengrass.FleetProvisioningByClaim.jar` *GreengrassInstaller*Ersetzen Sie es durch den Ordner, den Sie verwenden möchten.

  Linux or Unix

  ```
  curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-
  FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar
   > GreengrassInstaller/aws.greengrass.FleetProvisioningByClaim.jar
  ```

  Windows Command Prompt (CMD)

  ```
  curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-
  FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar
   > GreengrassInstaller/aws.greengrass.FleetProvisioningByClaim.jar
  ```

PowerShell

```
iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-
FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar -
OutFile GreengrassInstaller/aws.greengrass.FleetProvisioningByClaim.jar
```

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

Installieren Sie die AWS IoT Greengrass Core-Software

Führen Sie das Installationsprogramm mit Argumenten aus, die die folgenden Aktionen angeben:

- Die Installation erfolgt aus einer Teilkonfigurationsdatei, in der angegeben ist, dass das Fleet Provisioning-Plugin zur Bereitstellung von AWS Ressourcen verwendet werden soll. Die AWS IoT Greengrass Core-Software verwendet eine Konfigurationsdatei, die die Konfiguration jeder Greengrass-Komponente auf dem Gerät spezifiziert. Das Installationsprogramm erstellt eine vollständige Konfigurationsdatei aus der Teilkonfigurationsdatei, die Sie bereitstellen, und den AWS Ressourcen, die das Fleet Provisioning-Plugin erstellt.
- Geben Sie an, dass der `ggc_user` Systembenutzer zur Ausführung von Softwarekomponenten auf dem Kerngerät verwendet werden soll. Auf Linux-Geräten gibt dieser Befehl auch an, dass die `ggc_group` Systemgruppe verwendet werden soll, und das Installationsprogramm erstellt den Systembenutzer und die Systemgruppe für Sie.
- Richten Sie die AWS IoT Greengrass Core-Software als Systemdienst ein, der beim Booten ausgeführt wird. Auf Linux-Geräten erfordert dies das Systemd-Init-System.

> **Important**
>
> Auf Windows Core-Geräten müssen Sie die AWS IoT Greengrass Core-Software als Systemdienst einrichten.

Weitere Hinweise zu den Argumenten, die Sie angeben können, finden Sie unterInstaller-Argumente.

> **Note**
>
> Wenn Sie AWS IoT Greengrass auf einem Gerät mit begrenztem Arbeitsspeicher arbeiten, können Sie die Speichermenge steuern, die die AWS IoT Greengrass Core-Software verwendet. Um die Speicherzuweisung zu steuern, können Sie im `jvmOptions` Konfigurationsparameter Ihrer Nucleus-Komponente die Optionen für die JVM-Heap-Größe festlegen. Weitere Informationen finden Sie unter Steuern Sie die Speicherzuweisung mit JVM-Optionen.

Um die Core-Software zu installieren AWS IoT Greengrass

1. Überprüfen Sie die Version der AWS IoT Greengrass Core-Software.

   - *`GreengrassInstaller`*Ersetzen Sie es durch den Pfad zu dem Ordner, der die Software enthält.

   ```
   java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
   ```

2. Verwenden Sie einen Texteditor, um eine Konfigurationsdatei mit dem Namen `config.yaml` zu erstellen, die dem Installationsprogramm zur Verfügung gestellt werden soll.

   Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um die Datei mit GNU Nano zu erstellen.

   ```
   nano GreengrassInstaller/config.yaml
   ```

   Kopieren Sie den folgenden YAML-Inhalt in die Datei. Diese Teilkonfigurationsdatei spezifiziert Parameter für das Fleet Provisioning Plugin. Weitere Informationen zu den Optionen, die Sie angeben können, finden Sie unterKonfigurieren des AWS IoT Flottenbereitstellungs-Plugins.

   Linux or Unix

   ```
   ---
   services:
     aws.greengrass.Nucleus:
       version: "2.12.6"
     aws.greengrass.FleetProvisioningByClaim:
       configuration:
   ```

```
      rootPath: "/greengrass/v2"
      awsRegion: "us-west-2"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredentialEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "/greengrass/v2/claim-certs/claim.pem.crt"
      claimCertificatePrivateKeyPath: "/greengrass/v2/claim-certs/
claim.private.pem.key"
      rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
      templateParameters:
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
```

Windows

```
---
services:
  aws.greengrass.Nucleus:
    version: "2.12.6"
  aws.greengrass.FleetProvisioningByClaim:
    configuration:
      rootPath: "C:\\greengrass\\v2"
      awsRegion: "us-west-2"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredentialEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "C:\\greengrass\\v2\\claim-certs\\claim.pem.crt"
      claimCertificatePrivateKeyPath: "C:\\greengrass\\v2\\claim-certs\
\claim.private.pem.key"
      rootCaPath: "C:\\greengrass\\v2\\AmazonRootCA1.pem"
      templateParameters:
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
```

Führen Sie dann die folgenden Schritte aus:

- Ersetzen Sie *2.12.6* durch die Version der AWS IoT Greengrass Core-Software.

- Ersetzen Sie jede Instanz von */greengrass/v2* oder *C:\greengrass\v2* durch den Greengrass-Stammordner.

  > **Note**
  >
  > Auf Windows-Geräten müssen Sie Pfadtrennzeichen als doppelte umgekehrte Schrägstriche (\\) angeben, z. B. `C:\\greengrass\\v2`

- Ersetzen Sie *us-west-2* durch die AWS Region, in der Sie die Bereitstellungsvorlage und andere Ressourcen erstellt haben.
- Ersetzen Sie das durch Ihren `iotDataEndpoint` Datenendpunkt AWS IoT .
- Ersetzen Sie den Endpunkt `iotCredentialEndpoint` durch Ihren AWS IoT Anmeldeinformationen-Endpunkt.
- *GreengrassCoreTokenExchangeRoleAlias*Ersetzen Sie es durch den Namen des Alias der Token-Exchange-Rolle.
- *GreengrassFleetProvisioningTemplate*Ersetzen Sie es durch den Namen der Vorlage für die Flottenbereitstellung.
- Ersetzen Sie den `claimCertificatePath` durch den Pfad zum Antragszertifikat auf dem Gerät.
- Ersetzen Sie den `claimCertificatePrivateKeyPath` durch den Pfad zum privaten Schlüssel des Anspruchszertifikats auf dem Gerät.
- Ersetzen Sie die Vorlagenparameter (`templateParameters`) durch die Werte, die für die Bereitstellung des Geräts verwendet werden sollen. Dieses Beispiel bezieht sich auf die Beispielvorlage, die `ThingGroupName` Parameter definiert`ThingName`.

> **Note**
>
> In dieser Konfigurationsdatei können Sie andere Konfigurationsoptionen wie die zu verwendenden Ports und den Netzwerk-Proxy anpassen, wie im folgenden Beispiel gezeigt. Weitere Informationen finden Sie unter Greengrass Nucleus-Konfiguration.
>
> Linux or Unix
>
> ```
> ---
> services:
>   aws.greengrass.Nucleus:
> ```

```
    version: "2.12.6"
    configuration:
      mqtt:
        port: 443
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "http://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
  aws.greengrass.FleetProvisioningByClaim:
    configuration:
      rootPath: "/greengrass/v2"
      awsRegion: "us-west-2"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-
west-2.amazonaws.com"
      iotCredentialEndpoint: "device-credentials-
prefix.credentials.iot.us-west-2.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "/greengrass/v2/claim-certs/claim.pem.crt"
      claimCertificatePrivateKeyPath: "/greengrass/v2/claim-certs/
claim.private.pem.key"
      rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
      templateParameters:
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
      mqttPort: 443
      proxyUrl: "http://my-proxy-server:1100"
      proxyUserName: "Mary_Major"
      proxyPassword: "pass@word1357"
```

Windows

```
---
services:
  aws.greengrass.Nucleus:
    version: "2.12.6"
    configuration:
      mqtt:
        port: 443
      greengrassDataPlanePort: 443
```

```
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "http://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
  aws.greengrass.FleetProvisioningByClaim:
    configuration:
      rootPath: "C:\\greengrass\\v2"
      awsRegion: "us-west-2"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-
west-2.amazonaws.com"
      iotCredentialEndpoint: "device-credentials-
prefix.credentials.iot.us-west-2.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "C:\\greengrass\\v2\\claim-certs\
\claim.pem.crt"
      claimCertificatePrivateKeyPath: "C:\\greengrass\\v2\\claim-certs\
\claim.private.pem.key"
      rootCaPath: "C:\\greengrass\\v2\\AmazonRootCA1.pem"
      templateParameters:
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
      mqttPort: 443
      proxyUrl: "http://my-proxy-server:1100"
      proxyUserName: "Mary_Major"
      proxyPassword: "pass@word1357"
```

Um einen HTTPS-Proxy zu verwenden, müssen Sie Version 1.1.0 oder höher des Fleet Provisioning-Plug-ins verwenden. Sie müssen zusätzlich Folgendes angeben `rootCaPathsystem`, wie im folgenden Beispiel gezeigt.

Linux or Unix

```
---
system:
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
services:
  ...
```

Windows

```
---
system:
  rootCaPath: "C:\\greengrass\\v2\\AmazonRootCA1.pem"
services:
  ...
```

3. Führen Sie das Installationsprogramm aus. Geben Sie `--trusted-plugin` an, ob das Fleet Provisioning-Plug-In bereitgestellt werden soll, und geben Sie `--init-config` an, dass die Konfigurationsdatei bereitgestellt werden soll.

   - */greengrass/v2*Ersetzen Sie es durch den Greengrass-Stammordner.
   - Ersetzen Sie jede Instanz von *GreengrassInstaller*durch den Ordner, in den Sie das Installationsprogramm entpackt haben.

   Linux or Unix

   ```
   sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
     -jar ./GreengrassInstaller/lib/Greengrass.jar \
     --trusted-plugin ./GreengrassInstaller/
   aws.greengrass.FleetProvisioningByClaim.jar \
     --init-config ./GreengrassInstaller/config.yaml \
     --component-default-user ggc_user:ggc_group \
     --setup-system-service true
   ```

   Windows Command Prompt (CMD)

   ```
   java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
     -jar ./GreengrassInstaller/lib/Greengrass.jar ^
     --trusted-plugin ./GreengrassInstaller/
   aws.greengrass.FleetProvisioningByClaim.jar ^
     --init-config ./GreengrassInstaller/config.yaml ^
     --component-default-user ggc_user ^
     --setup-system-service true
   ```

   PowerShell

   ```
   java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
   ```

```
  -jar ./GreengrassInstaller/lib/Greengrass.jar `
  --trusted-plugin ./GreengrassInstaller/
aws.greengrass.FleetProvisioningByClaim.jar `
  --init-config ./GreengrassInstaller/config.yaml `
  --component-default-user ggc_user `
  --setup-system-service true
```

> **Important**
>
> Auf Windows Core-Geräten müssen Sie angeben, `--setup-system-service true` dass die AWS IoT Greengrass Core-Software als Systemdienst eingerichtet werden soll.

Wenn Sie dies angeben`--setup-system-service true`, gibt das Installationsprogramm aus, `Successfully set up Nucleus as a system service` ob es die Software als Systemdienst eingerichtet und ausgeführt hat. Andernfalls gibt das Installationsprogramm keine Meldung aus, wenn es die Software erfolgreich installiert hat.

> **Note**
>
> Sie können das `deploy-dev-tools` Argument nicht verwenden, um lokale Entwicklungstools bereitzustellen, wenn Sie das Installationsprogramm ohne das `--provision true` Argument ausführen. Informationen zur direkten Bereitstellung der Greengrass-CLI auf Ihrem Gerät finden Sie unterGreengrass-Befehlszeilenschnittstelle.

4. Überprüfen Sie die Installation, indem Sie sich die Dateien im Stammordner ansehen.

   Linux or Unix

   ```
   ls /greengrass/v2
   ```

   Windows Command Prompt (CMD)

   ```
   dir C:\greengrass\v2
   ```

PowerShell

```
ls C:\greengrass\v2
```

Wenn die Installation erfolgreich war, enthält der Stammordner mehrere Ordner, z. B. `configpackages`, und`logs`.

Wenn Sie die AWS IoT Greengrass Core-Software als Systemdienst installiert haben, führt das Installationsprogramm die Software für Sie aus. Andernfalls müssen Sie die Software manuell ausführen. Weitere Informationen finden Sie unter Ausführen der AWS IoT Greengrass -Core-Software.

Weitere Informationen zur Konfiguration und Verwendung der Software finden Sie unter: AWS IoT Greengrass

- Konfigurieren Sie die AWS IoT Greengrass Core-Software
- Entwickeln von AWS IoT Greengrass Komponenten
- Bereitstellen von AWS IoT Greengrass Komponenten auf Geräten
- Greengrass-Befehlszeilenschnittstelle

Einrichten der AWS IoT Flottenbereitstellung für Greengrass-Core-Geräte

Um die -AWS IoT GreengrassCore-Software mit Flottenbereitstellung zu installieren, müssen Sie zunächst die folgenden Ressourcen in Ihrem einrichtenAWS-Konto. Diese Ressourcen ermöglichen es Geräten, sich bei Greengrass-Core-Geräten zu registrieren AWS IoT und als Greengrass-Core-Geräte zu arbeiten. Führen Sie die Schritte in diesem Abschnitt einmal aus, um diese Ressourcen in Ihrem zu erstellen und zu konfigurierenAWS-Konto.

- Eine IAM-Rolle für den Tokenaustausch, die -Core-Geräte verwenden, um Aufrufe von -AWSServices zu autorisieren.
- Ein -AWS IoTRollenalias, der auf die Token-Austauschrolle verweist.
- (Optional) Eine -AWS IoTRichtlinie, mit der die -Core-Geräte Aufrufe der - AWS IoT und -AWS IoT GreengrassServices autorisieren. Diese AWS IoT Richtlinie muss die `-iot:AssumeRoleWithCertificate`Berechtigung für den AWS IoT Rollenalias zulassen, der auf die Token-Exchange-Rolle verweist.

Sie können eine einzelne AWS IoT Richtlinie für alle Core-Geräte in Ihrer Flotte verwenden oder Ihre Flottenbereitstellungsvorlage so konfigurieren, dass eine -AWS IoTRichtlinie für jedes Core-Gerät erstellt wird.

- Eine AWS IoT Flottenbereitstellungsvorlage. Diese Vorlage muss Folgendes angeben:
  - Eine -AWS IoTObjektressource. Sie können eine Liste vorhandener Objektgruppen angeben, um Komponenten auf jedem Gerät bereitzustellen, wenn es online geht.
  - Eine -AWS IoTRichtlinienressource. Diese Ressource kann eine der folgenden Eigenschaften definieren:
    - Der Name einer vorhandenen AWS IoT Richtlinie. Wenn Sie diese Option wählen, verwenden die Core-Geräte, die Sie aus dieser Vorlage erstellen, dieselbe AWS IoT Richtlinie, und Sie können ihre Berechtigungen als Flotte verwalten.
    - Ein -AWS IoTRichtliniendokument. Wenn Sie diese Option wählen, verwendet jedes Core-Gerät, das Sie aus dieser Vorlage erstellen, eine eindeutige AWS IoT Richtlinie, und Sie können die Berechtigungen für jedes einzelne Core-Gerät verwalten.
  - Eine -AWS IoTZertifikatressource. Diese Zertifikatressource muss den `-AWS::IoT::Certificate::Id`Parameter verwenden, um das Zertifikat an das Core-Gerät anzuhängen. Weitere Informationen finden Sie unter J-ust-in-time Bereitstellung im AWS IoT -Entwicklerhandbuch.
- Ein AWS IoT Bereitstellungsantragszertifikat und ein privater Schlüssel für die Flottenbereitstellungsvorlage. Sie können dieses Zertifikat und diesen privaten Schlüssel während der Herstellung in Geräte einbetten, sodass sich die Geräte registrieren und selbst bereitstellen können, wenn sie online gehen.

> **Important**
>
> Die Bereitstellung privater Schlüssel für den Anspruch sollte jederzeit gesichert werden, auch auf Greengrass-Core-Geräten. Wir empfehlen Ihnen, Amazon- CloudWatch Metriken und -Protokolle zu verwenden, um Hinweise auf Missbrauch zu überwachen, z. B. die unbefugte Verwendung des Anspruchszertifikats zur Bereitstellung von Geräten. Wenn Sie einen Missbrauch feststellen, deaktivieren Sie das Bereitstellungsantragszertifikat, damit es nicht für die Gerätebereitstellung verwendet werden kann. Weitere Informationen finden Sie unter Überwachung AWS IoT im AWS IoT Core -Entwicklerhandbuch.
> Um die Anzahl der Geräte und welche Geräte, die sich selbst in Ihrem registrieren, besser verwalten zu könnenAWS-Konto, können Sie beim Erstellen einer Flottenbereitstellungsvorlage einen Hook für die Vorabbereitstellung angeben. Ein Pre-

Provisioning-Hook ist eine -AWS LambdaFunktion, die Vorlagenparameter validiert, die Geräte während der Registrierung bereitstellen. Sie können beispielsweise einen Hook für die Vorabbereitstellung erstellen, der eine Geräte-ID anhand einer Datenbank prüft, um sicherzustellen, dass das Gerät über die Berechtigung zur Bereitstellung verfügt. Weitere Informationen finden Sie unter Vorabbereitstellung von Hooks im -AWS IoT CoreEntwicklerhandbuch.

- Eine AWS IoT Richtlinie, die Sie dem Bereitstellungsantragszertifikat anfügen, damit Geräte die Flottenbereitstellungsvorlage registrieren und verwenden können.

Themen

- Erstellen einer Token-Exchange-Rolle
- Erstellen einer AWS IoT-Richtlinie
- Erstellen einer Flottenbereitstellungsvorlage
- Erstellen eines Bereitstellungsantragszertifikats und eines privaten Schlüssels

Erstellen einer Token-Exchange-Rolle

Greengrass-Core-Geräte verwenden eine IAM-Servicerolle, die als Token-Exchange-Rolle bezeichnet wird, um Aufrufe von -AWSServices zu autorisieren. Das Gerät verwendet den AWS IoT Anmeldeinformationsanbieter, um temporäre AWS Anmeldeinformationen für diese Rolle abzurufen, die es dem Gerät ermöglichen, mit zu interagierenAWS IoT, Protokolle an Amazon CloudWatch Logs zu senden und benutzerdefinierte Komponentenartefakte von Amazon S3 herunterzuladen. Weitere Informationen finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

Sie verwenden einen -AWS IoTRollenalias, um die Token-Exchange-Rolle für Greengrass-Core-Geräte zu konfigurieren. Mit Rollenaliasnamen können Sie die Token-Exchange-Rolle für ein Gerät ändern, die Gerätekonfiguration jedoch beibehalten. Weitere Informationen finden Sie unter Autorisieren von direkten Aufrufen an -AWSServices im AWS IoT Core -Entwicklerhandbuch.

In diesem Abschnitt erstellen Sie eine IAM-Rolle für den Tokenaustausch und einen -AWS IoTRollenalias, der auf die Rolle verweist. Wenn Sie bereits ein Greengrass-Core-Gerät eingerichtet haben, können Sie seine Token-Austauschrolle und seinen Rollenalias verwenden, anstatt neue zu erstellen.

So erstellen Sie eine IAM-Rolle für den Tokenaustausch

1. Erstellen Sie eine IAM-Rolle, die Ihr Gerät als Token-Austauschrolle verwenden kann. Gehen Sie wie folgt vor:

   a. Erstellen Sie eine -Datei, die das Vertrauensrichtliniendokument enthält, das die Token-Exchange-Rolle benötigt.

      Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen der Datei zu verwenden.

      ```
      nano device-role-trust-policy.json
      ```

      Kopieren Sie den folgenden JSON-Code in die Datei .

      ```
      {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Principal": {
              "Service": "credentials.iot.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
          }
        ]
      }
      ```

   b. Erstellen Sie die Token-Exchange-Rolle mit dem Vertrauensrichtliniendokument.

      - Ersetzen Sie *GreengrassV2TokenExchangeRole* durch den Namen der zu erstellenden IAM-Rolle.

      ```
      aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-
      policy-document file://device-role-trust-policy.json
      ```

      Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anforderung erfolgreich ist.

      ```
      {
      ```

```
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassV2TokenExchangeRole",
    "RoleId": "AROAZ2YMUHYHK5OKM77FB",
    "Arn": "arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole",
    "CreateDate": "2021-02-06T00:13:29+00:00",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "credentials.iot.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
  }
```

c. Erstellen Sie eine -Datei, die das Zugriffsrichtliniendokument enthält, das die Token-Exchange-Rolle benötigt.

   Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen der Datei zu verwenden.

   ```
   nano device-role-access-policy.json
   ```

   Kopieren Sie den folgenden JSON-Code in die Datei .

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": [
           "logs:CreateLogGroup",
           "logs:CreateLogStream",
           "logs:PutLogEvents",
           "logs:DescribeLogStreams",
           "s3:GetBucketLocation"
         ],
   ```

```
      "Resource": "*"
    }
  ]
}
```

> **Note**
>
> Diese Zugriffsrichtlinie erlaubt keinen Zugriff auf Komponentenartefakte in S3-Buckets. Um benutzerdefinierte Komponenten bereitzustellen, die Artefakte in Amazon S3 definieren, müssen Sie der Rolle Berechtigungen hinzufügen, damit Ihr Core-Gerät Komponentenartefakte abrufen kann. Weitere Informationen finden Sie unter Erlauben Sie den Zugriff auf S3-Buckets für Komponentenartefakte.
> Wenn Sie noch keinen S3-Bucket für Komponentenartefakte haben, können Sie diese Berechtigungen später hinzufügen, nachdem Sie einen Bucket erstellt haben.

d. Erstellen Sie die IAM-Richtlinie aus dem Richtliniendokument.

- Ersetzen Sie *GreengrassV2TokenExchangeRoleAccess* durch den Namen der zu erstellenden IAM-Richtlinie.

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --
policy-document file://device-role-access-policy.json
```

Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anforderung erfolgreich ist.

```
{
  "Policy": {
    "PolicyName": "GreengrassV2TokenExchangeRoleAccess",
    "PolicyId": "ANPAZ2YMUHYHACI7C5Z66",
    "Arn": "arn:aws:iam::123456789012:policy/
GreengrassV2TokenExchangeRoleAccess",
    "Path": "/",
    "DefaultVersionId": "v1",
    "AttachmentCount": 0,
    "PermissionsBoundaryUsageCount": 0,
    "IsAttachable": true,
    "CreateDate": "2021-02-06T00:37:17+00:00",
    "UpdateDate": "2021-02-06T00:37:17+00:00"
```

```
  }
}
```

e. Fügen Sie die IAM-Richtlinie an die Token-Exchange-Rolle an.

- Ersetzen Sie *GreengrassV2TokenExchangeRole* durch den Namen der IAM-Rolle.
- Ersetzen Sie den Richtlinien-ARN durch den ARN der IAM-Richtlinie, die Sie im vorherigen Schritt erstellt haben.

```
aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-
arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess
```

Der Befehl hat keine Ausgabe, wenn die Anforderung erfolgreich ist.

2. Erstellen Sie einen -AWS IoTRollenalias, der auf die Token-Exchange-Rolle verweist.

- Ersetzen Sie durch *GreengrassCoreTokenExchangeRoleAlias* den Namen des zu erstellenden Rollenalias.
- Ersetzen Sie den Rollen-ARN durch den ARN der IAM-Rolle, die Sie im vorherigen Schritt erstellt haben.

```
aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-
arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole
```

Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anforderung erfolgreich ist.

```
{
  "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
}
```

> **Note**
>
> Um einen Rollenalias zu erstellen, müssen Sie über die Berechtigung verfügen, die IAM-Rolle für den Tokenaustausch an zu übergebenAWS IoT. Wenn Sie beim Versuch, einen Rollenalias zu erstellen, eine Fehlermeldung erhalten, überprüfen Sie, ob Ihr AWS Benutzer über diese Berechtigung verfügt. Weitere Informationen finden Sie unter

Erteilen von Berechtigungen an einen Benutzer zum Übergeben einer Rolle an einen -AWSService im AWS Identity and Access Management -Benutzerhandbuch.

Erstellen einer AWS IoT-Richtlinie

Nachdem Sie ein Gerät als -AWS IoTObjekt registriert haben, kann dieses Gerät ein digitales Zertifikat verwenden, um sich bei zu authentifizierenAWS. Dieses Zertifikat enthält eine oder mehrere AWS IoT Richtlinien, die die Berechtigungen definieren, die ein Gerät mit dem Zertifikat verwenden kann. Diese Richtlinien ermöglichen es dem Gerät, mit AWS IoT und zu kommunizierenAWS IoT Greengrass.

Mit der AWS IoT Flottenbereitstellung stellen Geräte eine Verbindung zu her, um ein Gerätezertifikat AWS IoT zu erstellen und herunterzuladen. In der Flottenbereitstellungsvorlage, die Sie im nächsten Abschnitt erstellen, können Sie angeben, ob dieselbe AWS IoT Richtlinie an die Zertifikate aller Geräte AWS IoT anfügt oder eine neue Richtlinie für jedes Gerät erstellt.

In diesem Abschnitt erstellen Sie eine -AWS IoTRichtlinie, die an die Zertifikate aller Geräte AWS IoT angefügt wird. Mit diesem Ansatz können Sie Berechtigungen für alle Geräte als Flotte verwalten. Wenn Sie lieber eine neue AWS IoT Richtlinie für jedes Gerät erstellen möchten, können Sie diesen Abschnitt überspringen und auf die darin enthaltene Richtlinie verweisen, wenn Sie Ihre Flottenvorlage definieren.

So erstellen Sie eine AWS IoT-Richtlinie

- Erstellen Sie eine -AWS IoTRichtlinie, die die AWS IoT Berechtigungen für Ihre Flotte von Greengrass-Core-Geräten definiert. Die folgende Richtlinie ermöglicht den Zugriff auf alle MQTT-Themen und Greengrass-Operationen, sodass Ihr Gerät mit benutzerdefinierten Anwendungen und zukünftigen Änderungen funktioniert, die neue Greengrass-Operationen erfordern. Diese Richtlinie gewährt auch die `-iot:AssumeRoleWithCertificate`Berechtigung, die es Ihren Geräten ermöglicht, die Token-Exchange-Rolle zu verwenden, die Sie im vorherigen Abschnitt erstellt haben. Sie können diese Richtlinie je nach Anwendungsfall einschränken. Weitere Informationen finden Sie unter Minimale AWS IoT Richtlinie für -AWS IoT Greengrass V2Core-Geräte.

  Gehen Sie wie folgt vor:

  a. Erstellen Sie eine -Datei, die das AWS IoT Richtliniendokument enthält, das Greengrass-Core-Geräte benötigen.

Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen der Datei zu verwenden.

```
nano greengrass-v2-iot-policy.json
```

Kopieren Sie den folgenden JSON-Code in die Datei .

- Ersetzen Sie die `iot:AssumeRoleWithCertificate` Ressource durch den ARN des AWS IoT Rollenalias, den Sie im vorherigen Abschnitt erstellt haben.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive",
        "iot:Connect",
        "greengrass:*"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:AssumeRoleWithCertificate",
      "Resource": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
    }
  ]
}
```

b. Erstellen Sie eine -AWS IoTRichtlinie aus dem Richtliniendokument.

- Ersetzen Sie *GreengrassV2IoTThingPolicy* durch den Namen der zu erstellenden Richtlinie.

```
aws iot create-policy --policy-name GreengrassV2IoTThingPolicy --policy-
document file://greengrass-v2-iot-policy.json
```

Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anforderung erfolgreich ist.

```
{
  "policyName": "GreengrassV2IoTThingPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
  "policyDocument": "{
    \"Version\": \"2012-10-17\",
    \"Statement\": [
      {
        \"Effect\": \"Allow\",
        \"Action\": [
          \"iot:Publish\",
          \"iot:Subscribe\",
          \"iot:Receive\",
          \"iot:Connect\",
          \"greengrass:*\"
        ],
        \"Resource\": [
          \"*\"
        ]
      },
      {
        \"Effect\": \"Allow\",
        \"Action\": \"iot:AssumeRoleWithCertificate\",
        \"Resource\": \"arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias\"
      }
    ]
  }",
  "policyVersionId": "1"
}
```

Erstellen einer Flottenbereitstellungsvorlage

AWS IoT Flottenbereitstellungsvorlagen definieren, wie AWS IoT Objekte, Richtlinien und Zertifikate bereitgestellt werden. Um Greengrass-Core-Geräte mit dem Flottenbereitstellungs-Plugin bereitzustellen, müssen Sie eine Vorlage erstellen, die Folgendes angibt:

- Eine -AWS IoTObjektressource. Sie können eine Liste vorhandener Objektgruppen angeben, um Komponenten auf jedem Gerät bereitzustellen, wenn es online geht.
- Eine -AWS IoTRichtlinienressource. Diese Ressource kann eine der folgenden Eigenschaften definieren:
  - Der Name einer vorhandenen AWS IoT Richtlinie. Wenn Sie diese Option wählen, verwenden die Core-Geräte, die Sie aus dieser Vorlage erstellen, dieselbe AWS IoT Richtlinie, und Sie können ihre Berechtigungen als Flotte verwalten.
  - Ein -AWS IoTRichtliniendokument. Wenn Sie diese Option wählen, verwendet jedes Core-Gerät, das Sie aus dieser Vorlage erstellen, eine eindeutige AWS IoT Richtlinie, und Sie können die Berechtigungen für jedes einzelne Core-Gerät verwalten.
- Eine -AWS IoTZertifikatressource. Diese Zertifikatressource muss den `AWS::IoT::Certificate::Id` Parameter verwenden, um das Zertifikat an das Core-Gerät anzuhängen. Weitere Informationen finden Sie unter J-ust-in-time Bereitstellung im AWS IoT - Entwicklerhandbuch.

In der Vorlage können Sie angeben, um das AWS IoT Objekt einer Liste vorhandener Objektgruppen hinzuzufügen. Wenn das Core-Gerät AWS IoT Greengrass zum ersten Mal eine Verbindung zu herstellt, erhält es Greengrass-Bereitstellungen für jede Objektgruppe, in der es Mitglied ist. Sie können Objektgruppen verwenden, um die neueste Software auf jedem Gerät bereitzustellen, sobald sie online ist. Weitere Informationen finden Sie unter Bereitstellen von AWS IoT Greengrass Komponenten auf Geräten.

Der AWS IoT Service benötigt Berechtigungen zum Erstellen und Aktualisieren von AWS IoT Ressourcen in Ihrem AWS-Konto bei der Bereitstellung von Geräten. Um dem AWS IoT Service Zugriff zu gewähren, erstellen Sie eine IAM-Rolle und geben sie an, wenn Sie die Vorlage erstellen. AWS IoT bietet eine -verwaltete Richtlinie, AWSIoTThingsRegistration, die den Zugriff auf alle Berechtigungen ermöglicht, die bei der Bereitstellung von Geräten verwenden AWS IoT könnte. Sie können diese verwaltete Richtlinie verwenden oder eine benutzerdefinierte Richtlinie erstellen, die die Berechtigungen in der verwalteten Richtlinie für Ihren Anwendungsfall einschränkt.

In diesem Abschnitt erstellen Sie eine IAM-Rolle, die AWS IoT es ermöglicht, Ressourcen für Geräte bereitzustellen, und Sie erstellen eine Flottenbereitstellungsvorlage, die diese IAM-Rolle verwendet.

So erstellen Sie eine Flottenbereitstellungsvorlage

1. Erstellen Sie eine IAM-Rolle, die annehmen AWS IoT kann, um Ressourcen in Ihrem bereitzustellenAWS-Konto. Gehen Sie wie folgt vor:

   a. Erstellen Sie eine -Datei, die das Vertrauensrichtliniendokument enthält, das AWS IoT die Übernahme der Rolle ermöglicht.

      Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen der Datei zu verwenden.

      ```
      nano aws-iot-trust-policy.json
      ```

      Kopieren Sie den folgenden JSON-Code in die Datei .

      ```
      {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Principal": {
              "Service": "iot.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
          }
        ]
      }
      ```

   b. Erstellen Sie eine IAM-Rolle mit dem Vertrauensrichtliniendokument.

      - Ersetzen Sie *GreengrassFleetProvisioningRole* durch den Namen der zu erstellenden IAM-Rolle.

      ```
      aws iam create-role --role-name GreengrassFleetProvisioningRole --assume-role-
      policy-document file://aws-iot-trust-policy.json
      ```

Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anforderung erfolgreich ist.

```
{
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassFleetProvisioningRole",
    "RoleId": "AROAZ2YMUHYHK5OKM77FB",
    "Arn": "arn:aws:iam::123456789012:role/GreengrassFleetProvisioningRole",
    "CreateDate": "2021-07-26T00:15:12+00:00",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "iot.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
  }
}
```

c. Überprüfen Sie die AWSIoTThingsRegistration Richtlinie, die den Zugriff auf alle Berechtigungen ermöglicht, die bei der Bereitstellung von Geräten verwenden AWS IoT könnte. Sie können diese verwaltete Richtlinie verwenden oder eine benutzerdefinierte Richtlinie erstellen, die eingeschränkte Berechtigungen für Ihren Anwendungsfall definiert. Wenn Sie eine benutzerdefinierte Richtlinie erstellen möchten, tun Sie dies jetzt.

d. Fügen Sie die IAM-Richtlinie an die Flottenbereitstellungsrolle an.

- Ersetzen Sie *GreengrassFleetProvisioningRole* durch den Namen der IAM-Rolle.
- Wenn Sie im vorherigen Schritt eine benutzerdefinierte Richtlinie erstellt haben, ersetzen Sie den Richtlinien-ARN durch den ARN der zu verwendenden IAM-Richtlinie.

```
aws iam attach-role-policy --role-name GreengrassFleetProvisioningRole --
policy-arn arn:aws:iam::aws:policy/service-role/AWSIoTThingsRegistration
```

Der Befehl hat keine Ausgabe, wenn die Anforderung erfolgreich ist.

2. (Optional) Erstellen Sie einen Hook für die Vorabbereitstellung, bei dem es sich um eine -AWS LambdaFunktion handelt, die Vorlagenparameter validiert, die Geräte während der Registrierung angeben. Sie können einen Hook vor der Bereitstellung verwenden, um mehr Kontrolle darüber zu erhalten, welche und wie viele Geräte in Ihrem integriert sindAWS-Konto. Weitere Informationen finden Sie unter Vorabbereitstellung von Hooks im -AWS IoT CoreEntwicklerhandbuch.
3. Erstellen Sie eine Flottenbereitstellungsvorlage. Gehen Sie wie folgt vor:

   a. Erstellen Sie eine -Datei, die das Bereitstellungsvorlagendokument enthält.

      Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen der Datei zu verwenden.

      ```
      nano greengrass-fleet-provisioning-template.json
      ```

      Schreiben Sie das Bereitstellungsvorlagendokument. Sie können mit der folgenden Beispielbereitstellungsvorlage beginnen, die angibt, dass ein -AWS IoTObjekt mit den folgenden Eigenschaften erstellt werden soll:

      - Der Name des Objekts ist der Wert, den Sie im `ThingName` Vorlagenparameter angeben.
      - Das Objekt ist Mitglied der Objektgruppe, die Sie im `ThingGroupName` Vorlagenparameter angeben. Die Objektgruppe muss in Ihrem vorhanden seinAWS-Konto.
      - Dem Objektzertifikat ist die AWS IoT Richtlinie mit dem Namen `GreengrassV2IoTThingPolicy` angefügt.

      Weitere Informationen finden Sie unter Bereitstellungsvorlagen im AWS IoT Core - Entwicklerhandbuch.

      ```
      {
        "Parameters": {
          "ThingName": {
            "Type": "String"
          },
          "ThingGroupName": {
            "Type": "String"
      ```

```
    },
    "AWS::IoT::Certificate::Id": {
      "Type": "String"
    }
  },
  "Resources": {
    "MyThing": {
      "OverrideSettings": {
        "AttributePayload": "REPLACE",
        "ThingGroups": "REPLACE",
        "ThingTypeName": "REPLACE"
      },
      "Properties": {
        "AttributePayload": {},
        "ThingGroups": [
          {
            "Ref": "ThingGroupName"
          }
        ],
        "ThingName": {
          "Ref": "ThingName"
        }
      },
      "Type": "AWS::IoT::Thing"
    },
    "MyPolicy": {
      "Properties": {
        "PolicyName": "GreengrassV2IoTThingPolicy"
      },
      "Type": "AWS::IoT::Policy"
    },
    "MyCertificate": {
      "Properties": {
        "CertificateId": {
          "Ref": "AWS::IoT::Certificate::Id"
        },
        "Status": "Active"
      },
      "Type": "AWS::IoT::Certificate"
    }
  }
}
```

> **Note**
>
> *MyThing*, und *MyCertificate* sind beliebige Namen*MyPolicy*, die jede Ressourcenspezifikation in der Flottenbereitstellungsvorlage identifizieren. verwendet diese Namen AWS IoT nicht in den Ressourcen, die es aus der Vorlage erstellt. Sie können diese Namen verwenden oder sie durch Werte ersetzen, die Ihnen helfen, jede Ressource in der Vorlage zu identifizieren.

b. Erstellen Sie die Flottenbereitstellungsvorlage aus dem Dokument der Bereitstellungsvorlage.

- Ersetzen Sie *GreengrassFleetProvisioningTemplate* durch den Namen der zu erstellenden Vorlage.
- Ersetzen Sie die Vorlagenbeschreibung durch eine Beschreibung für Ihre Vorlage.
- Ersetzen Sie den ARN der Bereitstellungsrolle durch den ARN der Rolle, die Sie zuvor erstellt haben.

Linux or Unix

```
aws iot create-provisioning-template \
  --template-name GreengrassFleetProvisioningTemplate \
  --description "A provisioning template for Greengrass core devices." \
  --provisioning-role-arn "arn:aws:iam::123456789012:role/
GreengrassFleetProvisioningRole" \
  --template-body file://greengrass-fleet-provisioning-template.json \
  --enabled
```

Windows Command Prompt (CMD)

```
aws iot create-provisioning-template ^
  --template-name GreengrassFleetProvisioningTemplate ^
  --description "A provisioning template for Greengrass core devices." ^
  --provisioning-role-arn "arn:aws:iam::123456789012:role/
GreengrassFleetProvisioningRole" ^
  --template-body file://greengrass-fleet-provisioning-template.json ^
  --enabled
```

PowerShell

```
aws iot create-provisioning-template `
  --template-name GreengrassFleetProvisioningTemplate `
  --description "A provisioning template for Greengrass core devices." `
  --provisioning-role-arn "arn:aws:iam::123456789012:role/
GreengrassFleetProvisioningRole" `
  --template-body file://greengrass-fleet-provisioning-template.json `
  --enabled
```

> **Note**
>
> Wenn Sie einen Pre-Provisioning-Hook erstellt haben, geben Sie den ARN der Lambda-Funktion des Pre-Provisioning-Hooks mit dem `--pre-provisioning-hook` -Argument an.
>
> ```
> --pre-provisioning-hook targetArn=arn:aws:lambda:us-
> west-2:123456789012:function:GreengrassPreProvisioningHook
> ```

Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anforderung erfolgreich ist.

```
{
    "templateArn": "arn:aws:iot:us-west-2:123456789012:provisioningtemplate/
GreengrassFleetProvisioningTemplate",
    "templateName": "GreengrassFleetProvisioningTemplate",
    "defaultVersionId": 1
}
```

Erstellen eines Bereitstellungsantragszertifikats und eines privaten Schlüssels

Anforderungszertifikate sind X.509-Zertifikate, mit denen Geräte sich als AWS IoT Objekte registrieren und ein eindeutiges X.509-Gerätezertifikat abrufen können, das für reguläre Vorgänge verwendet werden kann. Nachdem Sie ein Anspruchszertifikat erstellt haben, fügen Sie eine -AWS IoTRichtlinie an, die es Geräten ermöglicht, damit sie eindeutige Gerätezertifikate erstellen und mit

einer Flottenbereitstellungsvorlage bereitstellen können. Geräte mit dem Anspruchszertifikat können nur mithilfe der Bereitstellungsvorlage bereitstellen, die Sie in der AWS IoT Richtlinie zulassen.

In diesem Abschnitt erstellen Sie das Antragszertifikat und konfigurieren es für Geräte zur Verwendung mit der Flottenbereitstellungsvorlage, die Sie im vorherigen Abschnitt erstellt haben.

> **Important**
>
> Die Bereitstellung privater Schlüssel für den Anspruch sollte jederzeit gesichert werden, auch auf Greengrass-Core-Geräten. Wir empfehlen Ihnen, Amazon- CloudWatch Metriken und -Protokolle zu verwenden, um Hinweise auf Missbrauch zu überwachen, z. B. die unbefugte Verwendung des Anspruchszertifikats zur Bereitstellung von Geräten. Wenn Sie einen Missbrauch feststellen, deaktivieren Sie das Bereitstellungsantragszertifikat, damit es nicht für die Gerätebereitstellung verwendet werden kann. Weitere Informationen finden Sie unter Überwachung AWS IoT im AWS IoT Core -Entwicklerhandbuch.
> Um die Anzahl der Geräte und welche Geräte, die sich selbst in Ihrem registrieren, besser verwalten zu könnenAWS-Konto, können Sie beim Erstellen einer Flottenbereitstellungsvorlage einen Hook für die Vorabbereitstellung angeben. Ein Pre-Provisioning-Hook ist eine -AWS LambdaFunktion, die Vorlagenparameter validiert, die Geräte während der Registrierung bereitstellen. Sie können beispielsweise einen Hook für die Vorabbereitstellung erstellen, der eine Geräte-ID anhand einer Datenbank prüft, um sicherzustellen, dass das Gerät über die Berechtigung zur Bereitstellung verfügt. Weitere Informationen finden Sie unter Vorabbereitstellung von Hooks im -AWS IoT CoreEntwicklerhandbuch.

So erstellen Sie ein Bereitstellungsantragszertifikat und einen privaten Schlüssel

1. Erstellen Sie einen Ordner, in dem Sie das Antragszertifikat und den privaten Schlüssel herunterladen.

   ```
   mkdir claim-certs
   ```

2. Erstellen und speichern Sie ein Zertifikat und einen privaten Schlüssel für die Bereitstellung. AWS IoT stellt Clientzertifikate bereit, die von der Amazon Root Certificate Authority (CA) signiert wurden.

Linux or Unix

```
aws iot create-keys-and-certificate \
  --certificate-pem-outfile "claim-certs/claim.pem.crt" \
  --public-key-outfile "claim-certs/claim.public.pem.key" \
  --private-key-outfile "claim-certs/claim.private.pem.key" \
  --set-as-active
```

Windows Command Prompt (CMD)

```
aws iot create-keys-and-certificate ^
  --certificate-pem-outfile "claim-certs/claim.pem.crt" ^
  --public-key-outfile "claim-certs/claim.public.pem.key" ^
  --private-key-outfile "claim-certs/claim.private.pem.key" ^
  --set-as-active
```

PowerShell

```
aws iot create-keys-and-certificate `
  --certificate-pem-outfile "claim-certs/claim.pem.crt" `
  --public-key-outfile "claim-certs/claim.public.pem.key" `
  --private-key-outfile "claim-certs/claim.private.pem.key" `
  --set-as-active
```

Die Antwort enthält Informationen über das Zertifikat, falls die Anforderung erfolgreich ist. Speichern Sie den ARN des Zertifikats, um ihn später zu verwenden.

3. Erstellen und fügen Sie eine -AWS IoTRichtlinie an, die es Geräten ermöglicht, das Zertifikat zu verwenden, um eindeutige Gerätezertifikate zu erstellen und mit der Flottenbereitstellungsvorlage bereitzustellen. Die folgende Richtlinie ermöglicht den Zugriff auf die MQTT-API für die Gerätebereitstellung. Weitere Informationen finden Sie unter MQTT-API für die Gerätebereitstellung im AWS IoT Core -Entwicklerhandbuch.

   Gehen Sie wie folgt vor:

   a. Erstellen Sie eine -Datei, die das AWS IoT Richtliniendokument enthält, das Greengrass-Core-Geräte benötigen.

Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen der Datei zu verwenden.

```
nano greengrass-provisioning-claim-iot-policy.json
```

Kopieren Sie den folgenden JSON-Code in die Datei .

- Ersetzen Sie jede Instance der *Region* durch die AWS-Region, in der Sie die Flottenbereitstellung eingerichtet haben.
- Ersetzen Sie jede Instance von *account-id* durch Ihre AWS-Konto -ID.
- Ersetzen Sie jede Instance von *GreengrassFleetProvisioningTemplate* durch den Namen der Flottenbereitstellungsvorlage, die Sie im vorherigen Abschnitt erstellt haben.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:Connect",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:region:account-id:topic/$aws/certificates/create/*",
        "arn:aws:iot:region:account-id:topic/$aws/provisioning-
templates/GreengrassFleetProvisioningTemplate/provision/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
      "Resource": [
        "arn:aws:iot:region:account-id:topicfilter/$aws/certificates/create/*",
```

```
        "arn:aws:iot:region:account-id:topicfilter/$aws/provisioning-
templates/GreengrassFleetProvisioningTemplate/provision/*"
      ]
    }
  ]
}
```

b. Erstellen Sie eine -AWS IoTRichtlinie aus dem Richtliniendokument.

- Ersetzen Sie durch *GreengrassProvisioningClaimPolicy* den Namen der zu erstellenden Richtlinie.

```
aws iot create-policy --policy-name GreengrassProvisioningClaimPolicy --policy-
document file://greengrass-provisioning-claim-iot-policy.json
```

Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anforderung erfolgreich ist.

```
{
  "policyName": "GreengrassProvisioningClaimPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassProvisioningClaimPolicy",
  "policyDocument": "{
    \"Version\": \"2012-10-17\",
    \"Statement\": [
      {
        \"Effect\": \"Allow\",
        \"Action\": \"iot:Connect\",
        \"Resource\": \"*\"
      },
      {
        \"Effect\": \"Allow\",
        \"Action\": [
          \"iot:Publish\",
          \"iot:Receive\"
        ],
        \"Resource\": [
          \"arn:aws:iot:region:account-id:topic/$aws/certificates/create/*\",
          \"arn:aws:iot:region:account-id:topic/$aws/provisioning-
templates/GreengrassFleetProvisioningTemplate/provision/*\"
        ]
      },
```

```
      {
        \"Effect\": \"Allow\",
        \"Action\": \"iot:Subscribe\",
        \"Resource\": [
          \"arn:aws:iot:region:account-id:topicfilter/$aws/certificates/create/
*\",
          \"arn:aws:iot:region:account-id:topicfilter/$aws/provisioning-
templates/GreengrassFleetProvisioningTemplate/provision/*\"
        ]
      }
    ]
  }",
  "policyVersionId": "1"
}
```

4. Fügen Sie die AWS IoT Richtlinie an das Bereitstellungsantragszertifikat an.

   - Ersetzen Sie durch *GreengrassProvisioningClaimPolicy* den Namen der anzuhängenden Richtlinie.
   - Ersetzen Sie den Ziel-ARN durch den ARN des Bereitstellungsantragszertifikats.

   ```
   aws iot attach-policy --policy-name GreengrassProvisioningClaimPolicy --
   target arn:aws:iot:us-west-2:123456789012:cert/
   aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
   ```

   Der Befehl hat keine Ausgabe, wenn die Anforderung erfolgreich ist.

Sie verfügen jetzt über ein Bereitstellungsantragszertifikat und einen privaten Schlüssel, mit dem sich Geräte bei registrieren AWS IoT und sich selbst als Greengrass-Core-Geräte bereitstellen können. Sie können das Anspruchszertifikat und den privaten Schlüssel während der Herstellung in Geräte einbetten oder das Zertifikat und den Schlüssel vor der Installation der AWS IoT Greengrass Core-Software auf Geräte kopieren. Weitere Informationen finden Sie unter Installieren Sie die AWS IoT Greengrass Core-Software mit AWS IoT Flottenbereitstellung.

Konfigurieren des AWS IoT Flottenbereitstellungs-Plugins

Das AWS IoT Flottenbereitstellungs-Plugin bietet die folgenden Konfigurationsparameter, die Sie anpassen können, wenn Sie die AWS IoT Greengrass Core-Software mit Flottenbereitstellung installieren.

`rootPath`

Der Pfad zum Ordner, der als Stamm für die AWS IoT Greengrass Core-Software verwendet werden soll.

`awsRegion`

Die AWS-Region, die das Flottenbereitstellungs-Plugin zur Bereitstellung von AWS Ressourcen verwendet.

`iotDataEndpoint`

Der AWS IoT Datenendpunkt für Ihr AWS-Konto.

`iotCredentialEndpoint`

Der AWS IoT Anmeldeinformationsendpunkt für Ihr AWS-Konto.

`iotRoleAlias`

Der AWS IoT Rollenalias, der auf eine IAM-Rolle für den Tokenaustausch verweist. Der AWS IoT Anmeldeinformationsanbieter übernimmt diese Rolle, um dem Greengrass-Kerngerät die Interaktion mit -AWSServices zu ermöglichen. Weitere Informationen finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

`provisioningTemplate`

Die AWS IoT Flottenbereitstellungsvorlage, die zur Bereitstellung von AWS Ressourcen verwendet werden soll. Diese Vorlage muss Folgendes angeben:

- Eine -AWS IoTObjektressource. Sie können eine Liste vorhandener Objektgruppen angeben, um Komponenten auf jedem Gerät bereitzustellen, wenn es online geht.
- Eine -AWS IoTRichtlinienressource. Diese Ressource kann eine der folgenden Eigenschaften definieren:
  - Der Name einer vorhandenen AWS IoT Richtlinie. Wenn Sie diese Option wählen, verwenden die Core-Geräte, die Sie aus dieser Vorlage erstellen, dieselbe AWS IoT Richtlinie, und Sie können ihre Berechtigungen als Flotte verwalten.
  - Ein -AWS IoTRichtliniendokument. Wenn Sie diese Option wählen, verwendet jedes Core-Gerät, das Sie aus dieser Vorlage erstellen, eine eindeutige AWS IoT Richtlinie, und Sie können die Berechtigungen für jedes einzelne Core-Gerät verwalten.
- Eine -AWS IoTZertifikatressource. Diese Zertifikatressource muss den `AWS::IoT::Certificate::Id` Parameter verwenden, um das Zertifikat an das Core-Gerät

anzuhängen. Weitere Informationen finden Sie unter J-ust-in-time Bereitstellung im AWS IoT - Entwicklerhandbuch.

Weitere Informationen finden Sie unter Bereitstellungsvorlagen im AWS IoT Core - Entwicklerhandbuch.

`claimCertificatePath`

Der Pfad zum Bereitstellungsantragszertifikat für die Bereitstellungsvorlage, die Sie in angeben`provisioningTemplate`. Weitere Informationen finden Sie unter CreateProvisioningClaim in der AWS IoT Core-API-Referenz.

`claimCertificatePrivateKeyPath`

Der Pfad zum privaten Schlüssel des Bereitstellungsantragszertifikats für die Bereitstellungsvorlage, die Sie in angeben`provisioningTemplate`. Weitere Informationen finden Sie unter CreateProvisioningClaim in der AWS IoT Core-API-Referenz.

> **Important**
>
> Die Bereitstellung privater Schlüssel für den Anspruch sollte jederzeit gesichert werden, auch auf Greengrass-Core-Geräten. Wir empfehlen Ihnen, Amazon- CloudWatch Metriken und -Protokolle zu verwenden, um Hinweise auf Missbrauch zu überwachen, z. B. die unbefugte Verwendung des Anspruchszertifikats zur Bereitstellung von Geräten. Wenn Sie einen Missbrauch feststellen, deaktivieren Sie das Bereitstellungsantragszertifikat, damit es nicht für die Gerätebereitstellung verwendet werden kann. Weitere Informationen finden Sie unter Überwachung AWS IoT im AWS IoT Core -Entwicklerhandbuch.
> Um die Anzahl der Geräte und welche Geräte, die sich selbst in Ihrem registrieren, besser verwalten zu könnenAWS-Konto, können Sie beim Erstellen einer Flottenbereitstellungsvorlage einen Hook für die Vorabbereitstellung angeben. Ein Pre-Provisioning-Hook ist eine -AWS LambdaFunktion, die Vorlagenparameter validiert, die Geräte während der Registrierung bereitstellen. Sie können beispielsweise einen Hook für die Vorabbereitstellung erstellen, der eine Geräte-ID anhand einer Datenbank prüft, um sicherzustellen, dass das Gerät über die Berechtigung zur Bereitstellung verfügt. Weitere Informationen finden Sie unter Vorabbereitstellung von Hooks im -AWS IoT CoreEntwicklerhandbuch.

`rootCaPath`

Der Pfad zum Amazon Root Certificate Authority (CA)-Zertifikat.

`templateParameters`

(Optional) Die Zuordnung der Parameter, die der Flottenbereitstellungsvorlage bereitgestellt werden sollen. Weitere Informationen finden Sie im Abschnitt Parameter von Bereitstellungsvorlagen im AWS IoT Core -Entwicklerhandbuch.

`deviceId`

(Optional) Die Geräte-ID, die als Client-ID verwendet werden soll, wenn das Flottenbereitstellungs-Plugin eine MQTT-Verbindung zu erstelltAWS IoT.

Standard: Eine zufällige UUID.

`mqttPort`

(Optional) Der Port, der für MQTT-Verbindungen verwendet werden soll.

Standard: `8883`

`proxyUrl`

(Optional) Die URL des Proxy-Servers im Format `scheme://userinfo@host:port`. Um einen HTTPS-Proxy zu verwenden, müssen Sie Version 1.1.0 oder höher des Flottenbereitstellungs-Plugins verwenden.

- `scheme` – Das Schema, das `http` oder sein muss`https`.

  > **Important**
  >
  > Greengrass-Core-Geräte müssen Greengrass-Kern v2.5.0 oder höher ausführen, um HTTPS-Proxys verwenden zu können.
  > Wenn Sie einen HTTPS-Proxy konfigurieren, müssen Sie das Proxyserver-CA-Zertifikat zum Amazon-Root-CA-Zertifikat des Core-Geräts hinzufügen. Weitere Informationen finden Sie unter Ermöglichen Sie dem Kerngerät, einem HTTPS-Proxy zu vertrauen.

- `userinfo` – (Optional) Der Benutzername und das Passwort. Wenn Sie diese Informationen in der angeben`url`, ignoriert das Greengrass-Kerngerät die `password` Felder `username` und .
- `host` – Der Hostname oder die IP-Adresse des Proxy-Servers.
- `port` – (Optional) Die Portnummer. Wenn Sie den Port nicht angeben, verwendet das Greengrass-Core-Gerät die folgenden Standardwerte:

- `http` – 80
- `https` – 443

`proxyUserName`

(Optional) Der Benutzername, der den Proxy-Server authentifiziert.

`proxyPassword`

(Optional) Der Benutzername, der den Proxy-Server authentifiziert.

csrPath

(Optional) Der Pfad zur CSR-Datei (Certificate Signing Request), die zum Erstellen des Gerätezertifikats aus einer CSR verwendet werden soll. Weitere Informationen finden Sie unter Bereitstellung nach Anspruch im AWS IoT Core Entwicklerhandbuch für .

csrPrivateKeyPfad

(Optional, erforderlich, wenn deklariert `csrPath` ist) Der Pfad zu dem privaten Schlüssel, der zum Generieren der CSR verwendet wird. Der private Schlüssel muss zum Generieren der CSR verwendet worden sein. Weitere Informationen finden Sie unter Bereitstellung nach Anspruch im AWS IoT Core Entwicklerhandbuch für .

AWS IoT Changelog des Plug-ins für die Flottenbereitstellung

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen des Plug-ins AWS IoT Fleet Provisioning by Claim (`aws.greengrass.FleetProvisioningByClaim`) beschrieben.

Version	Änderungen
1.2.1	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem das Fleet Provisioning Plugin während eines Greengrass Nucleus-Starts offline ist. Das Fleet Provisioning-Plugin versucht nun auf unbestimmte Zeit, MQTT Connect-Aufrufe zu wiederholen.
1.2.0	Fehlerkorrekturen und Verbesserungen • Integriert die Unterstützung für die Gerätebereitstellung über eine Zertifikatsignieranforderung mit konfigurierbarem privaten Schlüsselpfad.

Version	Änderungen
	• Kleinere Korrekturen und Verbesserungen.
1.1.0	Fehlerkorrekturen und Verbesserungen • Fügt Unterstützung für zusätzliche Dateipfadformate hinzu, wenn Sie das Plugin auf Windows-Geräten konfigurieren. • Fügt Unterstützung für HTTPS-Netzwerk-Proxykonfigurationen hinzu. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy und Ermöglichen Sie dem Kerngerät, einem HTTPS-Proxy zu vertrauen.
1.0.0	Erste Version

Installieren Sie die AWS IoT Greengrass Core-Software mit benutzerdefinierter Ressourcenbereitstellung

Diese Funktion ist für Version 2.4.0 und höher der Greengrass Nucleus-Komponente verfügbar.

Das AWS IoT Greengrass Core-Softwareinstallationsprogramm bietet eine Java-Schnittstelle, die Sie in einem benutzerdefinierten Plugin implementieren können, das die erforderlichen Ressourcen bereitstellt. AWS Sie können ein Provisioning-Plugin entwickeln, um benutzerdefinierte X.509-Clientzertifikate zu verwenden oder komplexe Bereitstellungsschritte auszuführen, die andere Installationsprozesse nicht unterstützen. Weitere Informationen finden Sie im Entwicklerhandbuch unter Erstellen eigener Client-Zertifikate.AWS IoT Core

Um bei der Installation der AWS IoT Greengrass Core-Software ein benutzerdefiniertes Provisioning-Plugin auszuführen, erstellen Sie eine JAR-Datei, die Sie dem Installationsprogramm zur Verfügung stellen. Das Installationsprogramm führt das Plugin aus und das Plugin gibt eine Bereitstellungskonfiguration zurück, die die AWS Ressourcen für das Greengrass-Core-Gerät definiert. Das Installationsprogramm verwendet diese Informationen, um die AWS IoT Greengrass Core-Software auf dem Gerät zu konfigurieren. Weitere Informationen finden Sie unter Entwickeln Sie benutzerdefinierte Provisioning-Plugins.

> **Important**
>
> Bevor Sie die AWS IoT Greengrass Core-Software herunterladen, überprüfen Sie, ob Ihr Core-Gerät die Anforderungen für die Installation und Ausführung der AWS IoT Greengrass Core-Software v2.0 erfüllt.

Themen

Voraussetzungen

Um die AWS IoT Greengrass Core-Software mit benutzerdefinierter Bereitstellung zu installieren, müssen Sie über Folgendes verfügen:

- Eine JAR-Datei für ein benutzerdefiniertes Provisioning-Plugin, das das implementiert. `DeviceIdentityInterface` Das benutzerdefinierte Provisioning-Plugin muss Werte für jeden System- und Nucleus-Konfigurationsparameter zurückgeben. Andernfalls müssen Sie diese Werte während der Installation in der Konfigurationsdatei angeben. Weitere Informationen finden Sie unter Entwickeln Sie benutzerdefinierte Provisioning-Plugins.

Richten Sie die Geräteumgebung ein

Folgen Sie den Schritten in diesem Abschnitt, um ein Linux- oder Windows-Gerät einzurichten, das als Ihr AWS IoT Greengrass Kerngerät verwendet werden soll.

Richten Sie ein Linux-Gerät ein

Um ein Linux-Gerät einzurichten für AWS IoT Greengrass V2

1. Installieren Sie die Java-Runtime, die für die Ausführung der AWS IoT Greengrass Core-Software erforderlich ist. Wir empfehlen, dass Sie die Langzeit-Support-Versionen von Amazon

Corretto oder OpenJDK verwenden. Version 8 oder höher ist erforderlich. Die folgenden Befehle zeigen Ihnen, wie Sie OpenJDK auf Ihrem Gerät installieren.

- Für Debian- oder Ubuntu-basierte Distributionen:

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

- Für Red Hat-basierte Distributionen:

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

- Für Amazon Linux 2:

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

- Für Amazon Linux 2023:

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

Führen Sie nach Abschluss der Installation den folgenden Befehl aus, um zu überprüfen, ob Java auf Ihrem Linux-Gerät ausgeführt wird.

```
java -version
```

Der Befehl druckt die Version von Java, die auf dem Gerät ausgeführt wird. Bei einer Debian-basierten Distribution könnte die Ausgabe beispielsweise dem folgenden Beispiel ähneln.

```
openjdk version "11.0.9.1" 2020-11-04
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
```

2. (Optional) Erstellen Sie den Standardsystembenutzer und die Standardgruppe, die Komponenten auf dem Gerät ausführen. Sie können auch festlegen, dass der AWS IoT Greengrass Core-Software-Installer diesen Benutzer und diese Gruppe während der Installation mit dem `--component-default-user` Installer-Argument erstellt. Weitere Informationen finden Sie unter Installer-Argumente.

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

```
sudo groupadd --system ggc_group
```

3. Stellen Sie sicher, dass der Benutzer, der die AWS IoT Greengrass Core-Software ausführt (in der Regel`root`), über die erforderlichen Rechte verfügt, die Software `sudo` mit jedem beliebigen Benutzer und jeder Gruppe auszuführen.

   a. Führen Sie den folgenden Befehl aus, um die `/etc/sudoers` Datei zu öffnen.

   ```
   sudo visudo
   ```

   b. Stellen Sie sicher, dass die Berechtigung für den Benutzer wie im folgenden Beispiel aussieht.

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

4. (Optional) Um containerisierte Lambda-Funktionen auszuführen, müssen Sie cgroups v1 aktivieren und Sie müssen die Speicher - und Geräte-Cgroups aktivieren und mounten. Wenn Sie nicht vorhaben, containerisierte Lambda-Funktionen auszuführen, können Sie diesen Schritt überspringen.

   Um diese Cgroups-Optionen zu aktivieren, starten Sie das Gerät mit den folgenden Linux-Kernelparametern.

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

   Informationen zum Anzeigen und Einstellen der Kernel-Parameter für Ihr Gerät finden Sie in der Dokumentation zu Ihrem Betriebssystem und Bootloader. Folgen Sie den Anweisungen, um die Kernel-Parameter dauerhaft einzustellen.

5. Installieren Sie alle anderen erforderlichen Abhängigkeiten auf Ihrem Gerät, wie in der Liste der Anforderungen unter angegebenAnforderungen an Speichergeräte.

Richten Sie ein Windows-Gerät ein

> **Note**
>
> Diese Funktion ist für Version 2.5.0 und höher der Greengrass Nucleus-Komponente verfügbar.

So richten Sie ein Windows-Gerät ein für AWS IoT Greengrass V2

1. Installieren Sie die Java-Runtime, die für die Ausführung der AWS IoT Greengrass Core-Software erforderlich ist. Wir empfehlen, dass Sie die Langzeit-Support-Versionen von Amazon Corretto oder OpenJDK verwenden. Version 8 oder höher ist erforderlich.
2. Prüfen Sie, ob Java in der Systemvariablen PATH verfügbar ist, und fügen Sie es hinzu, falls nicht. Auf dem LocalSystem Konto wird die AWS IoT Greengrass Core-Software ausgeführt, sodass Sie der Systemvariablen PATH statt der Benutzervariablen PATH für Ihren Benutzer Java hinzufügen müssen. Gehen Sie wie folgt vor:

   a. Drücken Sie die Windows-Taste, um das Startmenü zu öffnen.

   b. Geben Sie **`environment variables`** ein, um im Startmenü nach den Systemoptionen zu suchen.

   c. Wählen Sie in den Suchergebnissen des Startmenüs die Option Systemumgebungsvariablen bearbeiten aus, um das Fenster mit den Systemeigenschaften zu öffnen.

   d. Wählen Sie Umgebungsvariablen... um das Fenster mit den Umgebungsvariablen zu öffnen.

   e. Wählen Sie unter Systemvariablen die Option Pfad und dann Bearbeiten aus. Im Fenster Umgebungsvariable bearbeiten können Sie jeden Pfad in einer separaten Zeile anzeigen.

   f. Überprüfen Sie, ob der Pfad zum `bin` Ordner der Java-Installation vorhanden ist. Der Pfad könnte dem folgenden Beispiel ähneln.

   ```
   C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
   ```

   g. Wenn der `bin` Ordner der Java-Installation in Path fehlt, wählen Sie Neu, um ihn hinzuzufügen, und klicken Sie dann auf OK.
3. Öffnen Sie die Windows-Eingabeaufforderung (`cmd.exe`) als Administrator.
4. Erstellen Sie den Standardbenutzer für das LocalSystem Konto auf dem Windows-Gerät. Ersetzen Sie *`das Passwort`* durch ein sicheres Passwort.

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

   > **Tip**
   >
   > Abhängig von Ihrer Windows-Konfiguration ist das Benutzerkennwort möglicherweise so eingestellt, dass es an einem Datum in der future abläuft. Um sicherzustellen, dass Ihre

Greengrass-Anwendungen weiterhin funktionieren, verfolgen Sie, wann das Passwort abläuft, und aktualisieren Sie es, bevor es abläuft. Sie können das Benutzerkennwort auch so einrichten, dass es niemals abläuft.

- Führen Sie den folgenden Befehl aus, um zu überprüfen, wann ein Benutzer und sein Passwort ablaufen.

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

- Führen Sie den folgenden Befehl aus, um das Passwort eines Benutzers so einzustellen, dass es nie abläuft.

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

- Wenn Sie Windows 10 oder höher verwenden und der wmicBefehl veraltet ist, führen Sie den folgenden PowerShell Befehl aus.

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

5. Laden Sie das PsExecProgramm von Microsoft herunter und installieren Sie es auf dem Gerät.
6. Verwenden Sie das PsExec Hilfsprogramm, um den Benutzernamen und das Passwort für den Standardbenutzer in der Credential Manager-Instanz für das LocalSystem Konto zu speichern. Ersetzen Sie *das Passwort* durch das Passwort des Benutzers, das Sie zuvor festgelegt haben.

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

Wenn das PsExec License Agreementgeöffnet wird, stimmen Sie Acceptder Lizenz zu und führen Sie den Befehl aus.

> **Note**
>
> Auf Windows-Geräten wird auf dem LocalSystem Konto der Greengrass-Nucleus ausgeführt, und Sie müssen das PsExec Hilfsprogramm verwenden, um die Standardbenutzerinformationen im LocalSystem Konto zu speichern. Wenn Sie die Credential Manager-Anwendung verwenden, werden diese Informationen nicht im

Konto, sondern im Windows-Konto des aktuell angemeldeten Benutzers gespeichert.
LocalSystem

Laden Sie die AWS IoT Greengrass Core-Software herunter

Sie können die neueste Version der AWS IoT Greengrass Core-Software von der folgenden Adresse herunterladen:

- https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip

> **Note**
>
> Sie können eine bestimmte Version der AWS IoT Greengrass Core-Software von der folgenden Adresse herunterladen. Ersetzen Sie die *Version* durch die Version, die Sie herunterladen möchten.
>
> ```
> https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
> ```

Um die AWS IoT Greengrass Core-Software herunterzuladen

1. Laden Sie die Core-Software auf Ihrem AWS IoT Greengrass Core-Gerät in eine Datei mit dem Namen herunter`greengrass-nucleus-latest.zip`.

   Linux or Unix

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
   latest.zip > greengrass-nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
   latest.zip > greengrass-nucleus-latest.zip
   ```

PowerShell

```
iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
latest.zip -OutFile greengrass-nucleus-latest.zip
```

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

2. (Optional) Um die Greengrass Nucleus-Softwaresignatur zu überprüfen

> **Note**
>
> Diese Funktion ist mit Greengrass Nucleus Version 2.9.5 und höher verfügbar.

a. Verwenden Sie den folgenden Befehl, um die Signatur Ihres Greengrass-Kernartefakts zu überprüfen:

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

Der Dateiname kann je nach installierter JDK-Version anders aussehen. Ersetzen Sie es *jdk17.0.6_10* durch die JDK-Version, die Sie installiert haben.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

Der Dateiname sieht je nach installierter JDK-Version möglicherweise anders aus. Ersetzen Sie es *jdk17.0.6_10* durch die JDK-Version, die Sie installiert haben.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -
verify -certs -verbose greengrass-nucleus-latest.zip
```

b. Der `jarsigner` Aufruf liefert eine Ausgabe, die die Ergebnisse der Überprüfung angibt.

i. Wenn die Greengrass Nucleus-Zip-Datei signiert ist, enthält die Ausgabe die folgende Anweisung:

```
jar verified.
```

ii. Wenn die Greengrass Nucleus-Zip-Datei nicht signiert ist, enthält die Ausgabe die folgende Anweisung:

```
jar is unsigned.
```

c. Wenn Sie die `-certs` Option Jarsigner zusammen mit den `-verbose` Optionen `-verify` und angegeben haben, enthält die Ausgabe auch detaillierte Informationen zum Unterzeichnerzertifikat.

3. Entpacken Sie die AWS IoT Greengrass Core-Software in einen Ordner auf Ihrem Gerät. *`GreengrassInstaller`*Ersetzen Sie es durch den Ordner, den Sie verwenden möchten.

   Linux or Unix

   ```
   unzip greengrass-nucleus-latest.zip -d GreengrassInstaller && rm greengrass-
   nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir GreengrassInstaller && tar -xf greengrass-nucleus-latest.zip -
   C GreengrassInstaller && del greengrass-nucleus-latest.zip
   ```

   PowerShell

   ```
   Expand-Archive -Path greengrass-nucleus-latest.zip -DestinationPath .\
   \GreengrassInstaller
   rm greengrass-nucleus-latest.zip
   ```

4. (Optional) Führen Sie den folgenden Befehl aus, um die Version der AWS IoT Greengrass Core-Software zu sehen.

   ```
   java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
   ```

> **Important**
>
> Wenn Sie eine Version von Greengrass Nucleus vor v2.4.0 installieren, entfernen Sie diesen Ordner nicht, nachdem Sie die AWS IoT Greengrass Core-Software installiert haben. Die AWS IoT Greengrass Core-Software verwendet die Dateien in diesem Ordner zur Ausführung.
> Wenn Sie die neueste Version der Software heruntergeladen haben, installieren Sie v2.4.0 oder höher, und Sie können diesen Ordner entfernen, nachdem Sie die AWS IoT Greengrass Core-Software installiert haben.

Installieren Sie die Core-Software AWS IoT Greengrass

Führen Sie das Installationsprogramm mit Argumenten aus, die die folgenden Aktionen angeben:

- Installieren Sie aus einer Teilkonfigurationsdatei, in der angegeben ist, dass Sie Ihr benutzerdefiniertes Provisioning-Plugin für die Bereitstellung von AWS Ressourcen verwenden möchten. Die AWS IoT Greengrass Core-Software verwendet eine Konfigurationsdatei, die die Konfiguration jeder Greengrass-Komponente auf dem Gerät spezifiziert. Das Installationsprogramm erstellt eine vollständige Konfigurationsdatei aus der von Ihnen bereitgestellten Teilkonfigurationsdatei und den AWS Ressourcen, die das benutzerdefinierte Provisioning-Plugin erstellt.
- Geben Sie an, dass der `ggc_user` Systembenutzer Softwarekomponenten auf dem Kerngerät ausführen soll. Auf Linux-Geräten gibt dieser Befehl auch an, dass die `ggc_group` Systemgruppe verwendet werden soll, und das Installationsprogramm erstellt den Systembenutzer und die Systemgruppe für Sie.
- Richten Sie die AWS IoT Greengrass Core-Software als Systemdienst ein, der beim Booten ausgeführt wird. Auf Linux-Geräten erfordert dies das Systemd-Init-System.

> **Important**
>
> Auf Windows Core-Geräten müssen Sie die AWS IoT Greengrass Core-Software als Systemdienst einrichten.

Weitere Hinweise zu den Argumenten, die Sie angeben können, finden Sie unterInstaller-Argumente.

> **Note**
>
> Wenn Sie AWS IoT Greengrass auf einem Gerät mit begrenztem Arbeitsspeicher arbeiten, können Sie die Speichermenge steuern, die die AWS IoT Greengrass Core-Software verwendet. Um die Speicherzuweisung zu steuern, können Sie die Optionen für die JVM-Heap-Größe im `jvmOptions` Konfigurationsparameter Ihrer Nucleus-Komponente festlegen. Weitere Informationen finden Sie unter Steuern Sie die Speicherzuweisung mit JVM-Optionen.

So installieren Sie die AWS IoT Greengrass Core-Software (Linux)

1. Überprüfen Sie die Version der AWS IoT Greengrass Core-Software.

   - *`GreengrassInstaller`*Ersetzen Sie es durch den Pfad zu dem Ordner, der die Software enthält.

   ```
   java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
   ```

2. Verwenden Sie einen Texteditor, um eine Konfigurationsdatei mit dem Namen `config.yaml` zu erstellen, die dem Installationsprogramm zur Verfügung gestellt werden soll.

   Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um die Datei mit GNU Nano zu erstellen.

   ```
   nano GreengrassInstaller/config.yaml
   ```

   Kopieren Sie den folgenden YAML-Inhalt in die Datei.

   ```
   ---
   system:
     rootpath: "/greengrass/v2"
     # The following values are optional. Return them from the provisioning plugin or
    set them here.
     # certificateFilePath: ""
     # privateKeyPath: ""
     # rootCaPath: ""
     # thingName: ""
   services:
   ```

```
 aws.greengrass.Nucleus:
   version: "2.12.6"
   configuration:
     # The following values are optional. Return them from the provisioning plugin
or set them here.
     # awsRegion: ""
     # iotRoleAlias: ""
     # iotDataEndpoint: ""
     # iotCredEndpoint: ""
 com.example.CustomProvisioning:
   configuration:
     # You can specify configuration parameters to provide to your plugin.
     # pluginParameter: ""
```

Führen Sie dann die folgenden Schritte aus:

- Ersetzen Sie *2.12.6* durch die Version der AWS IoT Greengrass Core-Software.
- Ersetzen Sie jede Instanz von */greengrass/v2* durch den Greengrass-Stammordner.
- (Optional) Geben Sie System- und Nucleus-Konfigurationswerte an. Sie müssen diese Werte festlegen, wenn Ihr Provisioning-Plugin sie nicht bereitstellt.
- (Optional) Geben Sie die Konfigurationsparameter an, die für Ihr Provisioning-Plugin bereitgestellt werden sollen.

> **Note**
>
> In dieser Konfigurationsdatei können Sie weitere Konfigurationsoptionen anpassen, z. B. die zu verwendenden Ports und den Netzwerk-Proxy, wie im folgenden Beispiel gezeigt. Weitere Informationen finden Sie unter Greengrass Nucleus-Konfiguration.
>
> ```
> ---
> system:
>   rootpath: "/greengrass/v2"
>   # The following values are optional. Return them from the provisioning
>  plugin or set them here.
>   # certificateFilePath: ""
>   # privateKeyPath: ""
>   # rootCaPath: ""
>   # thingName: ""
> services:
>   aws.greengrass.Nucleus:
> ```

```
    version: "2.12.6"
    configuration:
      mqtt:
        port: 443
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "http://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
      # The following values are optional. Return them from the provisioning
 plugin or set them here.
      # awsRegion: ""
      # iotRoleAlias: ""
      # iotDataEndpoint: ""
      # iotCredEndpoint: ""
  com.example.CustomProvisioning:
    configuration:
      # You can specify configuration parameters to provide to your plugin.
      # pluginParameter: ""
```

3. Führen Sie das Installationsprogramm aus. Geben Sie `--trusted-plugin` an, ob Sie Ihr benutzerdefiniertes Provisioning-Plugin bereitstellen möchten, und geben Sie `--init-config` an, dass Sie die Konfigurationsdatei bereitstellen möchten.

   - Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Greengrass-Stammordner.
   - Ersetzen Sie jede Instanz von *GreengrassInstaller*durch den Ordner, in den Sie das Installationsprogramm entpackt haben.
   - Ersetzen Sie den Pfad zur JAR-Datei des benutzerdefinierten Provisioning-Plugins durch den Pfad zur JAR-Datei Ihres Plugins.

   Linux or Unix

   ```
   sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
     -jar ./GreengrassInstaller/lib/Greengrass.jar \
     --trusted-plugin /path/to/com.example.CustomProvisioning.jar \
     --init-config ./GreengrassInstaller/config.yaml \
     --component-default-user ggc_user:ggc_group \
   ```

```
  --setup-system-service true
```

Windows Command Prompt (CMD)

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
  -jar ./GreengrassInstaller/lib/Greengrass.jar ^
  --trusted-plugin /path/to/com.example.CustomProvisioning.jar ^
  --init-config ./GreengrassInstaller/config.yaml ^
  --component-default-user ggc_user ^
  --setup-system-service true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
  -jar ./GreengrassInstaller/lib/Greengrass.jar `
  --trusted-plugin /path/to/com.example.CustomProvisioning.jar `
  --init-config ./GreengrassInstaller/config.yaml `
  --component-default-user ggc_user `
  --setup-system-service true
```

Important

Auf Windows Core-Geräten müssen Sie angeben, `--setup-system-service true` dass die AWS IoT Greengrass Core-Software als Systemdienst eingerichtet werden soll.

Wenn Sie dies angeben`--setup-system-service true`, gibt das Installationsprogramm aus, `Successfully set up Nucleus as a system service` ob es die Software als Systemdienst eingerichtet und ausgeführt hat. Andernfalls gibt das Installationsprogramm keine Meldung aus, wenn es die Software erfolgreich installiert hat.

Note

Sie können das `deploy-dev-tools` Argument nicht verwenden, um lokale Entwicklungstools bereitzustellen, wenn Sie das Installationsprogramm ohne das `--provision true` Argument ausführen. Informationen zur direkten Bereitstellung der Greengrass-CLI auf Ihrem Gerät finden Sie unterGreengrass-Befehlszeilenschnittstelle.

4. Überprüfen Sie die Installation, indem Sie sich die Dateien im Stammordner ansehen.

   Linux or Unix

   ```
   ls /greengrass/v2
   ```

   Windows Command Prompt (CMD)

   ```
   dir C:\greengrass\v2
   ```

   PowerShell

   ```
   ls C:\greengrass\v2
   ```

   Wenn die Installation erfolgreich war, enthält der Stammordner mehrere Ordner, z. B. `configpackages`, und`logs`.

Wenn Sie die AWS IoT Greengrass Core-Software als Systemdienst installiert haben, führt das Installationsprogramm die Software für Sie aus. Andernfalls müssen Sie die Software manuell ausführen. Weitere Informationen finden Sie unter Ausführen der AWS IoT Greengrass -Core-Software.

Weitere Informationen zur Konfiguration und Verwendung der Software finden Sie unter: AWS IoT Greengrass

- Konfigurieren Sie die AWS IoT Greengrass Core-Software
- Entwickeln von AWS IoT Greengrass Komponenten
- Bereitstellen von AWS IoT Greengrass Komponenten auf Geräten
- Greengrass-Befehlszeilenschnittstelle

Entwickeln Sie benutzerdefinierte Provisioning-Plugins

Um ein benutzerdefiniertes Provisioning-Plugin zu entwickeln, erstellen Sie eine Java-Klasse, die die`com.aws.greengrass.provisioning.DeviceIdentityInterface`-Schnittstelle implementieren. Sie können die Greengrass Nucleus JAR-Datei in Ihr Projekt aufnehmen, um auf diese Schnittstelle und ihre Klassen zuzugreifen. Diese Schnittstelle definiert eine Methode, die

eine Plugin-Konfiguration eingibt und eine Provisioning-Konfiguration ausgibt. Die Provisioning-Konfiguration definiert Konfigurationen für das System und dieGreengrass Kernkomponenteaus. DieAWS IoT GreengrassCore-Software-Installer verwendet diese Provisioning-Konfiguration, um dieAWS IoT GreengrassKernsoftware auf einem Gerät.

Nachdem Sie ein benutzerdefiniertes Provisioning-Plugin entwickelt haben, erstellen Sie es als JAR-Datei, die Sie demAWS IoT GreengrassKernsoftware-Installer, um Ihr Plugin während der Installation auszuführen. Das Installationsprogramm führt Ihr benutzerdefiniertes Provisioning-Plugin in derselben JVM aus, die das Installationsprogramm verwendet, sodass Sie ein JAR erstellen können, das nur Ihren Plugin-Code enthält.

> **Note**
>
> DieAWS IoTFlottenbereitstellungs-Pluginimplementiert die`DeviceIdentityInterface`um Flottenbereitstellung während der Installation zu verwenden. Das Plugin für die Flottenbereitstellung ist Open Source, sodass Sie den Quellcode untersuchen können, um ein Beispiel für die Verwendung der Provisioning-Plugin-Schnittstelle zu sehen. Weitere Informationen finden Sie im .AWS IoTFlottenbereitstellungs-Pluginauf GitHub:

Themen

Voraussetzungen

Um ein benutzerdefiniertes Provisioning-Plugin zu entwickeln, müssen Sie eine Java-Klasse erstellen, die die folgenden Anforderungen erfüllt:

- Verwendet die`com.aws.greengrass`Paket oder ein Paket innerhalb der`com.aws.greengrass`-Paket.
- Hat einen Konstruktor ohne Argumente.
- Implementiert die`DeviceIdentityInterface`-Schnittstelle implementieren. Weitere Informationen finden Sie unter Implementieren Sie DeviceIdentityInterface Schnittstelle.

Implementieren Sie DeviceIdentityInterface Schnittstelle

So verwenden Sie dencom.aws.greengrass.provisioning.DeviceIdentityInterfacefügen Sie in Ihrem benutzerdefinierten Plugin den Greengrass-Kern als Abhängigkeit zu Ihrem Projekt hinzu.

So verwenden Sie den DeviceIdentityInterface in einem benutzerdefinierten Provisioning-Plugin-Projekt

- Sie können die Greengrass-Kern-JAR-Datei als Bibliothek hinzufügen oder den Greengrass-Kern als Maven-Abhängigkeit hinzufügen. Führen Sie eine der folgenden Aktionen aus:
  - Um die Greengrass Nucleus JAR-Datei als Bibliothek hinzuzufügen, laden Sie dieAWS IoT GreengrassKernsoftware, die den Greengrass-Kern JAR enthält. Sie können die neueste Version vonAWS IoT GreengrassCore-Software von folgendem Speicherort aus:
    - https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip

    Sie finden die Greengrass Nucleus JAR-Datei (`Greengrass.jar`) im`lib`-Ordner in der ZIP-Datei. Fügen Sie diese JAR-Datei zu Ihrem Projekt hinzu.
  - Um den Greengrass-Kern in einem Maven-Projekt zu konsumieren, fügen Sie eine Abhängigkeit von`nucleus`-Artefakt im`com.aws.greengrass`Gruppe. Sie müssen auch die`greengrass-common`Repository, da der Greengrass-Kern im Maven Central Repository nicht verfügbar ist.

    ```
    <project ...>
        ...
        <repositories>
            <repository>
                <id>greengrass-common</id>
                <name>greengrass common</name>
                <url>https://d2jrmugq4soldf.cloudfront.net/snapshots</url>
            </repository>
        </repositories>
        ...
        <dependencies>
            <dependency>
                <groupId>com.aws.greengrass</groupId>
                <artifactId>nucleus</artifactId>
                <version>2.5.0-SNAPSHOT</version>
                <scope>provided</scope>
    ```

```
        </dependency>
    </dependencies>
</project>
```

Die DeviceIdentityInterface-Schnittstelle

Die`com.aws.greengrass.provisioning.DeviceIdentityInterface`-Schnittstelle hat die folgende Form.

> **Note**
>
> Sie können diese Kurse auch imcom.aws.greengrass.provisioning-PaketderGreengrass Nucleus-Quellcodeauf GitHub:

```
public interface com.aws.greengrass.provisioning.DeviceIdentityInterface {
    ProvisionConfiguration updateIdentityConfiguration(ProvisionContext context)
            throws RetryableProvisioningException, InterruptedException;

    // Return the name of the plugin.
    String name();
}

com.aws.greengrass.provisioning.ProvisionConfiguration {
    SystemConfiguration systemConfiguration;
    NucleusConfiguration nucleusConfiguration
}

com.aws.greengrass.provisioning.ProvisionConfiguration.SystemConfiguration {
    String certificateFilePath;
    String privateKeyPath;
    String rootCAPath;
    String thingName;
}

com.aws.greengrass.provisioning.ProvisionConfiguration.NucleusConfiguration {
    String awsRegion;
    String iotCredentialsEndpoint;
    String iotDataEndpoint;
    String iotRoleAlias;
}
```

```
com.aws.greengrass.provisioning.ProvisioningContext {
    Map<String, Object> parameterMap;
    String provisioningPolicy;  // The policy is always "PROVISION_IF_NOT_PROVISIONED".
}

com.aws.greengrass.provisioning.exceptions.RetryableProvisioningException {}
```

Jeder Konfigurationswert im`SystemConfiguration`und`NucleusConfiguration`ist erforderlich, um dasAWS IoT GreengrassCore-Software, aber Sie können zurückkehren`null`aus. Wenn Ihr benutzerdefiniertes Provisioning-Plugin zurückkehrt`null`für jeden Konfigurationswert müssen Sie diesen Wert in der System- oder Kernkonfiguration angeben, wenn Sie die`config.yaml`zur Verfügung zu stellende DateiAWS IoT GreengrassCore-Software-Installationsprogramm. Wenn Ihr benutzerdefiniertes Provisioning-Plugin einen Wert ungleich Null für eine Option zurückgibt, die Sie auch in`config.yaml`, dann ersetzt das Installationsprogramm den Wert in`config.yaml`mit dem vom Plugin zurückgegebenen Wert.

Installer-Argumente

Die -AWS IoT GreengrassCore-Software enthält ein Installationsprogramm, das die Software einrichtet und die erforderlichen AWS Ressourcen für die Ausführung des Greengrass-Core-Geräts bereitstellt. Das Installationsprogramm enthält die folgenden Argumente, die Sie zur Konfiguration der Installation angeben können:

`-h`, `--help`

(Optional) Zeigen Sie die Hilfeinformationen des Installationsprogramms an.

`--version`

(Optional) Zeigen Sie die Version der -AWS IoT GreengrassCore-Software an.

`-Droot`

(Optional) Der Pfad zum Ordner, der als Stamm für die AWS IoT Greengrass Core-Software verwendet werden soll.

> **Note**
>
> Dieses Argument legt eine JVM-Eigenschaft fest, daher müssen Sie sie zuvor angeben, `-jar` wenn Sie das Installationsprogramm ausführen. Geben Sie beispielsweise `java -Droot="/greengrass/v2" -jar /path/to/Greengrass.jar` an.

Standard:

- Linux: `~/.greengrass`
- Windows: `%USERPROFILE%/.greengrass`

`-ar`, `--aws-region`

Die AWS-Region, die die AWS IoT Greengrass -Core-Software zum Abrufen oder Erstellen ihrer erforderlichen AWS Ressourcen verwendet.

`-p`, `--provision`

(Optional) Sie können dieses Gerät als -AWS IoTObjekt registrieren und die AWS Ressourcen bereitstellen, die das Core-Gerät benötigt. Wenn Sie angeben`true`, stellt die -AWS IoT GreengrassCore-Software ein -AWS IoTObjekt, (optional) eine -AWS IoTObjektgruppe, eine IAM-Rolle und einen -AWS IoTRollenalias bereit.

Standard: `false`

`-tn`, `--thing-name`

(Optional) Der Name des AWS IoT Objekts, das Sie als dieses Core-Gerät registrieren. Wenn das Objekt mit dem Namen in Ihrem nicht vorhanden istAWS-Konto, erstellt die AWS IoT Greengrass Core-Software es.

> **Note**
>
> Der Objektname darf keine Doppelpunktzeichen (`:`) enthalten.

Sie müssen angeben`--provision true`, um dieses Argument anzuwenden.

Standard: `GreengrassV2IotThing_` plus eine zufällige UUID.

`-tgn`, `--thing-group-name`

(Optional) Der Name der AWS IoT Objektgruppe, zu der Sie das AWS IoT Objekt dieses Core-Geräts hinzufügen. Wenn eine Bereitstellung auf diese Objektgruppe abzielt, erhält dieses Core-Gerät diese Bereitstellung, wenn es eine Verbindung zu herstelltAWS IoT Greengrass. Wenn die Objektgruppe mit diesem Namen in Ihrem nicht vorhanden istAWS-Konto, erstellt die AWS IoT Greengrass Core-Software sie.

> **Note**
>
> Der Objektgruppenname darf keine Doppelpunktzeichen (`:`) enthalten.

Sie müssen angeben`--provision true`, um dieses Argument anzuwenden.

`-tpn`, `--thing-policy-name`

Diese Funktion ist für v2.4.0 und höher der Greengrass-Kernkomponente verfügbar.

(Optional) Der Name der AWS IoT Richtlinie, die an das AWS IoT Objektzertifikat dieses Core-Geräts angehängt werden soll. Wenn die AWS IoT Richtlinie mit diesem Namen in Ihrem nicht vorhanden istAWS-Konto, erstellt die AWS IoT Greengrass Core-Software sie.

Die AWS IoT Greengrass -Core-Software erstellt standardmäßig eine zulässige AWS IoT Richtlinie. Sie können diese Richtlinie einschränken oder eine benutzerdefinierte Richtlinie erstellen, in der Sie die Berechtigungen für Ihren Anwendungsfall einschränken. Weitere Informationen finden Sie unter Minimale AWS IoT Richtlinie für -AWS IoT Greengrass V2Core-Geräte.

Sie müssen angeben`--provision true`, um dieses Argument anzuwenden.

Standard: `GreengrassV2IoTThingPolicy`

`-trn`, `--tes-role-name`

(Optional) Der Name der IAM-Rolle, die zum Abrufen von AWS Anmeldeinformationen verwendet werden soll, mit denen das Core-Gerät mit -AWSServices interagieren kann. Wenn die Rolle mit diesem Namen in Ihrem nicht vorhanden istAWS-Konto, erstellt die AWS IoT Greengrass Core-Software sie mit der `GreengrassV2TokenExchangeRoleAccess` Richtlinie . Diese Rolle hat keinen Zugriff auf Ihre S3-Buckets, in denen Sie Komponentenartefakte hosten. Daher müssen Sie den S3-Buckets und -Objekten Ihrer Artefakte Berechtigungen hinzufügen, wenn Sie eine

Komponente erstellen. Weitere Informationen finden Sie unter [Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS](#).

Sie müssen angeben`--provision true`, um dieses Argument anzuwenden.

Standard: `GreengrassV2TokenExchangeRole`

`-tra`, `--tes-role-alias-name`

(Optional) Der Name des AWS IoT Rollenalias, der auf die IAM-Rolle verweist, die AWS Anmeldeinformationen für dieses Core-Gerät bereitstellt. Wenn der Rollenalias mit diesem Namen in Ihrem nicht vorhanden istAWS-Konto, erstellt die -AWS IoT GreengrassCore-Software ihn und verweist ihn auf die von Ihnen angegebene IAM-Rolle.

Sie müssen angeben`--provision true`, um dieses Argument anzuwenden.

Standard: `GreengrassV2TokenExchangeRoleAlias`

`-ss`, `--setup-system-service`

(Optional) Sie können die AWS IoT Greengrass -Core-Software als Systemservice einrichten, der beim Start dieses Geräts ausgeführt wird. Der Systemservicename lautet `greengrass`. Weitere Informationen finden Sie unter [Den Greengrass Nucleus als Systemdienst konfigurieren](#).

Unter Linux-Betriebssystemen erfordert dieses Argument, dass das systemd init-System auf dem Gerät verfügbar ist.

> **Important**
>
> Auf Windows-Core-Geräten müssen Sie die AWS IoT Greengrass Core-Software als Systemservice einrichten.

Standard: `false`

`-u`, `--component-default-user`

Der Name oder die ID des Benutzers, den die AWS IoT Greengrass Core-Software zum Ausführen von Komponenten verwendet. Sie können beispielsweise **`ggc_user`** angeben. Dieser Wert ist erforderlich, wenn Sie das Installationsprogramm unter Windows-Betriebssystemen ausführen.

Unter Linux-Betriebssystemen können Sie optional auch die Gruppe angeben. Geben Sie den Benutzer und die Gruppe durch einen Doppelpunkt getrennt an. Beispiel: **`ggc_user:ggc_group`**

Für Linux-Betriebssysteme gelten die folgenden zusätzlichen Überlegungen:

- Wenn Sie als Stamm ausführen, ist der Standardkomponentenbenutzer der Benutzer, der in der Konfigurationsdatei definiert ist. Wenn die Konfigurationsdatei keinen Benutzer definiert, ist dies standardmäßig `ggc_user:ggc_group`. Wenn `ggc_user` oder `ggc_group` nicht vorhanden sind, erstellt die Software sie.
- Wenn Sie als Nicht-Root-Benutzer ausführen, verwendet die AWS IoT Greengrass -Core-Software diesen Benutzer, um Komponenten auszuführen.
- Wenn Sie keine Gruppe angeben, verwendet die AWS IoT Greengrass Core-Software die primäre Gruppe des Systembenutzers.

Weitere Informationen finden Sie unter Konfigurieren Sie den Benutzer, der die Komponenten ausführt.

`-d`, `--deploy-dev-tools`

(Optional) Sie können die Greengrass-CLI-Komponente herunterladen und auf diesem Core-Gerät bereitstellen. Sie können dieses Tool verwenden, um Komponenten auf diesem Core-Gerät zu entwickeln und zu debuggen.

> **Important**
>
> Wir empfehlen, diese Komponente nur in Entwicklungsumgebungen und nicht in Produktionsumgebungen zu verwenden. Diese Komponente bietet Zugriff auf Informationen und Vorgänge, die Sie normalerweise in einer Produktionsumgebung nicht benötigen. Folgen Sie dem Prinzip der geringsten Berechtigung, indem Sie diese Komponente nur auf Core-Geräten bereitstellen, auf denen Sie sie benötigen.

Sie müssen angeben`--provision true`, um dieses Argument anzuwenden.

Standard: `false`

`-init`, `--init-config`

(Optional) Der Pfad zur Konfigurationsdatei, die zur Installation der AWS IoT Greengrass Core-Software verwendet werden soll. Sie können diese Option verwenden, um beispielsweise neue Core-Geräte mit einer bestimmten Kernkonfiguration einzurichten.

> **Important**
>
> Die von Ihnen angegebene Konfigurationsdatei wird mit der vorhandenen Konfigurationsdatei auf dem Core-Gerät zusammengeführt. Dazu gehören die Komponenten und Komponentenkonfigurationen auf dem Core-Gerät. Wir empfehlen, dass die Konfigurationsdatei nur die Konfigurationen auflistet, die Sie ändern möchten.

`-tp`, `--trusted-plugin`

(Optional) Der Pfad zu einer JAR-Datei, die als vertrauenswürdiges Plugin geladen werden soll. Verwenden Sie diese Option, um Bereitstellungs-Plugin-JAR-Dateien bereitzustellen, z. B. um mit Flottenbereitstellung oder benutzerdefinierte Bereitstellung zu installieren, oder um mit dem privaten Schlüssel und dem Zertifikat in einem Hardware-Sicherheitsmodul zu installieren.

`-s`, `--start`

(Optional) Sie können die AWS IoT Greengrass -Core-Software starten, nachdem sie installiert und optional Ressourcen bereitgestellt hat.

Standard: `true`

Ausführen der AWS IoT Greengrass -Core-Software

Nachdem Sie die AWS IoT Greengrass Core-Software installiert haben, führen Sie sie aus, um Ihr Gerät mit zu verbindenAWS IoT Greengrass.

Wenn Sie die AWS IoT Greengrass Core-Software installieren, können Sie angeben, ob sie als Systemservice mit systemd installiert werden soll. Wenn Sie diese Option wählen, führt das Installationsprogramm die Software für Sie aus und konfiguriert sie so, dass sie beim Start Ihres Geräts ausgeführt wird.

> **Important**
>
> Auf Windows-Core-Geräten müssen Sie die AWS IoT Greengrass Core-Software als Systemservice einrichten.

Themen

Prüfen, ob die AWS IoT Greengrass Core-Software als Systemservice ausgeführt wird

Wenn Sie die AWS IoT Greengrass Core-Software installieren, können Sie das `--setup-system-service true` Argument angeben, um die AWS IoT Greengrass Core-Software als Systemservice zu installieren. Linux-Geräte erfordern das systemd init-System, um die AWS IoT Greengrass Core-Software als Systemservice einzurichten. Wenn Sie diese Option verwenden, führt das Installationsprogramm die Software für Sie aus und konfiguriert sie so, dass sie beim Start Ihres Geräts ausgeführt wird. Das Installationsprogramm gibt die folgende Meldung aus, wenn es die AWS IoT Greengrass -Core-Software erfolgreich als Systemservice installiert.

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

Wenn Sie zuvor die AWS IoT Greengrass Core-Software installiert haben und nicht über die Installationsprogrammausgabe verfügen, können Sie überprüfen, ob die Software als Systemservice installiert ist.

So überprüfen Sie, ob die AWS IoT Greengrass -Core-Software als Systemservice installiert ist

- Führen Sie den folgenden Befehl aus, um den Status des Greengrass-Systemservices zu überprüfen.

  Linux or Unix (systemd)

  ```
  sudo systemctl status greengrass.service
  ```

  Die Antwort ähnelt dem folgenden Beispiel, wenn die AWS IoT Greengrass Core-Software als Systemservice installiert und aktiv ist.

  ```
  # greengrass.service - Greengrass Core
     Loaded: loaded (/etc/systemd/system/greengrass.service; enabled; vendor
   preset: disabled)
     Active: active (running) since Thu 2021-02-11 01:33:44 UTC; 4 days ago
   Main PID: 16107 (sh)
  ```

```
   CGroup: /system.slice/greengrass.service
           ##16107 /bin/sh /greengrass/v2/alts/current/distro/bin/loader
           ##16111 java -Dlog.store=FILE -Droot=/greengrass/v2 -jar /greengrass/
v2/alts/current/distro/lib/Greengrass...
```

Wenn `systemctl` oder nicht gefunden `greengrass.service` wird, wird die -AWS IoT GreengrassCore-Software nicht als Systemservice installiert. Informationen zum Ausführen der Software finden Sie unter [Ausführen der AWS IoT Greengrass Core-Software ohne Systemservice](#).

Windows Command Prompt (CMD)

```
sc query greengrass
```

Die Antwort ähnelt dem folgenden Beispiel, wenn die AWS IoT Greengrass Core-Software als Windows-Service installiert und aktiv ist.

```
SERVICE_NAME: greengrass
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 4  RUNNING
                                (STOPPABLE, NOT_PAUSABLE, ACCEPTS_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
```

PowerShell

```
Get-Service greengrass
```

Die Antwort ähnelt dem folgenden Beispiel, wenn die AWS IoT Greengrass Core-Software als Windows-Service installiert und aktiv ist.

```
Status   Name               DisplayName
------   ----               -----------
Running  greengrass         greengrass
```

Ausführen der AWS IoT Greengrass Core-Software als Systemservice

Wenn die AWS IoT Greengrass -Core-Software als Systemservice installiert ist, können Sie den Systemservicemanager verwenden, um die Software zu starten, zu stoppen und zu verwalten. Weitere Informationen finden Sie unter Den Greengrass Nucleus als Systemdienst konfigurieren.

So führen Sie die AWS IoT Greengrass -Core-Software aus

- Führen Sie den folgenden Befehl aus, um die AWS IoT Greengrass Core-Software zu starten.

  Linux or Unix (systemd)

  ```
  sudo systemctl start greengrass.service
  ```

  Windows Command Prompt (CMD)

  ```
  sc start greengrass
  ```

  PowerShell

  ```
  Start-Service greengrass
  ```

Ausführen der AWS IoT Greengrass Core-Software ohne Systemservice

Wenn die AWS IoT Greengrass Core-Software auf Linux-Core-Geräten nicht als Systemservice installiert ist, können Sie das Loader-Skript der Software ausführen, um die Software auszuführen.

So führen Sie die AWS IoT Greengrass Core-Software ohne Systemservice aus

- Führen Sie den folgenden Befehl aus, um die AWS IoT Greengrass Core-Software zu starten. Wenn Sie diesen Befehl in einem Terminal ausführen, müssen Sie die Terminalsitzung geöffnet lassen, damit die AWS IoT Greengrass Core-Software weiter ausgeführt wird.
  - Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den von Ihnen verwendeten Greengrass-Stammordner.

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

Die Software gibt die folgende Meldung aus, wenn sie erfolgreich gestartet wird.

```
Launched Nucleus successfully.
```

Ausführen von AWS IoT Greengrass Core-Software in einem Docker-Container

AWS IoT Greengrass kann für die Ausführung in einem Docker-Container konfiguriert werden. Docker ist eine Plattform, die Ihnen die Tools zum Erstellen, Ausführen, Testen und Bereitstellen von Anwendungen bereitstellt, die auf Linux-Containern basieren. Wenn Sie ein AWS IoT Greengrass -Docker-Image ausführen, können Sie wählen, ob Sie Ihre AWS Anmeldeinformationen für den Docker-Container bereitstellen möchten, und dem AWS IoT Greengrass -Core-Softwareinstallationsprogramm erlauben, automatisch die AWS Ressourcen bereitzustellen, die ein Greengrass-Core-Gerät für den Betrieb benötigt. Wenn Sie keine Anmeldeinformationen bereitstellen AWS möchten, können Sie AWS Ressourcen manuell bereitstellen und AWS IoT Greengrass Core-Software im Docker-Container ausführen.

Themen

Unterstützte Plattformen und Anforderungen

Host-Computer müssen die folgenden Mindestanforderungen erfüllen, um die AWS IoT Greengrass Core-Software in einem Docker-Container zu installieren und auszuführen:

- Ein Linux-basiertes Betriebssystem mit einer Internetverbindung.

- Docker-Engine-Version 18.09 oder höher.
- (Optional) Docker Compose Version 1.22 oder höher. Docker Compose ist nur erforderlich, wenn Sie die Docker Compose CLI zum Ausführen Ihrer Docker-Images verwenden möchten.

Um Lambda-Funktionskomponenten innerhalb des Docker-Containers auszuführen, müssen Sie den Container so konfigurieren, dass er zusätzliche Anforderungen erfüllt. Weitere Informationen finden Sie unter Anforderungen an die Lambda-Funktion.

Ausführen von Komponenten im Prozessmodus

AWS IoT Greengrass unterstützt nicht die Ausführung von Lambda AWS-Funktionen oder von bereitgestellten Komponenten in einer isolierten Laufzeitumgebung innerhalb des AWS IoT Greengrass Docker-Containers. Sie müssen diese Komponenten im Prozessmodus ohne Isolierung ausführen.

Wenn Sie eine Lambda-Funktionskomponente konfigurieren, legen Sie den Isolationsmodus auf Kein Container fest. Weitere Informationen finden Sie unter Ausführen von -AWS LambdaFunktionen.

Wenn Sie eine der folgenden von AWSbereitgestellten Komponenten bereitstellen, aktualisieren Sie die Konfiguration für jede Komponente, um den `containerMode` Parameter auf festzulegen`NoContainer`. Weitere Informationen zu Konfigurationsaktualisierungen finden Sie unter Komponentenkonfigurationen aktualisieren.

- CloudWatch -Metriken
- Device Defender
- Firehose
- Adapter für das Bol-RTU-Protokoll
- Amazon SNS

AWS IoT Greengrass Downloads von Docker-Software

AWS IoT Greengrass stellt ein Dockerfile bereit, um ein Container-Image zu erstellen, auf dem AWS IoT Greengrass Core-Software und Abhängigkeiten auf einem Amazon Linux 2 (x86_64)-Basis-Image installiert sind. Sie können das Basis-Image im Dockerfile so ändern, dass es AWS IoT Greengrass auf einer anderen Plattformarchitektur ausgeführt wird.

Laden Sie das Dockerfile-Paket von herunterGitHub.

Das Dockerfile verwendet eine ältere Version von Greengrass. Sie sollten die Datei aktualisieren, um die gewünschte Version von Greengrass zu verwenden. Informationen zum Erstellen des AWS IoT Greengrass Container-Images aus der Dockerfile-Datei finden Sie unter Erstellen des AWS IoT Greengrass Container-Images aus einer Dockerfile-Datei.

Auswählen der Bereitstellung AWS von Ressourcen

Wenn Sie die - AWS IoT Greengrass Core-Software in einem Docker-Container installieren, können Sie wählen, ob die AWS Ressourcen, die ein Greengrass-Core-Gerät für den Betrieb benötigt, automatisch bereitgestellt oder Ressourcen verwendet werden sollen, die Sie manuell bereitstellen.

- Automatische Ressourcenbereitstellung – Das Installationsprogramm stellt das AWS IoT Objekt, die AWS IoT Objektgruppe, die IAM-Rolle und den AWS IoT Rollenalias bereit, wenn Sie das AWS IoT Greengrass Container-Image zum ersten Mal ausführen. Das Installationsprogramm kann auch die lokalen Entwicklungstools auf dem Core-Gerät bereitstellen, sodass Sie das Gerät verwenden können, um benutzerdefinierte Softwarekomponenten zu entwickeln und zu testen. Um diese Ressourcen automatisch bereitzustellen, müssen Sie Anmeldeinformationen als Umgebungsvariablen für das Docker-Image bereitstellen AWS .

  Um die automatische Bereitstellung zu verwenden, müssen Sie die Docker-Umgebungsvariable festlegen `PROVISION=true` und eine Datei mit Anmeldeinformationen bereitstellen, um Ihre AWS Anmeldeinformationen für den Container bereitzustellen.
- Manuelle Ressourcenbereitstellung – Wenn Sie dem Container keine AWS Anmeldeinformationen bereitstellen möchten, können Sie die AWS Ressourcen manuell bereitstellen, bevor Sie das AWS IoT Greengrass Container-Image ausführen. Sie müssen eine Konfigurationsdatei erstellen, um Informationen zu diesen Ressourcen für das AWS IoT Greengrass Core-Softwareinstallationsprogramm im Docker-Container bereitzustellen.

  Um die manuelle Bereitstellung zu verwenden, müssen Sie die Docker-Umgebungsvariable festlegen`PROVISION=false`. Die manuelle Bereitstellung ist die Standardoption.

Weitere Informationen finden Sie unter Erstellen des AWS IoT Greengrass Container-Images aus einer Dockerfile-Datei.

Erstellen des AWS IoT Greengrass Container-Images aus einer Dockerfile-Datei

AWS stellt eine Docker-Datei bereit, die Sie herunterladen und verwenden können, um AWS IoT Greengrass Core-Software in einem Docker-Container auszuführen. Dockerfiles enthalten Quellcode zum Erstellen von AWS IoT Greengrass Container-Images.

Bevor Sie ein -AWS IoT GreengrassContainer-Image erstellen, müssen Sie Ihre Dockerfile so konfigurieren, dass sie die Version der -AWS IoT GreengrassCore-Software auswählt, die Sie installieren möchten. Sie können auch Umgebungsvariablen konfigurieren, um auszuwählen, wie Ressourcen während der Installation bereitgestellt und andere Installationsoptionen angepasst werden sollen. In diesem Abschnitt wird beschrieben, wie Sie ein AWS IoT Greengrass Docker-Image aus einer Dockerfile konfigurieren und erstellen.

Herunterladen des Dockerfile-Pakets

Sie können das AWS IoT Greengrass Dockerfile-Paket von herunterladen GitHub:

AWS Greengrass Docker-Repository

Nachdem Sie das Paket heruntergeladen haben, extrahieren Sie den Inhalt in den *download-directory/*`aws-greengrass-docker-`*nucleus-version* Ordner auf Ihrem Computer. Das Dockerfile verwendet eine ältere Version von Greengrass. Sie sollten die Datei aktualisieren, um die gewünschte Version von Greengrass zu verwenden.

Angeben der AWS IoT Greengrass Core-Softwareversion

Verwenden Sie das folgende Build-Argument im Dockerfile, um die Version der AWS IoT Greengrass Core-Software anzugeben, die Sie im AWS IoT Greengrass Docker-Image verwenden möchten. Standardmäßig verwendet das Dockerfile die neueste Version der AWS IoT Greengrass Core-Software.

`GREENGRASS_RELEASE_VERSION`

Die Version der -AWS IoT GreengrassCore-Software. Standardmäßig lädt die Dockerfile die neueste verfügbare Version des Greengrass-Kerns herunter. Legen Sie den Wert auf die Version des Kerns fest, die Sie herunterladen möchten.

Festlegen von Umgebungsvariablen

Mit Umgebungsvariablen können Sie anpassen, wie AWS IoT Greengrass Core-Software im Docker-Container installiert wird. Sie können Umgebungsvariablen für Ihr AWS IoT Greengrass Docker-Image auf verschiedene Arten festlegen.

- Um dieselben Umgebungsvariablen zum Erstellen mehrerer Images zu verwenden, legen Sie Umgebungsvariablen direkt im Dockerfile fest.
- Wenn Sie verwenden, `docker run` um Ihren Container zu starten, übergeben Sie Umgebungsvariablen als Argumente im Befehl oder legen Sie Umgebungsvariablen in einer Umgebungsvariablendatei fest und übergeben Sie die Datei dann als Argument. Weitere Informationen zum Festlegen von Umgebungsvariablen in Docker finden Sie in den Umgebungsvariablen in der Docker-Dokumentation.
- Wenn Sie `docker-compose up` zum Starten Ihres Containers verwenden, legen Sie Umgebungsvariablen in einer Umgebungsvariablendatei fest und übergeben Sie die Datei dann als Argument. Weitere Informationen zum Festlegen von Umgebungsvariablen in Compose finden Sie in der Docker-Dokumentation.

Sie können die folgenden Umgebungsvariablen für das AWS IoT Greengrass Docker-Image konfigurieren.

> **Note**
>
> Ändern Sie die `TINI_KILL_PROCESS_GROUP` Variable im Dockerfile nicht. Diese Variable ermöglicht die Weiterleitung `SIGTERM` an alle PIDs in der PID-Gruppe, sodass die AWS IoT Greengrass Core-Software ordnungsgemäß heruntergefahren werden kann, wenn der Docker-Container gestoppt wird.

`GGC_ROOT_PATH`

(Optional) Der Pfad zum Ordner innerhalb des Containers, der als Root für die AWS IoT Greengrass Core-Software verwendet werden soll.

Standard: `/greengrass/v2`

`PROVISION`

(Optional) Bestimmt, ob der AWS IoT Greengrass Core AWS Ressourcen bereitstellt.

- Wenn Sie angeben`true`, registriert die -AWS IoT GreengrassCore-Software das Container-Image als -AWS IoTObjekt und stellt die AWS Ressourcen bereit, die das Greengrass-Core-Gerät benötigt. Die AWS IoT Greengrass -Core-Software stellt ein -AWS IoTObjekt, (optional) eine -AWS IoTObjektgruppe, eine IAM-Rolle und einen -AWS IoTRollenalias bereit. Weitere Informationen finden Sie unter Ausführen AWS IoT Greengrass in einem Docker-Container mit automatischer Ressourcenbereitstellung.
- Wenn Sie angeben`false`, müssen Sie eine Konfigurationsdatei erstellen, die dem AWS IoT Greengrass Core-Installationsprogramm zur Verfügung gestellt wird und angibt, dass die AWS Ressourcen und Zertifikate verwendet werden sollen, die Sie manuell erstellt haben. Weitere Informationen finden Sie unter Ausführen AWS IoT Greengrass in einem Docker-Container mit manueller Ressourcenbereitstellung.

Standard: `false`

`AWS_REGION`

(Optional) Die AWS-Region, die die AWS IoT Greengrass -Core-Software zum Abrufen oder Erstellen der erforderlichen AWS Ressourcen verwendet.

Standard: `us-east-1`.

`THING_NAME`

(Optional) Der Name des AWS IoT Objekts, das Sie als dieses Core-Gerät registrieren. Wenn das Objekt mit diesem Namen in Ihrem nicht vorhanden istAWS-Konto, erstellt die AWS IoT Greengrass Core-Software es.

Sie müssen angeben`PROVISION=true`, um dieses Argument anzuwenden.

Standard: `GreengrassV2IotThing_` plus eine zufällige UUID.

`THING_GROUP_NAME`

(Optional) Der Name der AWS IoT Objektgruppe, zu der Sie die dieses Core-Geräts hinzufügen AWS IoT Wenn eine Bereitstellung auf diese Objektgruppe abzielt, erhalten diese und andere Core-Geräte in dieser Gruppe diese Bereitstellung, wenn sie eine Verbindung zu herstellenAWS IoT Greengrass. Wenn die Objektgruppe mit diesem Namen in Ihrem nicht vorhanden istAWS-Konto, erstellt die AWS IoT Greengrass Core-Software sie.

Sie müssen angeben`PROVISION=true`, um dieses Argument anzuwenden.

`TES_ROLE_NAME`

(Optional) Der Name der IAM-Rolle, die zum Abrufen von AWS Anmeldeinformationen verwendet werden soll, mit denen das Greengrass-Kerngerät mit -AWSServices interagieren kann. Wenn die Rolle mit diesem Namen in Ihrem nicht vorhanden istAWS-Konto, erstellt die AWS IoT Greengrass Core-Software sie mit der `GreengrassV2TokenExchangeRoleAccess` Richtlinie . Diese Rolle hat keinen Zugriff auf Ihre S3-Buckets, in denen Sie Komponentenartefakte hosten. Daher müssen Sie den S3-Buckets und -Objekten Ihrer Artefakte Berechtigungen hinzufügen, wenn Sie eine Komponente erstellen. Weitere Informationen finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

Standard: `GreengrassV2TokenExchangeRole`

`TES_ROLE_ALIAS_NAME`

(Optional) Der Name des AWS IoT Rollenalias, der auf die IAM-Rolle verweist, die AWS Anmeldeinformationen für das Greengrass-Kerngerät bereitstellt. Wenn der Rollenalias mit diesem Namen in Ihrem nicht vorhanden istAWS-Konto, erstellt die -AWS IoT GreengrassCore-Software ihn und verweist ihn auf die von Ihnen angegebene IAM-Rolle.

Standard: `GreengrassV2TokenExchangeRoleAlias`

`COMPONENT_DEFAULT_USER`

(Optional) Der Name oder die ID des Systembenutzers und der Systemgruppe, die die AWS IoT Greengrass Core-Software zum Ausführen von Komponenten verwendet. Geben Sie den Benutzer und die Gruppe durch einen Doppelpunkt getrennt an. Die Gruppe ist optional. Sie können beispielsweise **`ggc_user:ggc_group`** oder **`ggc_user`** angeben.

- Wenn Sie als Stamm ausführen, wird standardmäßig der Benutzer und die Gruppe verwendet, die in der Konfigurationsdatei definiert sind. Wenn die Konfigurationsdatei keinen Benutzer und keine Gruppe definiert, ist dies standardmäßig `ggc_user:ggc_group`. Wenn `ggc_user` oder `ggc_group` nicht vorhanden sind, erstellt die Software sie.
- Wenn Sie als Nicht-Root-Benutzer ausführen, verwendet die AWS IoT Greengrass -Core-Software diesen Benutzer, um Komponenten auszuführen.
- Wenn Sie keine Gruppe angeben, verwendet die AWS IoT Greengrass Core-Software die primäre Gruppe des Systembenutzers.

Weitere Informationen finden Sie unter Konfigurieren Sie den Benutzer, der die Komponenten ausführt.

`DEPLOY_DEV_TOOLS`

Definiert, ob die Greengrass-CLI-Komponente im Container-Image heruntergeladen und bereitgestellt werden soll. Sie können die Greengrass-CLI verwenden, um Komponenten lokal zu entwickeln und zu debuggen.

> **Important**
>
> Wir empfehlen, diese Komponente nur in Entwicklungsumgebungen und nicht in Produktionsumgebungen zu verwenden. Diese Komponente bietet Zugriff auf Informationen und Abläufe, die Sie normalerweise in einer Produktionsumgebung nicht benötigen. Folgen Sie dem Prinzip der geringsten Berechtigung, indem Sie diese Komponente nur auf Core-Geräten bereitstellen, auf denen Sie sie benötigen.

Standard: `false`

`INIT_CONFIG`

(Optional) Der Pfad zur Konfigurationsdatei, die zum Installieren der AWS IoT Greengrass Core-Software verwendet werden soll. Mit dieser Option können Sie neue Greengrass-Core-Geräte mit einer bestimmten Kernkonfiguration einrichten oder manuell bereitgestellte Ressourcen angeben. Sie müssen Ihre Konfigurationsdatei in dem Pfad mounten, den Sie in diesem Argument angeben.

`TRUSTED_PLUGIN`

Diese Funktion ist für v2.4.0 und höher der Greengrass-Kernkomponente verfügbar.

(Optional) Der Pfad zu einer JAR-Datei, die als vertrauenswürdiges Plugin geladen werden soll. Verwenden Sie diese Option, um Bereitstellungs-Plugin-JAR-Dateien bereitzustellen, z. B. um mit Flottenbereitstellung oder benutzerdefinierter Bereitstellung zu installieren.

`THING_POLICY_NAME`

Diese Funktion ist für v2.4.0 und höher der Greengrass-Kernkomponente verfügbar.

(Optional) Der Name der AWS IoT Richtlinie, die an das AWS IoT Objektzertifikat dieses Core-Geräts angehängt werden soll. Wenn die AWS IoT Richtlinie mit diesem Namen in Ihrer AWS IoT Greengrass Core-Software nicht vorhanden ist, erstellt AWS-Konto sie.

Sie müssen angeben`PROVISION=true`, um dieses Argument anzuwenden.

> **Note**
>
> Die AWS IoT Greengrass -Core-Software erstellt standardmäßig eine zulässige AWS IoT Richtlinie. Sie können diese Richtlinie einschränken oder eine benutzerdefinierte Richtlinie erstellen, in der Sie die Berechtigungen für Ihren Anwendungsfall einschränken. Weitere Informationen finden Sie unter Minimale AWS IoT Richtlinie für -AWS IoT Greengrass V2Core-Geräte.

Angeben der zu installierenden Abhängigkeiten

Die RUN-Anweisung im AWS IoT Greengrass Dockerfile bereitet die Containerumgebung vor, um das AWS IoT Greengrass Core-Softwareinstallationsprogramm auszuführen. Sie können die Abhängigkeiten anpassen, die installiert werden, bevor das AWS IoT Greengrass Core-Softwareinstallationsprogramm im Docker-Container ausgeführt wird.

Erstellen des AWS IoT Greengrass Images

Verwenden Sie das AWS IoT Greengrass Dockerfile, um ein -AWS IoT GreengrassContainer-Image zu erstellen. Sie können die Docker CLI oder die Docker Compose CLI verwenden, um das Image zu erstellen und den Container zu starten. Sie können auch die Docker-CLI verwenden, um das Image zu erstellen, und dann Docker Compose verwenden, um Ihren Container von diesem Image aus zu starten.

Docker

1. Führen Sie auf dem Host-Computer den folgenden Befehl aus, um zu dem Verzeichnis zu wechseln, das die konfigurierte Dockerfile enthält.

   ```
   cd download-directory/aws-greengrass-docker-nucleus-version
   ```

2. Führen Sie den folgenden Befehl aus, um das AWS IoT Greengrass Container-Image aus der Dockerfile zu erstellen.

   ```
   sudo docker build -t "platform/aws-iot-greengrass:nucleus-version" ./
   ```

Docker Compose

1. Führen Sie auf dem Hostcomputer den folgenden Befehl aus, um zu dem Verzeichnis zu wechseln, das die Dockerfile- und die Compose-Datei enthält.

   ```
   cd download-directory/aws-greengrass-docker-nucleus-version
   ```

2. Führen Sie den folgenden Befehl aus, um die Compose-Datei zum Erstellen des AWS IoT Greengrass Container-Images zu verwenden.

   ```
   docker-compose -f docker-compose.yml build
   ```

Sie haben das AWS IoT Greengrass Container-Image erfolgreich erstellt. Auf dem Docker-Image ist die -AWS IoT GreengrassCore-Software installiert. Sie können die -AWS IoT GreengrassCore-Software jetzt in einem Docker-Container ausführen.

Ausführen AWS IoT Greengrass in einem Docker-Container mit automatischer Ressourcenbereitstellung

In diesem Tutorial erfahren Sie, wie Sie AWS IoT Greengrass Core-Software in einem Docker-Container mit automatisch bereitgestellten AWS Ressourcen und lokalen Entwicklungstools installieren und ausführen. Sie können diese Entwicklungsumgebung verwenden, um AWS IoT Greengrass Funktionen in einem Docker-Container zu erkunden. Die Software benötigt AWS Anmeldeinformationen, um diese Ressourcen bereitzustellen und die lokalen Entwicklungstools bereitzustellen.

Wenn Sie dem Container keine AWS Anmeldeinformationen bereitstellen können, können Sie die AWS Ressourcen bereitstellen, die das Core-Gerät für den Betrieb benötigt. Sie können die Entwicklungstools auch auf einem Core-Gerät bereitstellen, das als Entwicklungsgerät verwendet werden soll. Auf diese Weise können Sie dem Gerät weniger Berechtigungen erteilen, wenn Sie den Container ausführen. Weitere Informationen finden Sie unter Ausführen AWS IoT Greengrass in einem Docker-Container mit manueller Ressourcenbereitstellung.

Voraussetzungen

Um dieses Tutorial abzuschließen, benötigen Sie Folgendes.

- Ein(e) AWS-Konto. Falls Sie noch keines haben, beachten Sie die Informationen unter Richten Sie eine ein AWS-Konto.
- Ein AWS IAM-Benutzer mit Berechtigungen zum Bereitstellen der AWS IoT und IAM-Ressourcen für ein Greengrass-Core-Gerät. Das AWS IoT Greengrass-Core-Softwareinstallationsprogramm verwendet Ihre -AWSAnmeldeinformationen, um diese Ressourcen automatisch bereitzustellen. Informationen zur minimalen IAM-Richtlinie zur automatischen Bereitstellung von Ressourcen finden Sie unter Minimale IAM-Richtlinie für das Installationsprogramm zur Bereitstellung von Ressourcen.
- Ein AWS IoT Greengrass Docker-Image. Sie können ein Image aus der AWS IoT Greengrass Dockerfile erstellen.
- Der Host-Computer, auf dem Sie den Docker-Container ausführen, muss die folgenden Anforderungen erfüllen:
  - Ein Linux-basiertes Betriebssystem mit einer Internetverbindung.
  - Docker-Engine-Version 18.09 oder höher.
  - (Optional) Docker Compose Version 1.22 oder höher. Docker Compose ist nur erforderlich, wenn Sie die Docker Compose CLI zum Ausführen Ihrer Docker-Images verwenden möchten.

Konfigurieren Ihrer AWS-Anmeldeinformationen

In diesem Schritt erstellen Sie eine Datei mit Anmeldeinformationen auf dem Host-Computer, die Ihre AWS Sicherheitsanmeldeinformationen enthält. Wenn Sie das AWS IoT Greengrass Docker-Image ausführen, müssen Sie den Ordner, der diese Datei mit den Anmeldeinformationen enthält, `/root/.aws/` in im Docker-Container mounten. Das AWS IoT Greengrass Installationsprogramm verwendet diese Anmeldeinformationen, um Ressourcen in Ihrem bereitzustellenAWS-Konto. Informationen zur minimalen IAM-Richtlinie, die das Installationsprogramm zum automatischen Bereitstellen von Ressourcen benötigt, finden Sie unter Minimale IAM-Richtlinie für das Installationsprogramm zur Bereitstellung von Ressourcen.

1. Rufen Sie eine der folgenden Optionen ab.

   - Langfristige Anmeldeinformationen für einen IAM-Benutzer. Informationen zum Abrufen langfristiger Anmeldeinformationen finden Sie unter Verwalten von Zugriffsschlüsseln für IAM-Benutzer im IAM-Benutzerhandbuch.
   - (Empfohlen) Temporäre Anmeldeinformationen für eine IAM-Rolle. Informationen zum Abrufen temporärer Anmeldeinformationen finden Sie unter Verwenden temporärer Sicherheitsanmeldeinformationen mit AWS CLI dem im IAM-Benutzerhandbuch.

2. Erstellen Sie einen Ordner, in dem Sie Ihre Anmeldeinformationsdatei ablegen.

```
mkdir ./greengrass-v2-credentials
```

3. Verwenden Sie einen Texteditor, um eine Konfigurationsdatei mit dem Namen `credentials` im `./greengrass-v2-credentials` Ordner zu erstellen.

   Sie können beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen der `credentials` Datei zu verwenden.

```
nano ./greengrass-v2-credentials/credentials
```

4. Fügen Sie der `credentials` Datei Ihre AWS Anmeldeinformationen im folgenden Format hinzu.

```
[default]
aws_access_key_id = AKIAIOSFODNN7EXAMPLE
aws_secret_access_key = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
aws_session_token
 = AQoEXAMPLEH4aoAH0gNCAPy...truncated...zrkuWJOgQs8IZZaIv2BXIa2R4Olgk
```

   Nur `aws_session_token` für temporäre Anmeldeinformationen einschließen.

> **Important**
>
> Entfernen Sie die Datei mit den Anmeldeinformationen vom Hostcomputer, nachdem Sie den AWS IoT Greengrass Container gestartet haben. Wenn Sie die Datei mit den Anmeldeinformationen nicht entfernen, bleiben Ihre AWS Anmeldeinformationen im Container gemountet. Weitere Informationen finden Sie unter Ausführen der AWS IoT Greengrass Core-Software in einem Container.

Erstellen einer Umgebungsdatei

In diesem Tutorial wird eine Umgebungsdatei verwendet, um die Umgebungsvariablen festzulegen, die an das AWS IoT Greengrass -Core-Softwareinstallationsprogramm im Docker-Container übergeben werden. Sie können auch das Argument `-e` oder in Ihrem Befehl verwenden, um Umgebungsvariablen im Docker-Container festzulegen, oder Sie können die Variablen in einem `--env` -Block in der -`docker-compose.yml`Datei festlegen. `docker run` `environment`

1. Verwenden Sie einen Texteditor, um eine Umgebungsdatei mit dem Namen zu erstellen `.env`.

   Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen des `.env` im aktuellen Verzeichnis zu verwenden.

   ```
   nano .env
   ```

2. Kopieren Sie den folgenden Inhalt in die Datei .

   ```
   GGC_ROOT_PATH=/greengrass/v2
   AWS_REGION=region
   PROVISION=true
   THING_NAME=MyGreengrassCore
   THING_GROUP_NAME=MyGreengrassCoreGroup
   TES_ROLE_NAME=GreengrassV2TokenExchangeRole
   TES_ROLE_ALIAS_NAME=GreengrassCoreTokenExchangeRoleAlias
   COMPONENT_DEFAULT_USER=ggc_user:ggc_group
   ```

   Ersetzen Sie dann die folgenden Werte.

   - */greengrass/v2*. Der Greengrass-Stammordner, den Sie für die Installation verwenden möchten. Sie verwenden die `GGC_ROOT` Umgebungsvariable , um diesen Wert festzulegen.
   - *Region*. Die AWS-Region, in der Sie die Ressourcen erstellt haben.
   - *MyGreengrassCore*. Der Name des AWS IoT-Objekts. Wenn das Objekt nicht vorhanden ist, erstellt das Installationsprogramm es. Das Installationsprogramm lädt die Zertifikate herunter, um sich als das AWS IoT Objekt zu authentifizieren.
   - *MyGreengrassCoreGroup*. Der Name der AWS IoT Objektgruppe. Wenn die Objektgruppe nicht vorhanden ist, erstellt das Installationsprogramm sie und fügt ihr das Objekt hinzu. Wenn die Objektgruppe vorhanden ist und über eine aktive Bereitstellung verfügt, lädt das Core-Gerät die von der Bereitstellung angegebene Software herunter und führt sie aus.
   - *GreengrassV2TokenExchangeRole*. Ersetzen Sie durch den Namen der IAM-Token-Austauschrolle, die es dem Greengrass-Core-Gerät ermöglicht, temporäre AWS Anmeldeinformationen zu erhalten. Wenn die Rolle nicht vorhanden ist, erstellt das Installationsprogramm sie und erstellt und fügt eine Richtlinie mit dem Namen *GreengrassV2TokenExchangeRole*Access an. Weitere Informationen finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

- *GreengrassCoreTokenExchangeRoleAlias*. Der Token-Exchange-Rollenalias. Wenn der Rollenalias nicht vorhanden ist, erstellt das Installationsprogramm ihn und verweist ihn auf die von Ihnen angegebene IAM-Token-Austauschrolle. Weitere Informationen finden Sie unter

> **Note**
>
> Sie können die `DEPLOY_DEV_TOOLS` Umgebungsvariable auf setzen, `true` um die Greengrass-CLI-Komponente bereitzustellen, mit der Sie benutzerdefinierte Komponenten innerhalb des Docker-Containers entwickeln können. Wir empfehlen, diese Komponente nur in Entwicklungsumgebungen und nicht in Produktionsumgebungen zu verwenden. Diese Komponente bietet Zugriff auf Informationen und Vorgänge, die Sie normalerweise in einer Produktionsumgebung nicht benötigen. Folgen Sie dem Prinzip der geringsten Berechtigung, indem Sie diese Komponente nur auf Core-Geräten bereitstellen, auf denen Sie sie benötigen.

Ausführen der AWS IoT Greengrass Core-Software in einem Container

In diesem Tutorial erfahren Sie, wie Sie das Docker-Image starten, das Sie in einen Docker-Container integriert haben. Sie können die Docker CLI oder die Docker Compose CLI verwenden, um das AWS IoT Greengrass Core-Software-Image in einem Docker-Container auszuführen.

Docker

1. Führen Sie den folgenden Befehl aus, um den Docker-Container zu starten.

```
docker run --rm --init -it --name docker-image \
 -v path/to/greengrass-v2-credentials:/root/.aws/:ro \
 --env-file .env \
 -p 8883 \
 your-container-image:version
```

   Dieser Beispielbefehl verwendet die folgenden Argumente für die Docker-Ausführung :

   - `--rm`. Bereinigt den Container, wenn er beendet wird.
   - `--init`. Verwendet einen Init-Prozess im Container.

> **Note**
>
> Das `--init` Argument ist erforderlich, um die AWS IoT Greengrass Core-Software herunterzufahren, wenn Sie den Docker-Container stoppen.

- `-it`. (Optional) Führt den Docker-Container im Vordergrund als interaktiven Prozess aus. Sie können dies durch das `-d` Argument ersetzen, um den Docker-Container stattdessen im getrennten Modus auszuführen. Weitere Informationen finden Sie unter Trennen im Vergleich zum Vordergrund in der Docker-Dokumentation.
- `--name`. Führt einen Container mit dem Namen aus `aws-iot-greengrass`
- `-v`. Mountet ein Volume im Docker-Container, um die Konfigurationsdatei und die Zertifikatsdateien für die AWS IoT Greengrass Ausführung im Container verfügbar zu machen.
- `--env-file`. (Optional) Gibt die Umgebungsdatei an, um die Umgebungsvariablen festzulegen, die an das AWS IoT Greengrass Core-Softwareinstallationsprogramm im Docker-Container übergeben werden. Dieses Argument ist nur erforderlich, wenn Sie eine Umgebungsdatei zum Festlegen von Umgebungsvariablen erstellt haben. Wenn Sie keine Umgebungsdatei erstellt haben, können Sie `--env` Argumente verwenden, um Umgebungsvariablen direkt in Ihrem Docker-Ausführungsbefehl festzulegen.
- `-p`. (Optional) Veröffentlicht den 8883-Container-Port auf dem Host-Computer. Dieses Argument ist erforderlich, wenn Sie eine Verbindung herstellen und über MQTT kommunizieren möchten, da Port 8883 für MQTT-Datenverkehr AWS IoT Greengrass verwendet. Um andere Ports zu öffnen, verwenden Sie zusätzliche `-p` Argumente.

> **Note**
>
> Um Ihren Docker-Container mit erhöhter Sicherheit auszuführen, können Sie die Argumente `--cap-drop` und verwenden, um die Linux`--cap-add`-Funktionen für Ihren Container selektiv zu aktivieren. Weitere Informationen finden Sie unter Laufzeitberechtigungen und Linux-Funktionen in der Docker-Dokumentation.

2. Entfernen Sie die Anmeldeinformationen von `./greengrass-v2-credentials` auf dem Host-Gerät.

```
rm -rf ./greengrass-v2-credentials
```

> **Important**
>
> Sie entfernen diese Anmeldeinformationen, da sie umfassende Berechtigungen bieten, die das Core-Gerät nur während der Einrichtung benötigt. Wenn Sie diese Anmeldeinformationen nicht entfernen, können Greengrass-Komponenten und andere Prozesse, die im Container ausgeführt werden, darauf zugreifen. Wenn Sie AWS Anmeldeinformationen für eine Greengrass-Komponente bereitstellen müssen, verwenden Sie den Token-Exchange-Service. Weitere Informationen finden Sie unter Interagieren mit -AWSServices.

Docker Compose

1. Verwenden Sie einen Texteditor, um eine Docker-Compose-Datei mit dem Namen zu erstellen`docker-compose.yml`.

   Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen des `docker-compose.yml` im aktuellen Verzeichnis zu verwenden.

   ```
   nano docker-compose.yml
   ```

   > **Note**
   >
   > Sie können auch die neueste Version der von bereitgestellten ComposeAWS-Datei von herunterladen und verwendenGitHub.

2. Fügen Sie der Compose-Datei den folgenden Inhalt hinzu. Die Datei sollte ähnlich wie im folgenden Beispiel aussehen. Ersetzen Sie *`docker-image`* durch den Namen Ihres Docker-Images.

   ```
   version: '3.7'

   services:
     greengrass:
       init: true
       container_name: aws-iot-greengrass
       image: docker-image
   ```

```
    volumes:
      - ./greengrass-v2-credentials:/root/.aws/:ro
    env_file: .env
    ports:
      - "8883:8883"
```

Die folgenden Parameter in diesem Beispiel für die Compose-Datei sind optional:

- `ports`– Veröffentlicht die 8883-Container-Ports auf dem Host-Computer. Dieser Parameter ist erforderlich, wenn Sie eine Verbindung herstellen und über MQTT kommunizieren möchten, da Port 8883 für MQTT-Datenverkehr AWS IoT Greengrass verwendet.
- `env_file`– Gibt die Umgebungsdatei an, um die Umgebungsvariablen festzulegen, die an das AWS IoT Greengrass Core-Softwareinstallationsprogramm im Docker-Container übergeben werden. Dieser Parameter ist nur erforderlich, wenn Sie eine Umgebungsdatei zum Festlegen von Umgebungsvariablen erstellt haben. Wenn Sie keine Umgebungsdatei erstellt haben, können Sie den Umgebungsparameter verwenden, um die Variablen direkt in Ihrer Compose-Datei festzulegen.

> **Note**
>
> Um Ihren Docker-Container mit erhöhter Sicherheit auszuführen, können Sie `cap_drop` und `cap_add` in Ihrer Compose-Datei verwenden, um die Linux-Funktionen für Ihren Container selektiv zu aktivieren. Weitere Informationen finden Sie unter Laufzeitberechtigungen und Linux-Funktionen in der Docker-Dokumentation.

3. Führen Sie den folgenden Befehl aus, um den Docker-Container zu starten.

```
docker-compose -f docker-compose.yml up
```

4. Entfernen Sie die Anmeldeinformationen von `./greengrass-v2-credentials` auf dem Host-Gerät.

```
rm -rf ./greengrass-v2-credentials
```

> **Important**
>
> Sie entfernen diese Anmeldeinformationen, da sie umfassende Berechtigungen bieten, die das Core-Gerät nur während der Einrichtung benötigt. Wenn Sie diese Anmeldeinformationen nicht entfernen, können Greengrass-Komponenten und andere Prozesse, die im Container ausgeführt werden, darauf zugreifen. Wenn Sie AWS Anmeldeinformationen für eine Greengrass-Komponente bereitstellen müssen, verwenden Sie den Token-Exchange-Service. Weitere Informationen finden Sie unter Interagieren mit -AWSServices.

Nächste Schritte

AWS IoT Greengrass Die -Core-Software wird jetzt in einem Docker-Container ausgeführt. Führen Sie den folgenden Befehl aus, um die Container-ID für den aktuell ausgeführten Container abzurufen.

```
docker ps
```

Anschließend können Sie den folgenden Befehl ausführen, um auf den Container zuzugreifen und AWS IoT Greengrass Core-Software zu erkunden, die im Container ausgeführt wird.

```
docker exec -it container-id /bin/bash
```

Informationen zum Erstellen einer einfachen Komponente finden Sie unter Schritt 4: Entwickeln und Testen einer Komponente auf Ihrem Gerät in Tutorial: Erste Schritte mit AWS IoT Greengrass V2

> **Note**
>
> Wenn Sie verwenden, `docker exec` um Befehle im Docker-Container auszuführen, werden diese Befehle nicht in den Docker-Protokollen protokolliert. Um Ihre Befehle in den Docker-Protokollen zu protokollieren, fügen Sie eine interaktive Shell an den Docker-Container an. Weitere Informationen finden Sie unter Anfügen einer interaktiven Shell an den Docker-Container.

Die AWS IoT Greengrass Core-Protokolldatei heißt `greengrass.log` und befindet sich in */greengrass/v2*`/logs`. Komponentenprotokolldateien befinden sich ebenfalls im selben

Verzeichnis. Um Greengrass-Protokolle in ein temporäres Verzeichnis auf dem Host zu kopieren, führen Sie den folgenden Befehl aus:

```
docker cp container-id:/greengrass/v2/logs /tmp/logs
```

Wenn Sie Protokolle nach dem Beenden oder Entfernen eines Containers beibehalten möchten, empfehlen wir, nur das */greengrass/v2*/logs Verzeichnis an das temporäre Protokollverzeichnis auf dem Host zu binden, anstatt das gesamte Greengrass-Verzeichnis zu mounten. Weitere Informationen finden Sie unter Greengrass-Protokolle außerhalb des Docker-Containers beibehalten.

Um einen laufenden AWS IoT Greengrass Docker-Container zu stoppen, führen Sie `docker stop` oder aus`docker-compose -f docker-compose.yml stop`. Diese Aktion sendet `SIGTERM` an den Greengrass-Prozess und fährt alle zugehörigen Prozesse herunter, die im Container gestartet wurden. Der Docker-Container wird mit der `docker-init` ausführbaren Datei als Prozess-PID 1 initialisiert, was beim Entfernen aller verbleibenden Zombie-Prozesse hilft. Weitere Informationen finden Sie unter Specify an init process in der Docker-Dokumentation.

Informationen zur Behebung von Problemen mit der Ausführung AWS IoT Greengrass in einem Docker-Container finden Sie unter Fehlerbehebung bei AWS IoT Greengrass in einem Docker-Container.

Ausführen AWS IoT Greengrass in einem Docker-Container mit manueller Ressourcenbereitstellung

In diesem Tutorial erfahren Sie, wie Sie AWS IoT Greengrass Core-Software im Docker-Container mit manuell bereitgestellten AWS Ressourcen installieren und ausführen.

Themen

- Voraussetzungen
- Abrufen von AWS IoT Endpunkten
- Erstellen eines -AWS IoTObjekts
- Erstellen des Objektzertifikats
- Konfigurieren des Objektzertifikats
- Erstellen einer Token-Exchange-Rolle
- Herunterladen von Zertifikaten auf das Gerät
- Erstellen einer Konfigurationsdatei

Voraussetzungen

Zum Durcharbeiten dieses Tutorials ist Folgendes erforderlich:

- Ein(e) AWS-Konto. Falls Sie noch keines haben, beachten Sie die Informationen unter Richten Sie eine ein AWS-Konto.
- Ein AWS IoT Greengrass Docker-Image. Sie können ein Image aus dem AWS IoT Greengrass Dockerfile erstellen.
- Der Hostcomputer, auf dem Sie den Docker-Container ausführen, muss die folgenden Anforderungen erfüllen:
  - Ein Linux-basiertes Betriebssystem mit einer Internetverbindung.
  - Docker-Engine-Version 18.09 oder höher.
  - (Optional) Docker Compose Version 1.22 oder höher. Docker Compose ist nur erforderlich, wenn Sie die Docker Compose CLI zum Ausführen Ihrer Docker-Images verwenden möchten.

Abrufen von AWS IoT Endpunkten

Rufen Sie die AWS IoT Endpunkte für Ihr ab AWS-Kontound speichern Sie sie zur späteren Verwendung. Ihr Gerät verwendet diese Endpunkte, um eine Verbindung zu herzustellenAWS IoT. Gehen Sie wie folgt vor:

1. Rufen Sie den AWS IoT Datenendpunkt für Ihr abAWS-Konto.

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

   Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anforderung erfolgreich ist.

   ```
   {
     "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
   }
   ```

2. Rufen Sie den AWS IoT Anmeldeinformationsendpunkt für Ihr abAWS-Konto.

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

Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anforderung erfolgreich ist.

```
{
  "endpointAddress": "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
}
```

Erstellen eines -AWS IoTObjekts

AWS IoT -Objekte stellen Geräte und logische Entitäten dar, die eine Verbindung zu herstellenAWS IoT. Greengrass-Core-Geräte sind AWS IoT Dinge. Wenn Sie ein Gerät als -AWS IoTObjekt registrieren, kann dieses Gerät ein digitales Zertifikat verwenden, um sich bei zu authentifizierenAWS.

In diesem Abschnitt erstellen Sie ein -AWS IoTObjekt, das Ihr Gerät repräsentiert.

So erstellen Sie ein -AWS IoTObjekt

1. Erstellen Sie ein -AWS IoTObjekt für Ihr Gerät. Führen Sie auf Ihrem Entwicklungscomputer den folgenden Befehl aus.

   - Ersetzen Sie *MyGreengrassCore* durch den zu verwendenden Objektnamen. Dieser Name ist auch der Name Ihres Greengrass-Kerngeräts.

     > **Note**
     >
     > Der Objektname darf keine Doppelpunktzeichen (`:`) enthalten.

   ```
   aws iot create-thing --thing-name MyGreengrassCore
   ```

   Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anforderung erfolgreich ist.

   ```
   {
     "thingName": "MyGreengrassCore",
     "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
   ```

```
  "thingId": "8cb4b6cd-268e-495d-b5b9-1713d71dbf42"
}
```

2. (Optional) Fügen Sie das AWS IoT Objekt einer neuen oder vorhandenen Objektgruppe hinzu. Sie verwenden Objektgruppen, um Flotten von Greengrass-Core-Geräten zu verwalten. Wenn Sie Softwarekomponenten auf Ihren Geräten bereitstellen, können Sie einzelne Geräte oder Gerätegruppen anvisieren. Sie können ein Gerät zu einer Objektgruppe mit einer aktiven Greengrass-Bereitstellung hinzufügen, um die Softwarekomponenten dieser Objektgruppe auf dem Gerät bereitzustellen. Gehen Sie wie folgt vor:

   a. (Optional) Erstellen Sie eine -AWS IoTObjektgruppe.

      - Ersetzen Sie *MyGreengrassCoreGroup* durch den Namen der zu erstellenden Objektgruppe.

      > **Note**
      >
      > Der Objektgruppenname darf keine Doppelpunktzeichen (`:`) enthalten.

      ```
      aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
      ```

      Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anforderung erfolgreich ist.

      ```
      {
        "thingGroupName": "MyGreengrassCoreGroup",
        "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
      MyGreengrassCoreGroup",
        "thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"
      }
      ```

   b. Fügen Sie das AWS IoT Objekt einer Objektgruppe hinzu.

      - Ersetzen Sie *MyGreengrassCore* durch den Namen Ihres AWS IoT Objekts.
      - Ersetzen Sie *MyGreengrassCoreGroup* durch den Namen der Objektgruppe.

```
aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-
name MyGreengrassCoreGroup
```

Der Befehl hat keine Ausgabe, wenn die Anforderung erfolgreich ist.

Erstellen des Objektzertifikats

Wenn Sie ein Gerät als -AWS IoTObjekt registrieren, kann dieses Gerät ein digitales Zertifikat verwenden, um sich bei zu authentifizierenAWS. Dieses Zertifikat ermöglicht dem Gerät die Kommunikation mit AWS IoT und AWS IoT Greengrass.

In diesem Abschnitt erstellen und laden Sie Zertifikate herunter, mit denen Ihr Gerät eine Verbindung zu herstellen kannAWS.

So erstellen Sie das Objektzertifikat

1. Erstellen Sie einen Ordner, in dem Sie die Zertifikate für das AWS IoT Objekt herunterladen.

   ```
   mkdir greengrass-v2-certs
   ```

2. Erstellen Sie die Zertifikate für das AWS IoT Objekt und laden Sie sie herunter.

   ```
   aws iot create-keys-and-certificate --set-as-active --certificate-pem-outfile
    greengrass-v2-certs/device.pem.crt --public-key-outfile greengrass-v2-certs/
   public.pem.key --private-key-outfile greengrass-v2-certs/private.pem.key
   ```

   Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anforderung erfolgreich ist.

   ```
   {
     "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
   aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
     "certificateId":
    "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
     "certificatePem": "-----BEGIN CERTIFICATE-----
   MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w
    0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
    WF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIw
    EAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5
    jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
   ```

```
 MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBb
 WF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMx
 HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wgZ8wDQYJKoZIhvcNAQE
 BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
 k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
 ITxOUSQv7c7ugFFDzQGBzZswY6786m86gpEIbb3OhjZnzcvQAaRHhdlQWIMm2nr
 AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
 KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJIlJ00zbhNYS5f6Guo
 EDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
 3rrszlaEXAMPLE=
-----END CERTIFICATE-----",
  "keyPair": {
    "PublicKey": "-----BEGIN PUBLIC KEY-----\
MIIBIjANBgkqhkEXAMPLEQEFAAOCAQ8AMIIBCgKCAQEAEXAMPLE1nnyJwKSMHw4h\
MMEXAMPLEuuN/dMAS3fyce8DW/4+EXAMPLEyjmoF/YVF/gHr99VEEXAMPLE5VF13\
59VK7cEXAMPLE67GK+y+jikqXOgHh/xJTwo
+sGpWEXAMPLEDz18xOd2ka4tCzuWEXAMPLEahJbYkCPUBSU8opVkR7qkEXAMPLE1DR6sx2HocliOOLtu6Fkw91swQWE
\\GB3ZPrNh0PzQYvjUStZeccyNCx2EXAMPLEvp9mQOUXP6plfgxwKRX2fEXAMPLEDa\
hJLXkX3rHU2xbxJSq7D+XEXAMPLEcw+LyFhI5mgFRl88eGdsAEXAMPLElnI9EesG\
FQIDAQAB\
-----END PUBLIC KEY-----\
",
    "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\
key omitted for security reasons\
-----END RSA PRIVATE KEY-----\
"
  }
}
```

Speichern Sie den Amazon-Ressourcennamen (ARN) des Zertifikats, um das Zertifikat später zu konfigurieren.

Konfigurieren des Objektzertifikats

Fügen Sie das Objektzertifikat an das AWS IoT Objekt an, das Sie zuvor erstellt haben, und fügen Sie dem Zertifikat eine -AWS IoTRichtlinie hinzu, um die AWS IoT Berechtigungen für das Core-Gerät zu definieren.

So konfigurieren Sie das Objektzertifikat

1. Fügen Sie das Zertifikat an das AWS IoT Objekt an.

- Ersetzen Sie *MyGreengrassCore* durch den Namen Ihres AWS IoT Objekts.
- Ersetzen Sie den Amazon-Ressourcennamen (ARN) des Zertifikats durch den ARN des Zertifikats, das Sie im vorherigen Schritt erstellt haben.

```
aws iot attach-thing-principal --thing-name MyGreengrassCore
 --principal arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

Der Befehl hat keine Ausgabe, wenn die Anforderung erfolgreich ist.

2. Erstellen und fügen Sie eine -AWS IoTRichtlinie an, die die AWS IoT Berechtigungen für Ihr Greengrass-Kerngerät definiert. Die folgende Richtlinie ermöglicht den Zugriff auf alle MQTT-Themen und Greengrass-Operationen, sodass Ihr Gerät mit benutzerdefinierten Anwendungen und zukünftigen Änderungen funktioniert, die neue Greengrass-Operationen erfordern. Sie können diese Richtlinie je nach Anwendungsfall einschränken. Weitere Informationen finden Sie unter Minimale AWS IoT Richtlinie für -AWS IoT Greengrass V2Core-Geräte.

   Wenn Sie bereits ein Greengrass-Core-Gerät eingerichtet haben, können Sie dessen AWS IoT Richtlinie anfügen, anstatt ein neues zu erstellen.

   Gehen Sie wie folgt vor:

   a. Erstellen Sie eine -Datei, die das AWS IoT Richtliniendokument enthält, das Greengrass-Core-Geräte benötigen.

      Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen der Datei zu verwenden.

      ```
      nano greengrass-v2-iot-policy.json
      ```

      Kopieren Sie den folgenden JSON-Code in die Datei .

      ```
      {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Action": [
      ```

```
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive",
        "iot:Connect",
        "greengrass:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

b. Erstellen Sie eine -AWS IoTRichtlinie aus dem Richtliniendokument.

- Ersetzen Sie *GreengrassV2IoTThingPolicy* durch den Namen der zu erstellenden Richtlinie.

```
aws iot create-policy --policy-name GreengrassV2IoTThingPolicy --policy-
document file://greengrass-v2-iot-policy.json
```

Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anforderung erfolgreich ist.

```
{
  "policyName": "GreengrassV2IoTThingPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
  "policyDocument": "{
    \\"Version\\": \\"2012-10-17\\",
    \\"Statement\\": [
      {
        \\"Effect\\": \\"Allow\\",
        \\"Action\\": [
          \\"iot:Publish\\",
          \\"iot:Subscribe\\",
          \\"iot:Receive\\",
          \\"iot:Connect\\",
          \\"greengrass:*\\"
        ],
        \\"Resource\\": [
          \\"*\\"
```

```
        ]
      }
    ]
  }",
  "policyVersionId": "1"
}
```

c. Fügen Sie die AWS IoT Richtlinie an das Objektzertifikat AWS IoT an.

- Ersetzen Sie *GreengrassV2IoTThingPolicy* durch den Namen der anzuhängenden Richtlinie.
- Ersetzen Sie den Ziel-ARN durch den ARN des Zertifikats für Ihr AWS IoT Objekt.

```
aws iot attach-policy --policy-name GreengrassV2IoTThingPolicy
 --target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

Der Befehl hat keine Ausgabe, wenn die Anforderung erfolgreich ist.

Erstellen einer Token-Exchange-Rolle

Greengrass-Core-Geräte verwenden eine IAM-Servicerolle, die als Token-Exchange-Rolle bezeichnet wird, um Aufrufe von -AWSServices zu autorisieren. Das Gerät verwendet den AWS IoT Anmeldeinformationsanbieter, um temporäre AWS Anmeldeinformationen für diese Rolle abzurufen, die es dem Gerät ermöglichen, mit zu interagierenAWS IoT, Protokolle an Amazon CloudWatch Logs zu senden und benutzerdefinierte Komponentenartefakte von Amazon S3 herunterzuladen. Weitere Informationen finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

Sie verwenden einen -AWS IoTRollenalias, um die Token-Exchange-Rolle für Greengrass-Core-Geräte zu konfigurieren. Mit Rollenaliasnamen können Sie die Token-Exchange-Rolle für ein Gerät ändern, die Gerätekonfiguration jedoch beibehalten. Weitere Informationen finden Sie unter Autorisieren von direkten Aufrufen an -AWSServices im AWS IoT Core -Entwicklerhandbuch.

In diesem Abschnitt erstellen Sie eine IAM-Rolle für den Tokenaustausch und einen -AWS IoTRollenalias, der auf die Rolle verweist. Wenn Sie bereits ein Greengrass-Core-Gerät eingerichtet haben, können Sie seine Token-Austauschrolle und seinen Rollenalias verwenden, anstatt neue zu erstellen. Anschließend konfigurieren Sie das AWS IoT Objekt Ihres Geräts so, dass es diese Rolle und diesen Alias verwendet.

So erstellen Sie eine IAM-Rolle für den Tokenaustausch

1. Erstellen Sie eine IAM-Rolle, die Ihr Gerät als Token-Austauschrolle verwenden kann. Gehen Sie wie folgt vor:

   a. Erstellen Sie eine -Datei, die das Vertrauensrichtliniendokument enthält, das die Token-Exchange-Rolle benötigt.

      Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen der Datei zu verwenden.

      ```
      nano device-role-trust-policy.json
      ```

      Kopieren Sie den folgenden JSON-Code in die Datei .

      ```
      {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Principal": {
              "Service": "credentials.iot.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
          }
        ]
      }
      ```

   b. Erstellen Sie die Token-Exchange-Rolle mit dem Vertrauensrichtliniendokument.

      - Ersetzen Sie *GreengrassV2TokenExchangeRole* durch den Namen der zu erstellenden IAM-Rolle.

      ```
      aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-
      policy-document file://device-role-trust-policy.json
      ```

      Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anforderung erfolgreich ist.

      ```
      {
      ```

```
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassV2TokenExchangeRole",
    "RoleId": "AROAZ2YMUHYHK5OKM77FB",
    "Arn": "arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole",
    "CreateDate": "2021-02-06T00:13:29+00:00",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "credentials.iot.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
  }
```

c. Erstellen Sie eine -Datei, die das Zugriffsrichtliniendokument enthält, das die Token-Exchange-Rolle benötigt.

   Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen der Datei zu verwenden.

   ```
   nano device-role-access-policy.json
   ```

   Kopieren Sie den folgenden JSON-Code in die Datei .

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": [
           "logs:CreateLogGroup",
           "logs:CreateLogStream",
           "logs:PutLogEvents",
           "logs:DescribeLogStreams",
           "s3:GetBucketLocation"
         ],
   ```

```
      "Resource": "*"
    }
  ]
}
```

> **Note**
>
> Diese Zugriffsrichtlinie erlaubt keinen Zugriff auf Komponentenartefakte in S3-Buckets. Um benutzerdefinierte Komponenten bereitzustellen, die Artefakte in Amazon S3 definieren, müssen Sie der Rolle Berechtigungen hinzufügen, damit Ihr Core-Gerät Komponentenartefakte abrufen kann. Weitere Informationen finden Sie unter Erlauben Sie den Zugriff auf S3-Buckets für Komponentenartefakte.
> Wenn Sie noch keinen S3-Bucket für Komponentenartefakte haben, können Sie diese Berechtigungen später hinzufügen, nachdem Sie einen Bucket erstellt haben.

d. Erstellen Sie die IAM-Richtlinie aus dem Richtliniendokument.

- Ersetzen Sie *GreengrassV2TokenExchangeRoleAccess* durch den Namen der zu erstellenden IAM-Richtlinie.

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --
policy-document file://device-role-access-policy.json
```

Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anforderung erfolgreich ist.

```
{
  "Policy": {
    "PolicyName": "GreengrassV2TokenExchangeRoleAccess",
    "PolicyId": "ANPAZ2YMUHYHACI7C5Z66",
    "Arn": "arn:aws:iam::123456789012:policy/
GreengrassV2TokenExchangeRoleAccess",
    "Path": "/",
    "DefaultVersionId": "v1",
    "AttachmentCount": 0,
    "PermissionsBoundaryUsageCount": 0,
    "IsAttachable": true,
    "CreateDate": "2021-02-06T00:37:17+00:00",
    "UpdateDate": "2021-02-06T00:37:17+00:00"
```

```
  }
}
```

e. Fügen Sie die IAM-Richtlinie an die Token-Exchange-Rolle an.

- Ersetzen Sie *GreengrassV2TokenExchangeRole* durch den Namen der IAM-Rolle.
- Ersetzen Sie den Richtlinien-ARN durch den ARN der IAM-Richtlinie, die Sie im vorherigen Schritt erstellt haben.

```
aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-
arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess
```

Der Befehl hat keine Ausgabe, wenn die Anforderung erfolgreich ist.

2. Erstellen Sie einen -AWS IoTRollenalias, der auf die Token-Exchange-Rolle verweist.

- Ersetzen Sie durch *GreengrassCoreTokenExchangeRoleAlias* den Namen des zu erstellenden Rollenalias.
- Ersetzen Sie den Rollen-ARN durch den ARN der IAM-Rolle, die Sie im vorherigen Schritt erstellt haben.

```
aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-
arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole
```

Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anforderung erfolgreich ist.

```
{
  "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
}
```

> **Note**
>
> Um einen Rollenalias zu erstellen, müssen Sie über die Berechtigung verfügen, die IAM-Rolle für den Tokenaustausch an zu übergebenAWS IoT. Wenn Sie beim Versuch, einen Rollenalias zu erstellen, eine Fehlermeldung erhalten, überprüfen Sie, ob Ihr AWS Benutzer über diese Berechtigung verfügt. Weitere Informationen finden Sie unter

> Erteilen von Berechtigungen an einen Benutzer zum Übergeben einer Rolle an einen -AWSService im AWS Identity and Access Management -Benutzerhandbuch.

3. Erstellen und fügen Sie eine -AWS IoTRichtlinie an, die es Ihrem Greengrass-Kerngerät ermöglicht, den Rollenalias zu verwenden, um die Token-Exchange-Rolle zu übernehmen. Wenn Sie bereits ein Greengrass-Core-Gerät eingerichtet haben, können Sie seine AWS IoT Rollenaliasrichtlinie anfügen, anstatt eine neue zu erstellen. Gehen Sie wie folgt vor:

   a. (Optional) Erstellen Sie eine Datei, die das AWS IoT Richtliniendokument enthält, das der Rollenalias benötigt.

      Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen der Datei zu verwenden.

      ```
      nano greengrass-v2-iot-role-alias-policy.json
      ```

      Kopieren Sie den folgenden JSON-Code in die Datei .

      - Ersetzen Sie den Ressourcen-ARN durch den ARN Ihres Rollenalias.

      ```
      {
        "Version":"2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Action": "iot:AssumeRoleWithCertificate",
            "Resource": "arn:aws:iot:us-west-2:123456789012:rolealias/
      GreengrassCoreTokenExchangeRoleAlias"
          }
        ]
      }
      ```

   b. Erstellen Sie eine -AWS IoTRichtlinie aus dem Richtliniendokument.

      - Ersetzen Sie *GreengrassCoreTokenExchangeRoleAliasPolicy* durch den Namen der zu erstellenden AWS IoT Richtlinie.

```
aws iot create-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
 --policy-document file://greengrass-v2-iot-role-alias-policy.json
```

Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anforderung erfolgreich ist.

```
{
  "policyName": "GreengrassCoreTokenExchangeRoleAliasPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassCoreTokenExchangeRoleAliasPolicy",
  "policyDocument": "{
    \\"Version\\":\\"2012-10-17\\",
    \\"Statement\\": [
      {
        \\"Effect\\": \\"Allow\\",
        \\"Action\\": \\"iot:AssumeRoleWithCertificate\\",
        \\"Resource\\": \\"arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias\\"
      }
    ]
  }",
  "policyVersionId": "1"
}
```

c. Fügen Sie die AWS IoT Richtlinie an das Objektzertifikat AWS IoT an.

- Ersetzen Sie durch *GreengrassCoreTokenExchangeRoleAliasPolicy* den Namen der Rollen-AWS IoTAliasrichtlinie.
- Ersetzen Sie den Ziel-ARN durch den ARN des Zertifikats für Ihr AWS IoT Objekt.

```
aws iot attach-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
 --target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

Der Befehl hat keine Ausgabe, wenn die Anforderung erfolgreich ist.

Herunterladen von Zertifikaten auf das Gerät

Zuvor haben Sie das Zertifikat Ihres Geräts auf Ihren Entwicklungscomputer heruntergeladen. In diesem Abschnitt laden Sie das Amazon Root Certificate Authority (CA)-Zertifikat herunter. Wenn Sie die AWS IoT Greengrass Core-Software dann in Docker auf einem anderen Computer als Ihrem Entwicklungscomputer ausführen möchten, kopieren Sie die Zertifikate auf diesen Hostcomputer. Die -AWS IoT GreengrassCore-Software verwendet diese Zertifikate, um eine Verbindung zum AWS IoT Cloud-Service herzustellen.

So laden Sie Zertifikate auf das Gerät herunter

1. Laden Sie auf Ihrem Entwicklungscomputer das Amazon Root Certificate Authority (CA)-Zertifikat herunter. -AWS IoTZertifikate sind standardmäßig dem Amazon Root CA-Zertifikat zugeordnet.

   Linux or Unix

   ```
   sudo curl -o ./greengrass-v2-certs/AmazonRootCA1.pem https://
   www.amazontrust.com/repository/AmazonRootCA1.pem
   ```

   Windows Command Prompt (CMD)

   ```
   curl -o .\greengrass-v2-certs\AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   ```

   PowerShell

   ```
   iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile .
   \greengrass-v2-certs\AmazonRootCA1.pem
   ```

2. Wenn Sie die AWS IoT Greengrass Core-Software in Docker auf einem anderen Gerät als Ihrem Entwicklungscomputer ausführen möchten, kopieren Sie die Zertifikate auf den Hostcomputer. Wenn SSH und SCP auf dem Entwicklungscomputer und dem Hostcomputer aktiviert sind, können Sie den `scp` Befehl auf Ihrem Entwicklungscomputer verwenden, um die Zertifikate zu übertragen. Ersetzen Sie durch *`device-ip-address`* die IP-Adresse Ihres Hostcomputers.

   ```
   scp -r greengrass-v2-certs/ device-ip-address:~
   ```

Erstellen einer Konfigurationsdatei

1. Erstellen Sie auf dem Hostcomputer einen Ordner, in dem Sie Ihre Konfigurationsdatei speichern.

   ```
   mkdir ./greengrass-v2-config
   ```

2. Verwenden Sie einen Texteditor, um eine Konfigurationsdatei mit dem Namen `config.yaml` im `./greengrass-v2-config` Ordner zu erstellen.

   Sie können beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen des zu verwenden`config.yaml`.

   ```
   nano ./greengrass-v2-config/config.yaml
   ```

3. Kopieren Sie den folgenden YAML-Inhalt in die Datei . Diese Teilkonfigurationsdatei gibt Systemparameter und Greengrass-Kernparameter an.

   ```
   ---
   system:
     certificateFilePath: "/tmp/certs/device.pem.crt"
     privateKeyPath: "/tmp/certs/private.pem.key"
     rootCaPath: "/tmp/certs/AmazonRootCA1.pem"
     rootpath: "/greengrass/v2"
     thingName: "MyGreengrassCore"
   services:
     aws.greengrass.Nucleus:
       componentType: "NUCLEUS"
       version: "nucleus-version"
       configuration:
         awsRegion: "region"
         iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
         iotDataEndpoint: "device-data-prefix-ats.iot.region.amazonaws.com"
         iotCredEndpoint: "device-credentials-prefix.credentials.region.amazonaws.com"
   ```

   Ersetzen Sie dann die folgenden Werte:

   - */tmp/certs* . Das Verzeichnis im Docker-Container, in das Sie die heruntergeladenen Zertifikate mounten, wenn Sie den Container starten.
   - */greengrass/v2*. Der Greengrass-Stammordner, den Sie für die Installation verwenden möchten. Sie verwenden die GGC_ROOT Umgebungsvariable , um diesen Wert festzulegen.

- *MyGreengrassCore*. Der Name des AWS IoT-Objekts.
- *-Kernversion* . Die zu installierende Version der AWS IoT Greengrass Core-Software. Dieser Wert muss mit der Version des Docker-Images oder der Docker-Datei übereinstimmen, die Sie heruntergeladen haben. Wenn Sie das Greengrass-Docker-Image mit dem `-latest`Tag heruntergeladen haben, verwenden Sie , **`docker inspect`** ***image-id*** um die Image-Version anzuzeigen.
- *Region*. Die AWS-Region, in der Sie Ihre AWS IoT Ressourcen erstellt haben. Sie müssen auch denselben Wert für die `AWS_REGION` Umgebungsvariable in Ihrer Umgebungsdatei angeben.
- *GreengrassCoreTokenExchangeRoleAlias*. Der Token-Exchange-Rollenalias.
- *device-data-prefix*. Das Präfix für Ihren AWS IoT Datenendpunkt.
- *device-credentials-prefix*. Das Präfix für Ihren AWS IoT Anmeldeinformationsendpunkt.

Erstellen einer Umgebungsdatei

In diesem Tutorial wird eine Umgebungsdatei verwendet, um die Umgebungsvariablen festzulegen, die an das AWS IoT Greengrass -Core-Softwareinstallationsprogramm im Docker-Container übergeben werden. Sie können auch das Argument `-e` oder in Ihrem Befehl verwenden, um Umgebungsvariablen im Docker-Container festzulegen, oder Sie können die Variablen in einem `--env` -Block in der -`docker-compose.yml`Datei festlegen. `docker run` `environment`

1. Verwenden Sie einen Texteditor, um eine Umgebungsdatei mit dem Namen zu erstellen`.env`.

   Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen des `.env` im aktuellen Verzeichnis zu verwenden.

   ```
   nano .env
   ```

2. Kopieren Sie den folgenden Inhalt in die Datei .

   ```
   GGC_ROOT_PATH=/greengrass/v2
   AWS_REGION=region
   PROVISION=false
   COMPONENT_DEFAULT_USER=ggc_user:ggc_group
   INIT_CONFIG=/tmp/config/config.yaml
   ```

Ersetzen Sie dann die folgenden Werte.

- */greengrass/v2*. Der Pfad zum Stammordner, der zur Installation der AWS IoT Greengrass Core-Software verwendet werden soll.
- *Region*. Die AWS-Region, in der Sie Ihre AWS IoT Ressourcen erstellt haben. Sie müssen denselben Wert für den `awsRegion` Konfigurationsparameter in Ihrer Konfigurationsdatei angeben.
- */tmp/config/*. Der Ordner, in dem Sie die Konfigurationsdatei mounten, wenn Sie den Docker-Container starten.

> **Note**
>
> Sie können die `DEPLOY_DEV_TOOLS` Umgebungsvariable auf setzen, `true` um die Greengrass-CLI-Komponente bereitzustellen, mit der Sie benutzerdefinierte Komponenten innerhalb des Docker-Containers entwickeln können. Wir empfehlen, diese Komponente nur in Entwicklungsumgebungen und nicht in Produktionsumgebungen zu verwenden. Diese Komponente bietet Zugriff auf Informationen und Vorgänge, die Sie normalerweise in einer Produktionsumgebung nicht benötigen. Folgen Sie dem Prinzip der geringsten Berechtigung, indem Sie diese Komponente nur auf Core-Geräten bereitstellen, auf denen Sie sie benötigen.

Ausführen der AWS IoT Greengrass Core-Software in einem Container

Dieses Tutorial zeigt Ihnen, wie Sie das Docker-Image starten, das Sie in einen Docker-Container erstellt haben. Sie können die Docker CLI oder die Docker Compose CLI verwenden, um das AWS IoT Greengrass Core-Software-Image in einem Docker-Container auszuführen.

Docker

- In diesem Tutorial erfahren Sie, wie Sie das Docker-Image starten, das Sie in einen Docker-Container integriert haben.

```
docker run --rm --init -it --name docker-image \
 -v path/to/greengrass-v2-config:/tmp/config/:ro \
 -v path/to/greengrass-v2-certs:/tmp/certs:ro \
 --env-file .env \
```

```
-p 8883 \
your-container-image:version
```

Dieser Beispielbefehl verwendet die folgenden Argumente für die Docker-Ausführung :

- `--rm`. Bereinigt den Container, wenn er beendet wird.
- `--init`. Verwendet einen Init-Prozess im Container.

> **Note**
>
> Das `--init` Argument ist erforderlich, um die AWS IoT Greengrass Core-Software herunterzufahren, wenn Sie den Docker-Container stoppen.

- `-it`. (Optional) Führt den Docker-Container im Vordergrund als interaktiven Prozess aus. Sie können dies durch das `-d` Argument ersetzen, um den Docker-Container stattdessen im getrennten Modus auszuführen. Weitere Informationen finden Sie unter Trennen im Vergleich zum Vordergrund in der Docker-Dokumentation.
- `--name`. Führt einen Container mit dem Namen aus `aws-iot-greengrass`
- `-v`. Mountet ein Volume im Docker-Container, um die Konfigurationsdatei und die Zertifikatsdateien für die AWS IoT Greengrass Ausführung im Container verfügbar zu machen.
- `--env-file`. (Optional) Gibt die Umgebungsdatei an, um die Umgebungsvariablen festzulegen, die an das AWS IoT Greengrass Core-Softwareinstallationsprogramm im Docker-Container übergeben werden. Dieses Argument ist nur erforderlich, wenn Sie eine Umgebungsdatei zum Festlegen von Umgebungsvariablen erstellt haben. Wenn Sie keine Umgebungsdatei erstellt haben, können Sie `--env` Argumente verwenden, um Umgebungsvariablen direkt in Ihrem Docker-Ausführungsbefehl festzulegen.
- `-p`. (Optional) Veröffentlicht den 8883-Container-Port auf dem Host-Computer. Dieses Argument ist erforderlich, wenn Sie eine Verbindung herstellen und über MQTT kommunizieren möchten, da Port 8883 für MQTT-Datenverkehr AWS IoT Greengrass verwendet. Um andere Ports zu öffnen, verwenden Sie zusätzliche `-p` Argumente.

> **Note**
>
> Um Ihren Docker-Container mit erhöhter Sicherheit auszuführen, können Sie die Argumente `--cap-drop` und verwenden, um die Linux`--cap-add`-Funktionen

für Ihren Container selektiv zu aktivieren. Weitere Informationen finden Sie unter Laufzeitberechtigungen und Linux-Funktionen in der Docker-Dokumentation.

Docker Compose

1. Verwenden Sie einen Texteditor, um eine Docker-Compose-Datei mit dem Namen zu erstellen`docker-compose.yml`.

   Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen des `docker-compose.yml` im aktuellen Verzeichnis zu verwenden.

   ```
   nano docker-compose.yml
   ```

   > **Note**
   >
   > Sie können auch die neueste Version der von bereitgestellten ComposeAWS-Datei von herunterladen und verwendenGitHub.

2. Fügen Sie der Compose-Datei den folgenden Inhalt hinzu. Die Datei sollte ähnlich wie im folgenden Beispiel aussehen. Ersetzen Sie *`your-container-name:version`* durch den Namen Ihres Docker-Images.

   ```
   version: '3.7'

   services:
     greengrass:
       init: true
       build:
         context: .
       container_name: aws-iot-greengrass
       image: your-container-name:version
       volumes:
         - /path/to/greengrass-v2-config:/tmp/config/:ro
         - /path/to/greengrass-v2-certs:/tmp/certs:ro
       env_file: .env
       ports:
         - "8883:8883"
   ```

Die folgenden Parameter in diesem Beispiel für die Compose-Datei sind optional:

- `ports`– Veröffentlicht die 8883-Container-Ports auf dem Host-Computer. Dieser Parameter ist erforderlich, wenn Sie eine Verbindung herstellen und über MQTT kommunizieren möchten, da Port 8883 für MQTT-Datenverkehr AWS IoT Greengrass verwendet.
- `env_file`– Gibt die Umgebungsdatei an, um die Umgebungsvariablen festzulegen, die an das AWS IoT Greengrass Core-Softwareinstallationsprogramm im Docker-Container übergeben werden. Dieser Parameter ist nur erforderlich, wenn Sie eine Umgebungsdatei zum Festlegen von Umgebungsvariablen erstellt haben. Wenn Sie keine Umgebungsdatei erstellt haben, können Sie den Umgebungsparameter verwenden, um die Variablen direkt in Ihrer Compose-Datei festzulegen.

> **Note**
>
> Um Ihren Docker-Container mit erhöhter Sicherheit auszuführen, können Sie `cap_drop` und `cap_add` in Ihrer Compose-Datei verwenden, um die Linux-Funktionen für Ihren Container selektiv zu aktivieren. Weitere Informationen finden Sie unter Laufzeitberechtigungen und Linux-Funktionen in der Docker-Dokumentation.

3. Führen Sie den folgenden Befehl aus, um den Container zu starten.

```
docker-compose -f docker-compose.yml up
```

Nächste Schritte

AWS IoT Greengrass Core-Software wird jetzt in einem Docker-Container ausgeführt. Führen Sie den folgenden Befehl aus, um die Container-ID für den aktuell ausgeführten Container abzurufen.

```
docker ps
```

Sie können dann den folgenden Befehl ausführen, um auf den Container zuzugreifen und AWS IoT Greengrass Core-Software zu erkunden, die im Container ausgeführt wird.

```
docker exec -it container-id /bin/bash
```

Informationen zum Erstellen einer einfachen Komponente finden Sie unter Schritt 4: Entwickeln und Testen einer Komponente auf Ihrem Gerät in Tutorial: Erste Schritte mit AWS IoT Greengrass V2

> **Note**
>
> Wenn Sie verwenden, `docker exec` um Befehle im Docker-Container auszuführen, werden diese Befehle nicht in den Docker-Protokollen protokolliert. Um Ihre Befehle in den Docker-Protokollen zu protokollieren, fügen Sie eine interaktive Shell an den Docker-Container an. Weitere Informationen finden Sie unter Anfügen einer interaktiven Shell an den Docker-Container.

Die AWS IoT Greengrass Core-Protokolldatei heißt `greengrass.log` und befindet sich in */greengrass/v2*`/logs`. Komponentenprotokolldateien befinden sich ebenfalls im selben Verzeichnis. Um Greengrass-Protokolle in ein temporäres Verzeichnis auf dem Host zu kopieren, führen Sie den folgenden Befehl aus:

```
docker cp container-id:/greengrass/v2/logs /tmp/logs
```

Wenn Sie Protokolle beibehalten möchten, nachdem ein Container beendet oder entfernt wurde, empfehlen wir, nur das */greengrass/v2*`/logs` Verzeichnis an das temporäre Protokollverzeichnis auf dem Host zu binden, anstatt das gesamte Greengrass-Verzeichnis zu mounten. Weitere Informationen finden Sie unter Greengrass-Protokolle außerhalb des Docker-Containers beibehalten.

Um einen laufenden AWS IoT Greengrass Docker-Container zu stoppen, führen Sie `docker stop` oder aus`docker-compose -f docker-compose.yml stop`. Diese Aktion sendet `SIGTERM` an den Greengrass-Prozess und beendet alle zugehörigen Prozesse, die im Container gestartet wurden. Der Docker-Container wird mit der `docker-init` ausführbaren Datei als Prozess-PID 1 initialisiert, was beim Entfernen aller verbleibenden Zombie-Prozesse hilft. Weitere Informationen finden Sie unter Specify an init process in der Docker-Dokumentation.

Informationen zur Behebung von Problemen mit der Ausführung AWS IoT Greengrass in einem Docker-Container finden Sie unter Fehlerbehebung bei AWS IoT Greengrass in einem Docker-Container.

Fehlerbehebung bei AWS IoT Greengrass in einem Docker-Container

Verwenden Sie die folgenden Informationen, um Probleme mit der Ausführung AWS IoT Greengrass in einem Docker-Container zu beheben und Probleme mit AWS IoT Greengrass im Docker-Container zu beheben.

Themen

Beheben von Problemen beim Ausführen des Docker-Containers

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

Themen

Fehler: Interaktive Anmeldung von einem Nicht-TTY-Gerät aus nicht möglich

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

Fehler: Unbekannte Optionen: -no-include-email

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

Fehler: Eine Firewall blockiert die Freigabe von Dateien zwischen Fenstern und den Containern.

Möglicherweise erhalten Sie diesen Fehler oder eine `Firewall Detected` Meldung, 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 zur Ausführung autorisiert: 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: Sie haben Ihr Pull-Rate-Limit erreicht

Docker Hub begrenzt die Anzahl der Pull-Anforderungen, die anonyme und kostenlose Docker-Hub-Benutzer stellen können. Wenn Sie die Ratenlimits für anonyme oder freie Pull-Anfragen von Benutzern überschreiten, erhalten Sie einen der folgenden Fehler:

```
ERROR: toomanyrequests: Too Many Requests.
```

```
You have reached your pull rate limit.
```

Um diese Fehler zu beheben, können Sie einige Stunden warten, bevor Sie eine weitere Pull-Anforderung versuchen. Wenn Sie vorhaben, konsistent eine große Anzahl von Pull-Anforderungen einzureichen, finden Sie auf der Docker-Hub-Website Informationen zu Ratenlimits sowie Optionen für die Authentifizierung und Aktualisierung Ihres Docker-Kontos.

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.

Greengrass-Protokolle außerhalb des Docker-Containers beibehalten

Nachdem Sie einen AWS IoT Greengrass Container gestoppt haben, können Sie den folgenden `docker cp` Befehl verwenden, um die Greengrass-Protokolle aus dem Docker-Container in ein temporäres Protokollverzeichnis zu kopieren.

```
docker cp container-id:/greengrass/v2/logs /tmp/logs
```

Um Protokolle auch nach dem Beenden oder Entfernen eines Containers beizubehalten, müssen Sie den AWS IoT Greengrass Docker-Container nach dem Bind-Mounting des */greengrass/v2*`/logs` Verzeichnisses ausführen.

Um das */greengrass/v2*`/logs` Verzeichnis zu binden, führen Sie einen der folgenden Schritte aus, wenn Sie einen neuen AWS IoT Greengrass Docker-Container ausführen.

- Nehmen Sie `-v` */tmp/logs*`:`*/greengrass/v2*`/logs:ro` in Ihren `docker run` Befehl auf.

  Ändern Sie den `-volumes`Block in der Compose-Datei so, dass er die folgende Zeile enthält, bevor Sie Ihren `docker-compose up` Befehl ausführen.

  ```
  volumes:
   - /tmp/logs:/greengrass/v2/logs:ro
  ```

Anschließend können Sie Ihre Protokolle unter */tmp/logs* auf Ihrem Host überprüfen, um Greengrass-Protokolle anzuzeigen, während im Docker-Container ausgeführt AWS IoT Greengrass wird.

Informationen zum Ausführen von Greengrass-Docker-Containern finden Sie unter In AWS IoT Greengrass Docker mit manueller Bereitstellung ausführen und In AWS IoT Greengrass Docker mit automatischer Bereitstellung ausführen

Anfügen einer interaktiven Shell an den Docker-Container

Wenn Sie verwenden, `docker exec` um Befehle im Docker-Container auszuführen, werden diese Befehle nicht in den Docker-Protokollen erfasst. Die Protokollierung Ihrer Befehle in den Docker-Protokollen kann Ihnen helfen, den Status des Greengrass-Docker-Containers zu untersuchen. Führen Sie eine der folgenden Aktionen aus:

- Führen Sie den folgenden Befehl in einem separaten Terminal aus, um die Standardeingabe, -ausgabe und den Fehler Ihres Terminals an den laufenden Container anzuhängen. Auf diese Weise können Sie den Docker-Container von Ihrem aktuellen Terminal aus anzeigen und steuern.

```
docker attach container-id
```

- Führen Sie den folgenden Befehl in einem separaten Terminal aus. Auf diese Weise können Sie Ihre Befehle im interaktiven Modus ausführen, auch wenn der Container nicht angefügt ist.

```
docker exec -it container-id sh -c "command > /proc/1/fd/1"
```

Allgemeine Informationen zur AWS IoT Greengrass Fehlerbehebung finden Sie unter Fehlerbehebung.

Konfigurieren Sie die AWS IoT Greengrass Core-Software

Die AWS IoT Greengrass Core-Software bietet Optionen, mit denen Sie die Software konfigurieren können. Sie können Bereitstellungen erstellen, um die AWS IoT Greengrass Core-Software auf jedem Core-Gerät zu konfigurieren.

Themen

- Stellen Sie die Greengrass Nucleus-Komponente bereit
- Den Greengrass Nucleus als Systemdienst konfigurieren
- Steuern Sie die Speicherzuweisung mit JVM-Optionen
- Konfigurieren Sie den Benutzer, der die Komponenten ausführt
- Konfigurieren Sie die Systemressourcenlimits für Komponenten
- Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy
- Verwenden Sie ein Gerätezertifikat, das von einer privaten Zertifizierungsstelle signiert wurde
- Konfigurieren Sie MQTT-Timeouts und Cache-Einstellungen

Stellen Sie die Greengrass Nucleus-Komponente bereit

AWS IoT Greengrass stellt die AWS IoT Greengrass Core-Software als Komponente bereit, die Sie auf Ihren Greengrass-Core-Geräten bereitstellen können. Sie können eine Bereitstellung erstellen,

um dieselbe Konfiguration auf mehrere Greengrass-Core-Geräte anzuwenden. Weitere Informationen finden Sie unter Grüngraskern und Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Den Greengrass Nucleus als Systemdienst konfigurieren

Sie müssen die AWS IoT Greengrass Core-Software als Systemdienst im Init-System Ihres Geräts konfigurieren, um Folgendes zu tun:

- Starten Sie die AWS IoT Greengrass Core-Software, wenn das Gerät bootet. Dies ist eine gute Methode, wenn Sie große Geräteflotten verwalten.
- Installieren und starten Sie die Plugin-Komponenten. Bei mehreren der AWS bereitgestellten Komponenten handelt es sich um Plugin-Komponenten, sodass sie direkt mit dem Greengrass-Nucleus verbunden werden können. Weitere Informationen zu Komponententypen finden Sie unter. Komponententypen
- Wenden Sie over-the-air (OTA) -Updates auf die Core-Software des AWS IoT Greengrass Kerngeräts an. Weitere Informationen finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).
- Ermöglichen Sie es den Komponenten, die AWS IoT Greengrass Core-Software oder das Kerngerät neu zu starten, wenn eine Bereitstellung die Komponente auf eine neue Version aktualisiert oder bestimmte Konfigurationsparameter aktualisiert. Weitere Informationen finden Sie im Schritt zum Bootstrap-Lebenszyklus.

> **Important**
>
> Auf Windows Core-Geräten müssen Sie die AWS IoT Greengrass Core-Software als Systemdienst einrichten.

Themen

- Konfigurieren Sie den Nucleus als Systemdienst (Linux)
- Konfigurieren Sie den Nucleus als Systemdienst (Windows)

Konfigurieren Sie den Nucleus als Systemdienst (Linux)

Linux-Geräte unterstützen verschiedene Init-Systeme wie initd, systemd und SystemV. Sie verwenden das `--setup-system-service true` Argument bei der Installation der AWS IoT

Greengrass Core-Software, um den Nucleus als Systemdienst zu starten und ihn so zu konfigurieren, dass er beim Booten des Geräts gestartet wird. Das Installationsprogramm konfiguriert die AWS IoT Greengrass Core-Software als Systemdienst mit systemd.

Sie können den Nucleus auch manuell so konfigurieren, dass er als Systemdienst ausgeführt wird. Das folgende Beispiel ist eine Servicedatei für systemd.

```
[Unit]
Description=Greengrass Core

[Service]
Type=simple
PIDFile=/greengrass/v2/alts/loader.pid
RemainAfterExit=no
Restart=on-failure
RestartSec=10
ExecStart=/bin/sh /greengrass/v2/alts/current/distro/bin/loader

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

Nachdem Sie den Systemdienst konfiguriert haben, können Sie die folgenden Befehle ausführen, um das Starten des Geräts beim Systemstart und das Starten oder Beenden der AWS IoT Greengrass Core-Software zu konfigurieren.

- Um den Status des Dienstes zu überprüfen (systemd)

  ```
  sudo systemctl status greengrass.service
  ```

- Damit der Nucleus beim Booten des Geräts starten kann.

  ```
  sudo systemctl enable greengrass.service
  ```

- Um zu verhindern, dass der Nucleus beim Booten des Geräts startet.

  ```
  sudo systemctl disable greengrass.service
  ```

- Um die AWS IoT Greengrass Core-Software zu starten.

  ```
  sudo systemctl start greengrass.service
  ```

- Um die AWS IoT Greengrass Core-Software zu beenden.

```
sudo systemctl stop greengrass.service
```

Konfigurieren Sie den Nucleus als Systemdienst (Windows)

Sie verwenden das `--setup-system-service true` Argument bei der Installation der AWS IoT Greengrass Core-Software, um den Nucleus als Windows-Dienst zu starten und ihn so zu konfigurieren, dass er beim Booten des Geräts gestartet wird.

Nachdem Sie den Dienst konfiguriert haben, können Sie die folgenden Befehle ausführen, um das Starten des Geräts beim Systemstart und das Starten oder Beenden der AWS IoT Greengrass Core-Software zu konfigurieren. Sie müssen die Befehlszeile oder PowerShell als Administrator ausführen, um diese Befehle ausführen zu können.

Windows Command Prompt (CMD)

- Um den Status des Dienstes zu überprüfen

  ```
  sc query "greengrass"
  ```

- Damit der Nucleus beim Booten des Geräts starten kann.

  ```
  sc config "greengrass" start=auto
  ```

- Um zu verhindern, dass der Nucleus beim Booten des Geräts startet.

  ```
  sc config "greengrass" start=disabled
  ```

- Um die AWS IoT Greengrass Core-Software zu starten.

  ```
  sc start "greengrass"
  ```

- Um die AWS IoT Greengrass Core-Software zu beenden.

  ```
  sc stop "greengrass"
  ```

> **Note**
>
> Auf Windows-Geräten ignoriert die AWS IoT Greengrass Core-Software dieses Abschaltsignal, während sie die Prozesse der Greengrass-Komponenten herunterfährt. Wenn die AWS IoT Greengrass Core-Software das Signal zum Herunterfahren ignoriert, wenn Sie diesen Befehl ausführen, warten Sie einige Sekunden und versuchen Sie es erneut.

PowerShell

- Um den Status des Dienstes zu überprüfen

```
Get-Service -Name "greengrass"
```

- Damit der Nucleus beim Booten des Geräts starten kann.

```
Set-Service -Name "greengrass" -Status stopped -StartupType automatic
```

- Um zu verhindern, dass der Nucleus beim Booten des Geräts startet.

```
Set-Service -Name "greengrass" -Status stopped -StartupType disabled
```

- Um die AWS IoT Greengrass Core-Software zu starten.

```
Start-Service -Name "greengrass"
```

- Um die AWS IoT Greengrass Core-Software zu beenden.

```
Stop-Service -Name "greengrass"
```

> **Note**
>
> Auf Windows-Geräten ignoriert die AWS IoT Greengrass Core-Software dieses Abschaltsignal, während sie die Prozesse der Greengrass-Komponenten herunterfährt. Wenn die AWS IoT Greengrass Core-Software das Signal zum Herunterfahren ignoriert,

wenn Sie diesen Befehl ausführen, warten Sie einige Sekunden und versuchen Sie es erneut.

Steuern Sie die Speicherzuweisung mit JVM-Optionen

Wenn Sie AWS IoT Greengrass auf einem Gerät mit begrenztem Arbeitsspeicher arbeiten, können Sie die Optionen der Java Virtual Machine (JVM) verwenden, um die maximale Heap-Größe, die Garbage-Collection-Modi und die Compiler-Optionen zu steuern, mit denen die Speichermenge gesteuert wird, die die Core-Software verwendet. AWS IoT Greengrass Die Heap-Größe in der JVM bestimmt, wie viel Speicher eine Anwendung verwenden kann, bevor die Speicherbereinigung erfolgt oder bevor der Anwendung der Arbeitsspeicher ausgeht. Die maximale Heap-Größe gibt die maximale Größe des Arbeitsspeichers an, die bei einer großen Auslastung von der JVM für die Heap-Erweiterung zugewiesen werden kann.

Um die Speicherzuweisung zu steuern, erstellen Sie eine neue Bereitstellung oder überarbeiten Sie eine bestehende Bereitstellung, die die Nucleus-Komponente enthält, und geben Sie Ihre JVM-Optionen im `jvmOptions` Konfigurationsparameter in der Nucleus-Komponentenkonfiguration an.

Je nach Ihren Anforderungen können Sie die AWS IoT Greengrass Core-Software mit reduzierter Speicherzuweisung oder mit minimaler Speicherzuweisung ausführen.

Reduzierte Speicherzuweisung

Um die AWS IoT Greengrass Core-Software mit reduzierter Speicherzuweisung auszuführen, empfehlen wir Ihnen, das folgende Beispiel für ein Update zur Zusammenführung von Konfigurationen zu verwenden, um die JVM-Optionen in Ihrer Nucleus-Konfiguration festzulegen:

```
{
  "jvmOptions": "-Xmx64m -XX:+UseSerialGC -XX:TieredStopAtLevel=1"
}
```

Minimale Speicherzuweisung

Um die AWS IoT Greengrass Core-Software mit minimaler Speicherzuweisung auszuführen, empfehlen wir Ihnen, das folgende Beispiel für ein Update zur Zusammenführung von Konfigurationen zu verwenden, um die JVM-Optionen in Ihrer Nucleus-Konfiguration festzulegen:

```
{
```

```
  "jvmOptions": "-Xmx32m -XX:+UseSerialGC -Xint"
}
```

Diese Beispielupdates für die Zusammenführung von Konfigurationen verwenden die folgenden JVM-Optionen:

`-Xmx`*NN*`m`

Legt die maximale JVM-Heap-Größe fest.

Verwenden Sie für eine reduzierte Speicherzuweisung `-Xmx64m` als Startwert, um die Heap-Größe auf 64 MB zu begrenzen. Verwenden Sie für eine minimale Speicherzuweisung den Wert `-Xmx32m` als Startwert, um die Heap-Größe auf 32 MB zu begrenzen.

Sie können den `-Xmx` Wert je nach Ihren tatsächlichen Anforderungen erhöhen oder verringern. Es wird jedoch dringend empfohlen, die maximale Heap-Größe nicht unter 16 MB festzulegen. Wenn die maximale Heap-Größe für Ihre Umgebung zu niedrig ist, kann es bei der AWS IoT Greengrass Core-Software zu unerwarteten Fehlern kommen, da nicht genügend Arbeitsspeicher zur Verfügung steht.

`-XX:+UseSerialGC`

Gibt an, dass die serielle Garbage-Collection für den JVM-Heap-Speicherplatz verwendet werden soll. Der serielle Garbage-Collector ist langsamer, benötigt aber weniger Speicher als andere JVM-Garbage-Collection-Implementierungen.

`-XX:TieredStopAtLevel=1`

Weist die JVM an, den Java just-in-time (JIT) -Compiler einmal zu verwenden. Da JIT-kompilierter Code Speicherplatz im Gerätespeicher belegt, verbraucht die mehrfache Verwendung des JIT-Compilers mehr Speicher als eine einzelne Kompilierung.

`-Xint`

Weist die JVM an, den just-in-time (JIT-) Compiler nicht zu verwenden. Stattdessen wird die JVM nur im interpretierten Modus ausgeführt. Dieser Modus ist langsamer als das Ausführen von JIT-kompiliertem Code. Der kompilierte Code belegt jedoch keinen Speicherplatz.

Hinweise zum Erstellen von Updates zur Zusammenführung von Konfigurationen finden Sie unter[Komponentenkonfigurationen aktualisieren](#).

Konfigurieren Sie den Benutzer, der die Komponenten ausführt

Die AWS IoT Greengrass Core-Software kann Komponentenprozesse als Systembenutzer und Gruppe ausführen, die sich von der Gruppe unterscheiden, die die Software ausführt. Dies erhöht die Sicherheit, da Sie die AWS IoT Greengrass Core-Software als Root-Benutzer oder als Administratorbenutzer ausführen können, ohne Komponenten, die auf dem Kerngerät ausgeführt werden, diese Berechtigungen zu erteilen.

In der folgenden Tabelle ist angegeben, welche Komponententypen die AWS IoT Greengrass Core-Software als von Ihnen angegebener Benutzer ausführen kann. Weitere Informationen finden Sie unter Komponententypen.

Komponententyp	Konfigurieren Sie den Komponentenbenutzer
Nucleus	 Nein
Plug-In	 Nein
Generisch	 Ja
Lambda (nicht containerisiert)	 Ja

Komponententyp	Konfigurieren Sie den Komponentenbenutzer
Lambda (containerisiert)	 Ja

Sie müssen den Komponentenbenutzer erstellen, bevor Sie ihn in einer Bereitstellungskonfiguration angeben können. Auf Windows-Geräten müssen Sie außerdem den Benutzernamen und das Kennwort für den Benutzer in der Credential Manager-Instanz des Kontos speichern. LocalSystem Weitere Informationen finden Sie unter Richten Sie einen Komponentenbenutzer auf Windows-Geräten ein.

Wenn Sie den Komponentenbenutzer auf einem Linux-basierten Gerät konfigurieren, können Sie optional auch eine Gruppe angeben. Sie geben den Benutzer und die Gruppe, getrennt durch einen Doppelpunkt (`:`), im folgenden Format an:*`user`*`:`*`group`*. Wenn Sie keine Gruppe angeben, verwendet die AWS IoT Greengrass Core-Software standardmäßig die primäre Gruppe des Benutzers. Sie können entweder den Namen oder die ID verwenden, um den Benutzer und die Gruppe zu identifizieren.

Auf Linux-basierten Geräten können Sie Komponenten auch als Systembenutzer ausführen, der nicht existiert (auch als unbekannter Benutzer bezeichnet), um die Sicherheit zu erhöhen. Ein Linux-Prozess kann jeden anderen Prozess signalisieren, der von demselben Benutzer ausgeführt wird. Ein unbekannter Benutzer führt keine anderen Prozesse aus. Sie können also Komponenten als unbekannter Benutzer ausführen, um zu verhindern, dass Komponenten andere Komponenten auf dem Kerngerät signalisieren. Um Komponenten als unbekannter Benutzer auszuführen, geben Sie eine Benutzer-ID an, die auf dem Kerngerät nicht existiert. Sie können auch eine Gruppen-ID angeben, die nicht existiert, um sie als unbekannte Gruppe auszuführen.

Sie können den Benutzer für jede Komponente und für jedes Kerngerät konfigurieren.

- Für eine Komponente konfigurieren

  Sie können jede Komponente so konfigurieren, dass sie mit einem für diese Komponente spezifischen Benutzer ausgeführt wird. Wenn Sie eine Einrichtung erstellen, können Sie den Benutzer für jede Komponente in der `runWith` Konfiguration für diese Komponente angeben. Die AWS IoT Greengrass Core-Software führt Komponenten als der angegebene Benutzer aus, wenn Sie sie konfigurieren. Andernfalls werden Komponenten standardmäßig als

Standardbenutzer ausgeführt, den Sie für das Kerngerät konfigurieren. Weitere Informationen zur Angabe des Komponentenbenutzers in der Bereitstellungskonfiguration finden Sie unter dem `runWith`Konfigurationsparameter unterErstellen von Bereitstellungen.

- Konfigurieren Sie den Standardbenutzer für ein Kerngerät

  Sie können einen Standardbenutzer konfigurieren, den die AWS IoT Greengrass Core-Software zum Ausführen von Komponenten verwendet. Wenn die AWS IoT Greengrass Core-Software eine Komponente ausführt, prüft sie, ob Sie einen Benutzer für diese Komponente angegeben haben, und verwendet ihn, um die Komponente auszuführen. Wenn in der Komponente kein Benutzer angegeben ist, führt die AWS IoT Greengrass Core-Software die Komponente als Standardbenutzer aus, den Sie für das Kerngerät konfiguriert haben. Weitere Informationen finden Sie unter Konfigurieren Sie den Standardkomponentenbenutzer.

> **Note**
>
> Auf Windows-basierten Geräten müssen Sie mindestens einen Standardbenutzer angeben, um Komponenten auszuführen.
>
> Auf Linux-basierten Geräten gelten die folgenden Überlegungen, wenn Sie einen Benutzer nicht für die Ausführung von Komponenten konfigurieren:
>
> - Wenn Sie die AWS IoT Greengrass Core-Software als Root-Benutzer ausführen, führt die Software keine Komponenten aus. Sie müssen einen Standardbenutzer für die Ausführung von Komponenten angeben, wenn Sie sie als Root ausführen.
> - Wenn Sie die AWS IoT Greengrass Core-Software als Benutzer ausführen, der kein Root-Benutzer ist, führt die Software die Komponenten als dieser Benutzer aus.

Themen

Richten Sie einen Komponentenbenutzer auf Windows-Geräten ein

So richten Sie einen Komponentenbenutzer auf einem Windows-basierten Gerät ein

1. Erstellen Sie den Komponentenbenutzer im LocalSystem Konto auf dem Gerät.

```
net user /add component-user password
```

2. Verwenden Sie das PsExec Microsoft-Hilfsprogramm, um den Benutzernamen und das Passwort für den Komponentenbenutzer in der Credential Manager-Instanz für das LocalSystem Konto zu speichern.

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

> **Note**
>
> Auf Windows-basierten Geräten wird auf dem LocalSystem Konto der Greengrass-Nucleus ausgeführt, und Sie müssen das PsExec Hilfsprogramm verwenden, um die Benutzerinformationen der Komponente im Konto zu speichern. LocalSystem Wenn Sie die Credential Manager-Anwendung verwenden, werden diese Informationen nicht im Konto, sondern im Windows-Konto des aktuell angemeldeten Benutzers gespeichert. LocalSystem

Konfigurieren Sie den Standardkomponentenbenutzer

Sie können eine Bereitstellung verwenden, um den Standardbenutzer auf einem Kerngerät zu konfigurieren. In dieser Bereitstellung aktualisieren Sie die Nucleus-Komponentenkonfiguration.

> **Note**
>
> Mit der `--component-default-user` Option können Sie auch den Standardbenutzer festlegen, wenn Sie die AWS IoT Greengrass Core-Software installieren. Weitere Informationen finden Sie unter Installieren Sie die AWS IoT Greengrass Core-Software..

Erstellen Sie eine Bereitstellung, die das folgende Konfigurationsupdate für die `aws.greengrass.Nucleus` Komponente angibt.

Linux

```
{
  "runWithDefault": {
```

```
    "posixUser": "ggc_user:ggc_group"
  }
}
```

Windows

```
{
  "runWithDefault": {
    "windowsUser": "ggc_user"
  }
}
```

> **Note**
>
> Der von Ihnen angegebene Benutzer muss existieren, und der Benutzername und das Kennwort für diesen Benutzer müssen in der Credential Manager-Instanz des LocalSystem Kontos auf Ihrem Windows-Gerät gespeichert sein. Weitere Informationen finden Sie unter Richten Sie einen Komponentenbenutzer auf Windows-Geräten ein.

Das folgende Beispiel definiert eine Bereitstellung für ein Linux-basiertes Gerät, das als Standardbenutzer und `ggc_user` `ggc_group` als Standardgruppe konfiguriert wird. Das `merge` Konfigurationsupdate erfordert ein serialisiertes JSON-Objekt.

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.12.6",
      "configurationUpdate": {
        "merge": "{\"runWithDefault\":{\"posixUser\":\"ggc_user:ggc_group\"}}"
      }
    }
  }
}
```

Konfigurieren Sie die Systemressourcenlimits für Komponenten

> **Note**
>
> Diese Funktion ist für Version 2.4.0 und höher der Greengrass Nucleus-Komponente verfügbar. AWS IoT Greengrass unterstützt diese Funktion derzeit nicht auf Windows Core-Geräten.

Sie können die maximale CPU- und RAM-Auslastung konfigurieren, die die Prozesse der einzelnen Komponenten auf dem Kerngerät verwenden können.

Die folgende Tabelle zeigt die Komponententypen, die Systemressourcenlimits unterstützen. Weitere Informationen finden Sie unter Komponententypen.

Komponententyp	Konfigurieren Sie Systemressourcenlimits
Nucleus	 Nein
Plug-In	 Nein
Generisch	 Ja
Lambda (nicht containerisiert)	 Ja

Komponententyp	Konfigurieren Sie Systemressourcenlimits
Lambda (containerisiert)	 Nein

> **Important**
>
> Systemressourcenlimits werden nicht unterstützt, wenn Sie die AWS IoT Greengrass Core-Software in einem Docker-Container ausführen.

Sie können Systemressourcenlimits für jede Komponente und für jedes Kerngerät konfigurieren.

- Für eine Komponente konfigurieren

  Sie können jede Komponente mit spezifischen Systemressourcenlimits für diese Komponente konfigurieren. Wenn Sie eine Bereitstellung erstellen, können Sie die Systemressourcenlimits für jede Komponente in der Bereitstellung angeben. Wenn die Komponente Systemressourcenlimits unterstützt, wendet die AWS IoT Greengrass Core-Software die Grenzwerte auf die Prozesse der Komponente an. Wenn Sie keine Systemressourcenlimits für eine Komponente angeben, verwendet die AWS IoT Greengrass Core-Software alle Standardeinstellungen, die Sie für das Kerngerät konfiguriert haben. Weitere Informationen finden Sie unter Erstellen von Bereitstellungen.

- Konfigurieren Sie die Standardeinstellungen für ein Kerngerät

  Sie können die Standardgrenzwerte für Systemressourcen konfigurieren, die die AWS IoT Greengrass Core-Software für Komponenten anwendet, die diese Grenzwerte unterstützen. Wenn die AWS IoT Greengrass Core-Software eine Komponente ausführt, wendet sie die Systemressourcenlimits an, die Sie für diese Komponente angeben. Wenn diese Komponente keine Systemressourcenlimits festlegt, wendet die AWS IoT Greengrass Core-Software die standardmäßigen Systemressourcenlimits an, die Sie für das Kerngerät konfigurieren. Wenn Sie keine standardmäßigen Systemressourcenlimits angeben, wendet die AWS IoT Greengrass Core-Software standardmäßig keine Systemressourcenlimits an. Weitere Informationen finden Sie unter Konfigurieren Sie die Standardgrenzwerte für Systemressourcen.

Konfigurieren Sie die Standardgrenzwerte für Systemressourcen

Sie können die Greengrass Nucleus-Komponente einsetzen, um die standardmäßigen Systemressourcenlimits für ein Kerngerät zu konfigurieren. Um die standardmäßigen Systemressourcenlimits zu konfigurieren, erstellen Sie eine Bereitstellung, die das folgende Konfigurationsupdate für die `aws.greengrass.Nucleus` Komponente spezifiziert.

```
{
  "runWithDefault": {
    "systemResourceLimits": {
      "cpu": cpuTimeLimit,
      "memory": memoryLimitInKb
    }
  }
}
```

Das folgende Beispiel definiert eine Bereitstellung, bei der das CPU-Zeitlimit auf konfiguriert wird2, was einer Auslastung von 50% auf einem Gerät mit 4 CPU-Kernen entspricht. In diesem Beispiel wird auch die Speichernutzung auf 100 MB konfiguriert.

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.12.6",
      "configurationUpdate": {
        "merge": "{\"runWithDefault\":{\"systemResourceLimits\":\"cpus\":2,\"memory
\":102400}}}"
      }
    }
  }
}
```

Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy

AWS IoT Greengrass Kerngeräte kommunizieren über das MQTT-Messaging-Protokoll mit TLS-Client-Authentifizierung mit. AWS IoT Core 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. In

anderen restriktiven Umgebungen muss der gesamte Datenverkehr möglicherweise über einen Proxy geleitet werden, bevor eine Verbindung zum Internet hergestellt wird.

> **Note**
>
> Greengrass-Core-Geräte, auf denen Greengrass Nucleus Component v2.0.3 und früher ausgeführt wird, verwenden Port 8443, um eine Verbindung zum Datenebenen-Endpunkt herzustellen. AWS IoT Greengrass Diese Geräte müssen in der Lage sein, über Port 8443 eine Verbindung zu diesem Endpunkt herzustellen. Weitere Informationen finden Sie unter Zulassen von Gerätedatenverkehr über einen Proxy oder eine Firewall.

AWS IoT Greengrass Bietet die folgenden Konfigurationsoptionen, um die Kommunikation in diesen Szenarien zu ermöglichen:

- MQTT-Kommunikation über Port 443. Wenn Ihr Netzwerk Verbindungen zu Port 443 zulässt, können Sie das Greengrass-Core-Gerät so konfigurieren, dass es Port 443 für MQTT-Verkehr anstelle des Standardports 8883 verwendet. Dabei kann es sich um eine direkte Verbindung mit Port 443 oder eine Verbindung über einen Netzwerk-Proxy-Server handeln. Im Gegensatz zur Standardkonfiguration, die eine zertifikatsbasierte Client-Authentifizierung verwendet, verwendet MQTT auf Port 443 die Gerätedienstrolle für die Authentifizierung.

  Weitere Informationen finden Sie unter Konfigurieren Sie MQTT über Port 443.
- HTTPS-Kommunikation über Port 443. Die AWS IoT Greengrass Core-Software sendet standardmäßig HTTPS-Verkehr über Port 8443, aber Sie können sie so konfigurieren, dass Port 443 verwendet wird. AWS IoT Greengrass verwendet die TLS-Erweiterung Application Layer Protocol Network (ALPN), um diese Verbindung zu aktivieren. Wie bei der Standardkonfiguration verwendet HTTPS auf Port 443 die zertifikatsbasierte Clientauthentifizierung.

  > **Important**
  >
  > Um ALPN zu verwenden und die HTTPS-Kommunikation über Port 443 zu aktivieren, muss auf Ihrem Kerngerät Java 8 Update 252 oder höher ausgeführt werden. Alle Updates von Java Version 9 und höher unterstützen auch ALPN.

  Weitere Informationen finden Sie unter Konfigurieren Sie HTTPS über Port 443.

- Verbindung über einen Netzwerk-Proxy. Sie können einen Netzwerk-Proxyserver so konfigurieren, dass er als Vermittler für die Verbindung zum Greengrass-Core-Gerät fungiert. AWS IoT Greengrass unterstützt die Standardauthentifizierung für HTTP- und HTTPS-Proxys.

  Greengrass-Core-Geräte müssen Greengrass Nucleus v2.5.0 oder höher ausführen, um HTTPS-Proxys verwenden zu können.

  Die AWS IoT Greengrass Core-Software übergibt die Proxykonfiguration über die UmgebungsvariablenALL_PROXY,, HTTP_PROXY und an die Komponenten. HTTPS_PROXY NO_PROXY Komponenten müssen diese Einstellungen verwenden, um eine Verbindung über den Proxy herzustellen. Komponenten verwenden gängige Bibliotheken (wie boto3, cURL und das `requests` Python-Paket), die normalerweise diese Umgebungsvariablen standardmäßig verwenden, um Verbindungen herzustellen. Wenn eine Komponente auch diese Umgebungsvariablen spezifiziert, überschreibt AWS IoT Greengrass sie sie nicht.

  Weitere Informationen finden Sie unter Konfigurieren Sie einen Netzwerk-Proxy.

Konfigurieren Sie MQTT über Port 443

Sie können MQTT über Port 443 auf vorhandenen Kerngeräten konfigurieren oder wenn Sie die AWS IoT Greengrass Core-Software auf einem neuen Kerngerät installieren.

Themen

- Konfigurieren Sie MQTT über Port 443 auf vorhandenen Kerngeräten
- Konfigurieren Sie MQTT während der Installation über Port 443

Konfigurieren Sie MQTT über Port 443 auf vorhandenen Kerngeräten

Sie können eine Bereitstellung verwenden, um MQTT über Port 443 auf einem einzelnen Core-Gerät oder einer Gruppe von Core-Geräten zu konfigurieren. In dieser Bereitstellung aktualisieren Sie die Nucleus-Komponentenkonfiguration. Der Nucleus wird neu gestartet, wenn Sie seine `mqtt` Konfiguration aktualisieren.

Um MQTT über Port 443 zu konfigurieren, erstellen Sie ein Deployment, das das folgende Konfigurationsupdate für die `aws.greengrass.Nucleus` Komponente spezifiziert.

```
{
  "mqtt": {
    "port": 443
```

```
  }
}
```

Das folgende Beispiel definiert eine Bereitstellung, die MQTT über Port 443 konfiguriert. Das `merge` Konfigurationsupdate erfordert ein serialisiertes JSON-Objekt.

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.12.6",
      "configurationUpdate": {
        "merge": "{\"mqtt\":{\"port\":443}}"
      }
    }
  }
}
```

Konfigurieren Sie MQTT während der Installation über Port 443

Sie können MQTT über Port 443 konfigurieren, wenn Sie die AWS IoT Greengrass Core-Software auf einem Core-Gerät installieren. Verwenden Sie das `--init-config` Installer-Argument, um MQTT über Port 443 zu konfigurieren. Sie können dieses Argument angeben, wenn Sie die Installation mit manueller Bereitstellung, Flottenbereitstellung oder benutzerdefinierter Bereitstellung durchführen.

Konfigurieren Sie HTTPS über Port 443

Für diese Funktion ist Version Grüngraskern 2.0.4 oder höher erforderlich.

Sie können HTTPS über Port 443 auf vorhandenen Core-Geräten oder bei der Installation der AWS IoT Greengrass Core-Software auf einem neuen Core-Gerät konfigurieren.

Themen

- Konfigurieren Sie HTTPS über Port 443 auf vorhandenen Core-Geräten
- Konfigurieren Sie HTTPS während der Installation über Port 443

Konfigurieren Sie HTTPS über Port 443 auf vorhandenen Core-Geräten

Sie können eine Bereitstellung verwenden, um HTTPS über Port 443 auf einem einzelnen Core-Gerät oder einer Gruppe von Core-Geräten zu konfigurieren. In dieser Bereitstellung aktualisieren Sie die Nucleus-Komponentenkonfiguration.

Um HTTPS über Port 443 zu konfigurieren, erstellen Sie eine Bereitstellung, die das folgende Konfigurationsupdate für die `aws.greengrass.Nucleus` Komponente spezifiziert.

```
{
  "greengrassDataPlanePort": 443
}
```

Das folgende Beispiel definiert eine Bereitstellung, die HTTPS über Port 443 konfiguriert. Das `merge` Konfigurationsupdate erfordert ein serialisiertes JSON-Objekt.

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.12.6",
      "configurationUpdate": {
        "merge": "{\"greengrassDataPlanePort\":443}"
      }
    }
  }
}
```

Konfigurieren Sie HTTPS während der Installation über Port 443

Sie können HTTPS über Port 443 konfigurieren, wenn Sie die AWS IoT Greengrass Core-Software auf einem Core-Gerät installieren. Verwenden Sie das `--init-config` Installer-Argument, um HTTPS über Port 443 zu konfigurieren. Sie können dieses Argument angeben, wenn Sie die Installation mit manueller Bereitstellung, Flottenbereitstellung oder benutzerdefinierter Bereitstellung durchführen.

Konfigurieren Sie einen Netzwerk-Proxy

Gehen Sie wie in diesem Abschnitt beschrieben vor, um Greengrass-Core-Geräte so zu konfigurieren, dass sie sich über einen HTTP- oder HTTPS-Netzwerk-Proxy mit dem Internet verbinden. Weitere Informationen zu den Endpunkten und Anschlüssen, die Kerngeräte verwenden, finden Sie unter. Zulassen von Gerätedatenverkehr über einen Proxy oder eine Firewall

> **Important**
>
> Wenn auf Ihrem Kerngerät eine Version von Greengrass Nucleus vor v2.4.0 ausgeführt wird, muss die Rolle Ihres Geräts die folgenden Berechtigungen für die Verwendung eines Netzwerk-Proxys zulassen:

- `iot:Connect`
- `iot:Publish`
- `iot:Receive`
- `iot:Subscribe`

Dies ist notwendig, da das Gerät AWS Anmeldeinformationen vom Token-Austauschdienst verwendet, um MQTT-Verbindungen zu authentifizieren. AWS IoT Das Gerät verwendet MQTT, um Bereitstellungen vom zu empfangen und zu installieren. Ihr Gerät funktioniert also nur AWS Cloud, wenn Sie diese Berechtigungen für seine Rolle definieren. Geräte verwenden normalerweise X.509-Zertifikate, um MQTT-Verbindungen zu authentifizieren, aber Geräte können dies nicht tun, um sich zu authentifizieren, wenn sie einen Proxy verwenden.
Weitere Informationen zur Konfiguration der Geräterolle finden Sie unter. Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS

Themen

Konfigurieren Sie einen Netzwerk-Proxy auf vorhandenen Core-Geräten

Sie können eine Bereitstellung verwenden, um einen Netzwerk-Proxy auf einem einzelnen Core-Gerät oder einer Gruppe von Core-Geräten zu konfigurieren. In dieser Bereitstellung aktualisieren Sie die Nucleus-Komponentenkonfiguration. Der Nucleus wird neu gestartet, wenn Sie seine `networkProxy` Konfiguration aktualisieren.

Um einen Netzwerk-Proxy zu konfigurieren, erstellen Sie ein Deployment für die `aws.greengrass.Nucleus` Komponente, die das folgende Konfigurationsupdate zusammenführt. Dieses Konfigurationsupdate enthält das NetworkProxy-Objekt.

```
{
  "networkProxy": {
```

```
    "noProxyAddresses": "http://192.168.0.1,www.example.com",
    "proxy": {
      "url": "https://my-proxy-server:1100"
    }
  }
}
```

Das folgende Beispiel definiert eine Bereitstellung, die einen Netzwerk-Proxy konfiguriert. Das `merge` Konfigurationsupdate erfordert ein serialisiertes JSON-Objekt.

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.12.6",
      "configurationUpdate": {
        "merge": "{\"networkProxy\":{\"noProxyAddresses\":
\"http://192.168.0.1,www.example.com\",\"proxy\":{\"url\":\"https://my-proxy-
server:1100\",\"username\":\"Mary_Major\",\"password\":\"pass@word1357\"}}}"
      }
    }
  }
}
```

Konfigurieren Sie während der Installation einen Netzwerk-Proxy

Sie können einen Netzwerk-Proxy konfigurieren, wenn Sie die AWS IoT Greengrass Core-Software auf einem Core-Gerät installieren. Verwenden Sie das `--init-config` Installer-Argument, um den Netzwerk-Proxy zu konfigurieren. Sie können dieses Argument angeben, wenn Sie die Installation mit manueller Bereitstellung, Flottenbereitstellung oder benutzerdefinierter Bereitstellung durchführen.

Ermöglichen Sie dem Kerngerät, einem HTTPS-Proxy zu vertrauen

Wenn Sie ein Kerngerät für die Verwendung eines HTTPS-Proxys konfigurieren, müssen Sie die Zertifikatskette des Proxyservers zu der Zertifikatskette des Kerngeräts hinzufügen, damit es dem HTTPS-Proxy vertrauen kann. Andernfalls könnten auf dem Kerngerät Fehler auftreten, wenn es versucht, den Verkehr über den Proxy weiterzuleiten. Fügen Sie das Proxy-Server-CA-Zertifikat zur Amazon-Root-CA-Zertifikatsdatei des Kerngeräts hinzu.

Um dem Kerngerät zu ermöglichen, dem HTTPS-Proxy zu vertrauen

1. Suchen Sie die Amazon-Root-CA-Zertifikatsdatei auf dem Core-Gerät.

- Wenn Sie die AWS IoT Greengrass Core-Software mit automatischer Bereitstellung installiert haben, befindet sich die Amazon-Root-CA-Zertifikatsdatei unter*/greengrass/v2/*`rootCA.pem`.
- Wenn Sie die AWS IoT Greengrass Core-Software mit manueller oder Flottenbereitstellung installiert haben, befindet sich die Amazon-Root-CA-Zertifikatsdatei möglicherweise unter*/greengrass/v2/*`AmazonRootCA1.pem`.

Wenn das Amazon-Root-CA-Zertifikat an diesen Standorten nicht vorhanden ist, überprüfen Sie die `system.rootCaPath` Immobilie unter, */greengrass/v2/*`config/effectiveConfig.yaml` um den Speicherort zu ermitteln.

2. Fügen Sie den Inhalt der Proxyserver-CA-Zertifikatsdatei zur Amazon-Root-CA-Zertifikatsdatei hinzu.

   Das folgende Beispiel zeigt ein Proxyserver-CA-Zertifikat, das der Amazon-Root-CA-Zertifikatsdatei hinzugefügt wurde.

```
-----BEGIN CERTIFICATE-----
MIIEFTCCAv2gAwIQWgIVAMHSAzWG/5YVRYtRQOxXUTEpHuEmApzGCSqGSIb3DQEK
\nCwUAhuL9MQswCQwJVUzEPMAVUzEYMBYGA1UECgwP1hem9uLmNvbSBJbmMuMRww
... content of proxy CA certificate ...
+vHIRlt0e5JAm5\noTIZGoFbK82A0/nO7f/t5PSIDAim9V3Gc3pSXxCCAQoFYnui
GaPUlGk1gCE84a0X\n7Rp/lND/PuMZ/s8YjlkY2NmYmNjMCAXDTE5MTEyN2cM216
gJMIADggEPADf2/m45hzEXAMPLE=
-----END CERTIFICATE-----

-----BEGIN CERTIFICATE-----
MIIDQTCCAimgF6AwIBAgITBmyfz/5mjAo54vB4ikPmljZKyjANJmApzyMZFo6qBg
ADA5MQswCQYDVQQGEwJVUzEPMA0tMVT8QtPHRh8jrdkGA1UEChMGDV3QQDExBBKW
... content of root CA certificate ...
o/ufQJQWUCyziar1hem9uMRkwFwYVPSHCb2XV4cdFyQzR1KldZwgJcIQ6XUDgHaa
5MsI+yMRQ+hDaXJiobldXgjUka642M4UwtBV8oK2xJNDd2ZhwLnoQdeXeGADKkpy
rqXRfKoQnoZsG4q5WTP46EXAMPLE
-----END CERTIFICATE-----
```

Das NetworkProxy-Objekt

Geben Sie mithilfe des `networkProxy`-Objekts Informationen zum Netzwerk-Proxy an. Dieses Objekt enthält die folgenden Informationen:

`noProxyAddresses`

(Optional) Eine durch Kommas getrennte Liste von IP-Adressen oder Hostnamen, die vom Proxy ausgenommen sind.

`proxy`

Der Proxy, zu dem eine Verbindung hergestellt werden soll. Dieses Objekt enthält die folgenden Informationen:

`url`

Die URL des Proxyservers im Format`scheme://userinfo@host:port`.

- `scheme`— Das Schema, das `http` oder sein muss`https`.

  > **Important**
  >
  > Greengrass-Core-Geräte müssen Greengrass Nucleus v2.5.0 oder höher ausführen, um HTTPS-Proxys verwenden zu können.
  > Wenn Sie einen HTTPS-Proxy konfigurieren, müssen Sie das Proxy-Server-CA-Zertifikat zum Amazon-Root-CA-Zertifikat des Kerngeräts hinzufügen. Weitere Informationen finden Sie unter Ermöglichen Sie dem Kerngerät, einem HTTPS-Proxy zu vertrauen.

- `userinfo`— (Optional) Der Benutzername und das Passwort. Wenn Sie diese Informationen in der angeben`url`, ignoriert das Greengrass-Core-Gerät die Felder `username` und`password`.
- `host`— Der Hostname oder die IP-Adresse des Proxyservers.
- `port`— (Optional) Die Portnummer. Wenn Sie den Port nicht angeben, verwendet das Greengrass-Core-Gerät die folgenden Standardwerte:
  - `http`— 80
  - `https`— 443

`username`

(Optional) Der Benutzername, der den Proxyserver authentifiziert.

`password`

(Optional) Das Passwort, das den Proxyserver authentifiziert.

Verwenden Sie ein Gerätezertifikat, das von einer privaten Zertifizierungsstelle signiert wurde

Wenn Sie eine benutzerdefinierte private Zertifizierungsstelle (CA) verwenden, müssen Sie die Greengrass-Kerne auf setzen. **`greengrassDataPlaneEndpoint iotdata`** Sie können diese Option während der Bereitstellung oder Installation mithilfe des **`--init-config`** Installer-Arguments festlegen.

Sie können den Endpunkt der Greengrass-Datenebene anpassen, an den das Gerät eine Verbindung herstellt. Sie können diese Konfigurationsoption auf einstellen, **`iotdata`** um den Endpunkt der Greengrass-Datenebene auf denselben Endpunkt wie den IoT-Datenendpunkt festzulegen, den Sie mit dem **`iotDataEndpoint`** angeben können.

Konfigurieren Sie MQTT-Timeouts und Cache-Einstellungen

In der AWS IoT Greengrass Umgebung können Komponenten MQTT verwenden, um mit ihnen zu kommunizieren. AWS IoT Core Die AWS IoT Greengrass Core-Software verwaltet MQTT-Nachrichten für Komponenten. Wenn das Kerngerät die Verbindung zum verliert AWS Cloud, speichert die Software MQTT-Nachrichten im Cache, um es später erneut zu versuchen, wenn die Verbindung wiederhergestellt ist. Sie können Einstellungen wie Nachrichten-Timeouts und die Größe des Caches konfigurieren. Weitere Informationen finden Sie in den `mqtt` und den `mqtt.spooler` Konfigurationsparametern der Greengrass Nucleus-Komponente.

AWS IoT Core legt seinem MQTT-Nachrichtenbroker Dienstkontingente fest. Diese Kontingente gelten möglicherweise für Nachrichten, die Sie zwischen Kerngeräten und senden. AWS IoT Core Weitere Informationen finden Sie unter AWS IoT Core Message Broker Service-Kontingente in der Allgemeine AWS-Referenz.

Aktualisieren der AWS IoT Greengrass Core-Software (OTA)

Die -AWS IoT GreengrassCore-Software umfasst die Greengrass-Kernkomponente und andere optionale Komponenten, die Sie auf Ihren Geräten bereitstellen können, um over-the-air (OTA)-Updates der Software durchzuführen. Diese Funktion ist in die -AWS IoT GreengrassCore-Software integriert.

OTA-Updates machen Folgendes effizienter:

- Schließen von Sicherheitslücken.

- Behebung von Software-Stabilitätsproblemen.
- Bereitstellung neuer oder verbesserter Funktionen.

Themen

Voraussetzungen

Für die Bereitstellung von OTA-Updates der AWS IoT Greengrass Core-Software gelten die folgenden Anforderungen:

- Das Greengrass-Core-Gerät muss über eine Verbindung mit dem verfügenAWS Cloud, um die Bereitstellung zu empfangen.
- Das Greengrass-Core-Gerät muss korrekt konfiguriert und mit Zertifikaten und Schlüsseln für die Authentifizierung mit AWS IoT Core und bereitgestellt werdenAWS IoT Greengrass.
- Die AWS IoT Greengrass -Core-Software muss als Systemservice eingerichtet und ausgeführt werden. OTA-Aktualisierungen funktionieren nicht, wenn Sie den Kern aus der JAR-Datei ausführen`Greengrass.jar`. Weitere Informationen finden Sie unter Den Greengrass Nucleus als Systemdienst konfigurieren.

Überlegungen zu -Core-Geräten

Beachten Sie vor der Durchführung einer OTA-Aktualisierung die Auswirkungen auf die von Ihnen aktualisierten Core-Geräte und deren verbundene Client-Geräte:

- Der Greengrass-Kern wird heruntergefahren.
- Alle Komponenten, die auf dem Core-Gerät ausgeführt werden, werden ebenfalls heruntergefahren. Wenn diese Komponenten in lokale Ressourcen schreiben, verbleiben diese Ressourcen möglicherweise in einem falschen Zustand, es sei denn, sie werden ordnungsgemäß heruntergefahren. Komponenten können die prozessübergreifende Kommunikation verwenden, um die Kernkomponente anzuweisen, das Update aufschieben, bis sie die von ihnen verwendeten Ressourcen bereinigen.

- Während die Kernkomponente heruntergefahren wird, verliert das Core-Gerät seine Verbindungen mit den lokalen Geräten AWS Cloud und . Das Core-Gerät leitet keine Nachrichten von Client-Geräten weiter, während es herunterfährt.
- Langlebige Lambda-Funktionen, die als Komponenten ausgeführt werden, verlieren ihre dynamischen Statusinformationen und löschen alle ausstehenden Arbeit.

Aktualisierungsverhalten des Greengrass-Kerns

Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren AWS-Core-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.

Wenn sich die Version der Greengrass-Kernkomponente ändert, startet die AWS IoT Greengrass-Core-Software, die den -Kern und alle anderen Komponenten auf Ihrem Gerät enthält, neu, um die Änderungen anzuwenden. Aufgrund der Auswirkungen auf Core-Geräte bei der Aktualisierung der Kernkomponente möchten Sie möglicherweise steuern, wann eine neue Kern-Patch-Version auf Ihren Geräten bereitgestellt wird. Dazu müssen Sie die Greengrass-Kernkomponente direkt in Ihre Bereitstellung aufnehmen. Wenn Sie eine Komponente direkt einschließen, fügen Sie eine bestimmte Version dieser Komponente in Ihre Bereitstellungskonfiguration ein und verlassen sich nicht auf Komponentenabhängigkeiten, um diese Komponente auf Ihren Geräten bereitzustellen. Weitere Informationen zum Definieren von Abhängigkeiten in Ihren Komponentenrezepten finden Sie unter Rezeptformat.

Lesen Sie die folgende Tabelle, um das Aktualisierungsverhalten für die Greengrass-Kernkomponente basierend auf Ihren Aktionen und Bereitstellungskonfigurationen zu verstehen.

Aktion	Bereitstellungskonfiguration	Verhalten bei der Aktualisierung von Elementen
Fügen Sie einer Objektgruppe, auf die eine vorhandene Bereitstellung abzielt, neue Geräte hinzu, ohne die Bereitstellung zu überarbeiten.	Die Bereitstellung enthält nicht direkt den Greengrass-Kern. Die Bereitstellung enthält direkt mindestens eine	installiert auf neuen Geräten die neueste Patch-Version des Kerns, die alle Anforderungen an die Komponentenabhängigkeit erfüllt.

Aktion	Bereitstellungskonfiguration	Verhalten bei der Aktualisierung von Elementen
	AWSvon bereitgestellte Komponente oder eine benutzerdefinierte Komponente, die von einer von bereitgestellten Komponente oder vom GreengrassAWS-Kernus abhängt.	Auf vorhandenen Geräten aktualisiert die installierte Version des Kerns nicht.
Fügen Sie einer Objektgruppe, auf die eine vorhandene Bereitstellung abzielt, neue Geräte hinzu, ohne die Bereitstellung zu überarbeiten.	Die Bereitstellung enthält direkt eine bestimmte Version des Greengrass-Kerns.	installiert auf neuen Geräten die angegebene Kernversion. Auf vorhandenen Geräten aktualisiert die installierte Version des Kerns nicht.
Erstellen Sie eine neue Bereitstellung oder überarbeiten Sie eine vorhandene Bereitstellung.	Die Bereitstellung enthält nicht direkt den Greengrass-Kern. Die Bereitstellung enthält direkt mindestens eine AWSvon bereitgestellte Komponente oder eine benutzerdefinierte Komponente, die von einer von bereitgestellten Komponente oder vom GreengrassAWS-Kernus abhängt.	installiert auf allen Zielgeräten die neueste Patch-Version des Kerns, die alle Anforderungen an die Komponentenabhängigkeit erfüllt, einschließlich aller neuen Geräte, die Sie der Zielobjektgruppe hinzufügen.

Aktion	Bereitstellungskonfiguration	Verhalten bei der Aktualisierung von Elementen
Erstellen Sie eine neue Bereitstellung oder überarbeiten Sie eine vorhandene Bereitstellung.	Die Bereitstellung enthält direkt eine bestimmte Version des Greengrass-Kerns.	installiert auf allen Zielgeräten die angegebene Kernversion, einschließlich aller neuen Geräte, die Sie der Zielobjektgruppe hinzufügen.

Ausführen einer OTA-Aktualisierung

Um ein OTA-Update durchzuführen, erstellen Sie eine Bereitstellung, die die -Kernkomponente und die zu installierende Version enthält.

Deinstallieren der AWS IoT Greengrass -Core-Software

Sie können die AWS IoT Greengrass -Core-Software deinstallieren, um sie von einem Gerät zu entfernen, das Sie nicht als Greengrass-Core-Gerät verwenden möchten. Sie können diese Schritte auch verwenden, um eine fehlgeschlagene Installation zu bereinigen.

So deinstallieren Sie die AWS IoT Greengrass Core-Software

1. Wenn Sie die Software als Systemservice ausführen, müssen Sie den Service anhalten, deaktivieren und entfernen. Führen Sie die folgenden Befehle entsprechend Ihrem Betriebssystem aus.

   Linux

   1. Stoppen Sie den Service .

      ```
      sudo systemctl stop greengrass.service
      ```

   2. Deaktivieren Sie den Service.

      ```
      sudo systemctl disable greengrass.service
      ```

   3. Entfernen Sie den Service.

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

4. Überprüfen Sie, ob der Service gelöscht wurde.

```
sudo systemctl daemon-reload && sudo systemctl reset-failed
```

Windows (Command Prompt)

> **Note**
>
> Sie müssen die Eingabeaufforderung als Administrator ausführen, um diese Befehle auszuführen.

1. Stoppen Sie den Service .

```
sc stop "greengrass"
```

2. Deaktivieren Sie den Service.

```
sc config "greengrass" start=disabled
```

3. Entfernen Sie den Service.

```
sc delete "greengrass"
```

4. Starten Sie das Gerät neu.

Windows (PowerShell)

> **Note**
>
> Sie müssen PowerShell als Administrator ausführen, um diese Befehle auszuführen.

1. Stoppen Sie den Service .

```
Stop-Service -Name "greengrass"
```

2. Deaktivieren Sie den Service.

```
Set-Service -Name "greengrass" -Status stopped -StartupType disabled
```

3. Entfernen Sie den Service.

   - Für PowerShell 6.0 und höher:

```
Remove-Service -Name "greengrass" -Confirm:$false -Verbose
```

   - Für PowerShell Versionen vor 6.0:

```
Get-Item HKLM:\SYSTEM\CurrentControlSet\Services\greengrass | Remove-Item
 -Force -Verbose
```

4. Starten Sie das Gerät neu.

2. Entfernen Sie den Stammordner vom Gerät. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum Stammordner.

   Linux

```
sudo rm -rf /greengrass/v2
```

   Windows (Command Prompt)

```
rmdir /s /q C:\greengrass\v2
```

   Windows (PowerShell)

```
cmd.exe /c "rmdir /s /q C:\greengrass\v2"
```

3. Löschen Sie das Core-Gerät aus dem AWS IoT Greengrass Service. Dieser Schritt entfernt die Statusinformationen des Core-Geräts aus dem AWS Cloud. Stellen Sie sicher, dass Sie diesen Schritt ausführen, wenn Sie die AWS IoT Greengrass Core-Software auf einem Core-Gerät mit demselben Namen neu installieren möchten.

   - Gehen Sie wie folgt vor, um ein Core-Gerät aus der AWS IoT GreengrassKonsole zu löschen:

a. Navigieren Sie zur AWS IoT Greengrass-Konsole.
b. Wählen Sie Core-Geräte aus.
c. Wählen Sie das zu löschende Core-Gerät aus.
d. Wählen Sie Löschen aus.
e. Wählen Sie im Bestätigungsmodal Löschen aus.

- Um ein Core-Gerät mit der zu löschenAWS Command Line Interface, verwenden Sie die -DeleteCoreDeviceOperation. Führen Sie den folgenden Befehl aus und ersetzen Sie *MyGreengrassCore* durch den Namen des Core-Geräts.

```
aws greengrassv2 delete-core-device --core-device-thing-name MyGreengrassCore
```

AWS IoT Greengrass V2-Anleitungen

Sie können die folgenden Tutorials abschließen, um mehr über AWS IoT Greengrass V2 und seine Funktionen zu erfahren.

Themen

Tutorial: Entwickeln einer Greengrass-Komponente, die Komponentenaktualisierungen verzögert

Sie können dieses Tutorial abschließen, um eine Komponente zu entwickeln, die over-the-air Bereitstellungsaktualisierungen verzögert. Wenn Sie Updates auf Ihren Geräten bereitstellen, sollten Sie Updates basierend auf den folgenden Bedingungen verzögern:

- Das Gerät hat einen niedrigen Batteriestand.
- Das Gerät führt einen Prozess oder Auftrag aus, der nicht unterbrochen werden kann.
- Das Gerät verfügt über eine begrenzte oder teure Internetverbindung.

> **Note**
>
> Eine Komponente ist ein Softwaremodul, das auf -AWS IoT GreengrassCore-Geräten ausgeführt wird. Mit Komponenten können Sie komplexe Anwendungen als diskrete Bausteine erstellen und verwalten, die Sie von einem Greengrass-Kerngerät zu einem anderen wiederverwenden können.

In diesem Tutorial führen Sie folgende Aufgaben aus:

1. Installieren Sie die Greengrass Development Kit CLI (GDK CLI) auf Ihrem Entwicklungscomputer. Die GDK-CLI bietet Funktionen, mit denen Sie benutzerdefinierte Greengrass-Komponenten entwickeln können.
2. Entwickeln Sie eine Hello-World-Komponente, die Komponentenaktualisierungen verzögert, wenn der Batteriestand des Core-Geräts unter einem Schwellenwert liegt. Diese Komponente abonniert das Aktualisieren von Benachrichtigungen mithilfe des SubscribeToComponentUpdates IPC-Vorgangs. Wenn es die Benachrichtigung erhält, prüft es, ob der Batteriestand unter einem anpassbaren Schwellenwert liegt. Wenn der Batteriestand unter dem Schwellenwert liegt, verschiebt er die Aktualisierung mithilfe des DeferComponentUpdate IPC-Vorgangs um 30 Sekunden. Sie entwickeln diese Komponente auf Ihrem Entwicklungscomputer mithilfe der GDK-CLI.

   > **Note**
   >
   > Diese Komponente liest den Batteriestand aus einer Datei, die Sie auf dem Core-Gerät erstellen, um eine echte Batterie zu imitieren, sodass Sie dieses Tutorial auf einem Core-Gerät ohne Batterie abschließen können.

3. Veröffentlichen Sie diese Komponente im AWS IoT Greengrass Service.
4. Stellen Sie diese Komponente vom AWS Cloud auf einem Greengrass-Kerngerät bereit, um sie zu testen. Anschließend ändern Sie den virtuellen Batteriestand auf dem Core-Gerät und erstellen zusätzliche Bereitstellungen, um zu sehen, wie das Core-Gerät Updates aufschiebt, wenn der Batteriestand niedrig ist.

Sie können damit rechnen, 20–30 Minuten für dieses Tutorial zu verbringen.

Voraussetzungen

Zum Durcharbeiten dieses Tutorials ist Folgendes erforderlich:

- Ein(e) AWS-Konto. Falls Sie noch keines haben, beachten Sie die Informationen unter Richten Sie eine ein AWS-Konto.
- Ein AWS Identity and Access Management (IAM)-Benutzer mit Administratorberechtigungen.
- Ein Greengrass-Core-Gerät mit einer Internetverbindung. Weitere Informationen zum Einrichten eines Core-Geräts finden Sie unter Einrichtung von AWS IoT Greengrass Kerngeräten.

- Python 3.6 oder höher wurde für alle Benutzer auf dem Core-Gerät installiert und der PATH Umgebungsvariablen hinzugefügt. Unter Windows muss auch der Python Launcher für Windows für alle Benutzer installiert sein.

  > **Important**
  >
  > Unter Windows installiert Python standardmäßig nicht für alle Benutzer. Wenn Sie Python installieren, müssen Sie die Installation anpassen, um sie für die AWS IoT Greengrass Core-Software zum Ausführen von Python-Skripts zu konfigurieren. Wenn Sie beispielsweise das grafische Python-Installationsprogramm verwenden, gehen Sie wie folgt vor:
  >
  > 1. Wählen Sie Launcher installieren für alle Benutzer aus (empfohlen).
  > 2. Wählen Sie Customize installation.
  > 3. Wählen Sie Next.
  > 4. Wählen Sie Install for all users.
  > 5. Wählen Sie Add Python to environment variables.
  > 6. Wählen Sie Installieren aus.
  >
  > Weitere Informationen finden Sie unter Verwenden von Python unter Windows in der Python-3-Dokumentation.

- Ein Windows-, macOS- oder Unix-ähnlicher Entwicklungscomputer mit einer Internetverbindung.
  - Python 3.6 oder höher ist auf Ihrem Entwicklungscomputer installiert.
  - Git ist auf Ihrem Entwicklungscomputer installiert.
  - AWS Command Line Interface (AWS CLI) installiert und mit Anmeldeinformationen auf Ihrem Entwicklungscomputer konfiguriert. Weitere Informationen finden Sie unter Installieren, Aktualisieren und Deinstallieren der AWS CLI und Konfigurieren der AWS CLI im AWS Command Line Interface -Benutzerhandbuch.

    > **Note**
    >
    > Wenn Sie einen Raspberry Pi oder ein anderes 32-Bit-ARM-Gerät verwenden, installieren Sie AWS CLI V1. AWS CLI V2 ist für 32-Bit-ARM-Geräte nicht verfügbar.

Weitere Informationen finden Sie unter Installieren, Aktualisieren und Deinstallieren der AWS CLI Version 1.

Schritt 1: Installieren der Greengrass Development Kit CLI

Die Greengrass Development Kit CLI (GDK CLI) bietet Funktionen, mit denen Sie benutzerdefinierte Greengrass-Komponenten entwickeln können. Sie können die GDK-CLI verwenden, um benutzerdefinierte Komponenten zu erstellen, zu erstellen und zu veröffentlichen.

Wenn Sie die GDK-CLI nicht auf Ihrem Entwicklungscomputer installiert haben, führen Sie die folgenden Schritte aus, um sie zu installieren.

So installieren Sie die neueste Version der GDK-CLI

1. Führen Sie auf Ihrem Entwicklungscomputer den folgenden Befehl aus, um die neueste Version der GDK-CLI aus seinem GitHub Repository zu installieren.

   ```
   python3 -m pip install -U git+https://github.com/aws-greengrass/aws-greengrass-gdk-cli.git@v1.6.2
   ```

2. Führen Sie den folgenden Befehl aus, um zu überprüfen, ob die GDK-CLI erfolgreich installiert wurde.

   ```
   gdk --help
   ```

   Wenn der `gdk` Befehl nicht gefunden wird, fügen Sie seinen Ordner zu PATH hinzu.

   - Fügen Sie auf Linux-Geräten `/home/MyUser/.local/bin` zu PATH hinzu und ersetzen Sie durch *`MyUser`* den Namen Ihres Benutzers.
   - Fügen Sie auf Windows-Geräten *`PythonPath`*`\\Scripts` zu PATH hinzu und ersetzen Sie durch *`PythonPath`* den Pfad zum Python-Ordner auf Ihrem Gerät.

Schritt 2: Entwickeln einer Komponente, die Updates verzögert

In diesem Abschnitt entwickeln Sie eine Hello-World-Komponente in Python, die Komponentenaktualisierungen verzögert, wenn der Batteriestand des Core-Geräts unter einem Schwellenwert liegt, den Sie bei der Bereitstellung der Komponente konfigurieren. In dieser

Komponente verwenden Sie die IPC-Schnittstelle (Interprocess Communication) in AWS IoT Device SDK v2 für Python. Sie verwenden den SubscribeToComponentUpdates IPC-Vorgang, um Benachrichtigungen zu erhalten, wenn das Core-Gerät eine Bereitstellung erhält. Anschließend verwenden Sie die DeferComponentUpdate IPC-Operation, um das Update basierend auf dem Batteriestand des Geräts zu verschieben oder zu bestätigen.

So entwickeln Sie eine Hello-World-Komponente, die Updates verzögert

1. Erstellen Sie auf Ihrem Entwicklungscomputer einen Ordner für den Komponentenquellcode.

   ```
   mkdir com.example.BatteryAwareHelloWorld
   cd com.example.BatteryAwareHelloWorld
   ```

2. Verwenden Sie einen Texteditor, um eine Datei mit dem Namen zu erstellen`gdk-config.json`. Die GDK-CLI liest aus der GDK-CLI-Konfigurationsdatei mit dem Namen `gdk-config.json`, um Komponenten zu erstellen und zu veröffentlichen. Diese Konfigurationsdatei ist im Stammverzeichnis des Komponentenordners vorhanden.

   Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen der Datei zu verwenden.

   ```
   nano gdk-config.json
   ```

   Kopieren Sie den folgenden JSON-Code in die Datei .

   - Ersetzen Sie *`Amazon`* durch Ihren Namen.
   - Ersetzen Sie *`us-west-2`* durch die AWS-Region, in der Ihr Core-Gerät arbeitet. Die GDK-CLI veröffentlicht die Komponente in diesem AWS-Region.
   - Ersetzen Sie durch *`greengrass-component-artifacts`* das zu verwendende S3-Bucket-Präfix. Wenn Sie die GDK-CLI verwenden, um die Komponente zu veröffentlichen, lädt die GDK-CLI die Artefakte der Komponente in den S3-Bucket hochAWS-Region, dessen Name aus diesem Wert, dem und Ihrer AWS-Konto ID gebildet wird, und verwendet das folgende Format: *`bucketPrefix`*`-`*`region`*`-`*`accountId`*.

     Wenn Sie beispielsweise **`greengrass-component-artifacts`** und angeben **`us-west-2`**und Ihre AWS-Konto ID lautet**`123456789012`**, verwendet die GDK-CLI den S3-Bucket mit dem Namen `greengrass-component-artifacts-us-west-2-123456789012`.

```
{
  "component": {
    "com.example.BatteryAwareHelloWorld": {
      "author": "Amazon",
      "version": "NEXT_PATCH",
      "build": {
        "build_system" : "zip"
      },
      "publish": {
        "region": "us-west-2",
        "bucket": "greengrass-component-artifacts"
      }
    }
  },
  "gdk_version": "1.0.0"
}
```

Die Konfigurationsdatei gibt Folgendes an:

- Die zu verwendende Version, wenn die GDK-CLI die Greengrass-Komponente im AWS IoT Greengrass Cloud-Service veröffentlicht. `NEXT_PATCH` gibt an, die nächste Patch-Version nach der neuesten im AWS IoT Greengrass Cloud-Service verfügbaren Version auszuwählen. Wenn die Komponente noch keine Version im AWS IoT Greengrass Cloud-Service hat, verwendet die GDK-CLI `1.0.0`.
- Das Build-System für die Komponente. Wenn Sie das `zip` Build-System verwenden, packt die GDK-CLI die Quelle der Komponente in eine ZIP-Datei, die zum einzigen Artefakt der Komponente wird.
- Die AWS-Region, in der die GDK-CLI die Greengrass-Komponente veröffentlicht.
- Das Präfix für den S3-Bucket, in den die GDK-CLI die Artefakte der Komponente hochlädt.

3. Verwenden Sie einen Texteditor, um den Komponentenquellcode in einer Datei mit dem Namen zu erstellen`main.py`.

   Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen der Datei zu verwenden.

   ```
   nano main.py
   ```

Kopieren Sie den folgenden Python-Code in die Datei .

```
import json
import os
import sys
import time
import traceback

from pathlib import Path

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2

HELLO_WORLD_PRINT_INTERVAL = 15  # Seconds
DEFER_COMPONENT_UPDATE_INTERVAL = 30 * 1000  # Milliseconds


class BatteryAwareHelloWorldPrinter():
    def __init__(self, ipc_client: GreengrassCoreIPCClientV2, battery_file_path:
 Path, battery_threshold: float):
        self.battery_file_path = battery_file_path
        self.battery_threshold = battery_threshold
        self.ipc_client = ipc_client
        self.subscription_operation = None

    def on_component_update_event(self, event):
        try:
            if event.pre_update_event is not None:
                if self.is_battery_below_threshold():
                    self.defer_update(event.pre_update_event.deployment_id)
                    print('Deferred update for deployment %s' %
                          event.pre_update_event.deployment_id)
                else:
                    self.acknowledge_update(
                        event.pre_update_event.deployment_id)
                    print('Acknowledged update for deployment %s' %
                          event.pre_update_event.deployment_id)
            elif event.post_update_event is not None:
                print('Applied update for deployment')
        except:
            traceback.print_exc()

    def subscribe_to_component_updates(self):
        if self.subscription_operation == None:
```

```
            # SubscribeToComponentUpdates returns a tuple with the response and the
 operation.
            _, self.subscription_operation =
 self.ipc_client.subscribe_to_component_updates(
                on_stream_event=self.on_component_update_event)

    def close_subscription(self):
        if self.subscription_operation is not None:
            self.subscription_operation.close()
            self.subscription_operation = None

    def defer_update(self, deployment_id):
        self.ipc_client.defer_component_update(
            deployment_id=deployment_id,
 recheck_after_ms=DEFER_COMPONENT_UPDATE_INTERVAL)

    def acknowledge_update(self, deployment_id):
        # Specify recheck_after_ms=0 to acknowledge a component update.
        self.ipc_client.defer_component_update(
            deployment_id=deployment_id, recheck_after_ms=0)

    def is_battery_below_threshold(self):
        return self.get_battery_level() < self.battery_threshold

    def get_battery_level(self):
        # Read the battery level from the virtual battery level file.
        with self.battery_file_path.open('r') as f:
            data = json.load(f)
            return float(data['battery_level'])

    def print_message(self):
        message = 'Hello, World!'
        if self.is_battery_below_threshold():
            message += ' Battery level (%d) is below threshold (%d), so the
 component will defer updates' % (
                self.get_battery_level(), self.battery_threshold)
        else:
            message += ' Battery level (%d) is above threshold (%d), so the
 component will acknowledge updates' % (
                self.get_battery_level(), self.battery_threshold)
        print(message)


def main():
```

```
    # Read the battery threshold and virtual battery file path from command-line
args.
    args = sys.argv[1:]
    battery_threshold = float(args[0])
    battery_file_path = Path(args[1])
    print('Reading battery level from %s and deferring updates when below %d' % (
        str(battery_file_path), battery_threshold))

    try:
        # Create an IPC client and a Hello World printer that defers component
updates.
        ipc_client = GreengrassCoreIPCClientV2()
        hello_world_printer = BatteryAwareHelloWorldPrinter(
            ipc_client, battery_file_path, battery_threshold)
        hello_world_printer.subscribe_to_component_updates()
        try:
            # Keep the main thread alive, or the process will exit.
            while True:
                hello_world_printer.print_message()
                time.sleep(HELLO_WORLD_PRINT_INTERVAL)
        except InterruptedError:
            print('Subscription interrupted')
        hello_world_printer.close_subscription()
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)


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

Diese Python-Anwendung führt Folgendes aus:

- Liest den Batteriestand des Core-Geräts aus einer Datei mit dem virtuellen Batteriestand, die Sie später auf dem Core-Gerät erstellen werden. Diese Datei mit virtuellem Batteriestand imitiert eine echte Batterie, sodass Sie dieses Tutorial auf Core-Geräten ohne Batterie abschließen können.
- Liest Befehlszeilenargumente für den Batterieschwellenwert und den Pfad zur Datei mit dem virtuellen Batteriestand. Das Komponentenrezept legt diese Befehlszeilenargumente auf der Grundlage von Konfigurationsparametern fest, sodass Sie diese Werte anpassen können, wenn Sie die Komponente bereitstellen.

- Verwendet den IPC-Client V2 in der AWS IoT Device SDK v2 für Python, um mit der AWS IoT Greengrass Core-Software zu kommunizieren. Im Vergleich zum ursprünglichen IPC-Client reduziert der IPC-Client V2 die Menge an Code, die Sie schreiben müssen, um IPC in benutzerdefinierten Komponenten zu verwenden.
- Abonniert das Aktualisieren von Benachrichtigungen mithilfe der SubscribeToComponentUpdates IPC-Operation. Die AWS IoT Greengrass -Core-Software sendet Benachrichtigungen vor und nach jeder Bereitstellung. Die Komponente ruft die folgende Funktion jedes Mal auf, wenn sie eine Benachrichtigung erhält. Wenn die Benachrichtigung für eine bevorstehende Bereitstellung gilt, prüft die Komponente, ob der Batteriestand unter einem Schwellenwert liegt. Wenn der Batteriestand unter dem Schwellenwert liegt, verschiebt die Komponente die Aktualisierung mithilfe des DeferComponentUpdate IPC-Vorgangs um 30 Sekunden. Andernfalls bestätigt die Komponente das Update, wenn der Batteriestand nicht unter dem Schwellenwert liegt, sodass das Update fortgesetzt werden kann.

```
def on_component_update_event(self, event):
    try:
        if event.pre_update_event is not None:
            if self.is_battery_below_threshold():
                self.defer_update(event.pre_update_event.deployment_id)
                print('Deferred update for deployment %s' %
                      event.pre_update_event.deployment_id)
            else:
                self.acknowledge_update(
                    event.pre_update_event.deployment_id)
                print('Acknowledged update for deployment %s' %
                      event.pre_update_event.deployment_id)
        elif event.post_update_event is not None:
            print('Applied update for deployment')
    except:
        traceback.print_exc()
```

> **Note**
>
> Die -AWS IoT GreengrassCore-Software sendet keine Aktualisierungsbenachrichtigungen für lokale Bereitstellungen, daher stellen Sie diese Komponente mithilfe des AWS IoT Greengrass Cloud-Services bereit, um sie zu testen.

4. Verwenden Sie einen Texteditor, um das Komponentenrezept in einer Datei mit dem Namen `recipe.json` oder zu erstellen`recipe.yaml`. Das Komponentenrezept definiert die Metadaten der Komponente, die Standardkonfigurationsparameter und die plattformspezifischen Lebenszyklusskripts.

   JSON

   Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen der Datei zu verwenden.

   ```
   nano recipe.json
   ```

   Kopieren Sie den folgenden JSON-Code in die Datei .

   ```
   {
     "RecipeFormatVersion": "2020-01-25",
     "ComponentName": "COMPONENT_NAME",
     "ComponentVersion": "COMPONENT_VERSION",
     "ComponentDescription": "This Hello World component defers updates when the
    battery level is below a threshold.",
     "ComponentPublisher": "COMPONENT_AUTHOR",
     "ComponentConfiguration": {
       "DefaultConfiguration": {
         "BatteryThreshold": 50,
         "LinuxBatteryFilePath": "/home/ggc_user/virtual_battery.json",
         "WindowsBatteryFilePath": "C:\\Users\\ggc_user\\virtual_battery.json"
       }
     },
     "Manifests": [
       {
         "Platform": {
           "os": "linux"
         },
         "Lifecycle": {
           "install": "python3 -m pip install --user awsiotsdk --upgrade",
           "run": "python3 -u {artifacts:decompressedPath}/
   com.example.BatteryAwareHelloWorld/main.py \"{configuration:/BatteryThreshold}\"
    \"{configuration:/LinuxBatteryFilePath}\""
         },
         "Artifacts": [
           {
   ```

```
          "Uri": "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/
com.example.BatteryAwareHelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk --upgrade",
        "run": "py -3 -u {artifacts:decompressedPath}/
com.example.BatteryAwareHelloWorld/main.py \"{configuration:/BatteryThreshold}\"
 \"{configuration:/WindowsBatteryFilePath}\""
      },
      "Artifacts": [
        {
          "Uri": "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/
com.example.BatteryAwareHelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ]
    }
  ]
}
```

YAML

Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen der Datei zu verwenden.

```
nano recipe.yaml
```

Kopieren Sie die folgende YAML in die Datei .

```
---
RecipeFormatVersion: "2020-01-25"
ComponentName: "COMPONENT_NAME"
ComponentVersion: "COMPONENT_VERSION"
ComponentDescription: "This Hello World component defers updates when the
 battery level is below a threshold."
```

```
ComponentPublisher: "COMPONENT_AUTHOR"
ComponentConfiguration:
  DefaultConfiguration:
    BatteryThreshold: 50
    LinuxBatteryFilePath: "/home/ggc_user/virtual_battery.json"
    WindowsBatteryFilePath: "C:\\Users\\ggc_user\\virtual_battery.json"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: python3 -m pip install --user awsiotsdk --upgrade
      run: python3 -u {artifacts:decompressedPath}/
com.example.BatteryAwareHelloWorld/main.py "{configuration:/BatteryThreshold}"
 "{configuration:/LinuxBatteryFilePath}"
    Artifacts:
      - Uri: "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/
com.example.BatteryAwareHelloWorld.zip"
        Unarchive: ZIP
  - Platform:
      os: windows
    Lifecycle:
      install: py -3 -m pip install --user awsiotsdk --upgrade
      run: py -3 -u {artifacts:decompressedPath}/
com.example.BatteryAwareHelloWorld/main.py "{configuration:/BatteryThreshold}"
 "{configuration:/WindowsBatteryFilePath}"
    Artifacts:
      - Uri: "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/
com.example.BatteryAwareHelloWorld.zip"
        Unarchive: ZIP
```

Dieses Rezept gibt Folgendes an:

- Standardkonfigurationsparameter für den Batterieschwellenwert, den virtuellen Batteriedateipfad auf Linux-Core-Geräten und den virtuellen Batteriedateipfad auf Windows-Core-Geräten.
- Ein `install` Lebenszyklus, der die neueste Version von AWS IoT Device SDK v2 für Python installiert.
- Ein `run` Lebenszyklus, der die Python-Anwendung in ausführt`main.py`.
- Platzhalter wie `COMPONENT_NAME` und `COMPONENT_VERSION`, wobei die GDK-CLI Informationen ersetzt, wenn sie das Komponentenrezept erstellt.

Weitere Informationen zu Komponentenrezepten finden Sie unter AWS IoT Greengrass Referenz zum Komponenten-Rezept.

Schritt 3: Veröffentlichen der Komponente im AWS IoT Greengrass Service

In diesem Abschnitt veröffentlichen Sie die Hello-World-Komponente im AWS IoT Greengrass Cloud-Service. Nachdem eine Komponente im AWS IoT Greengrass Cloud-Service verfügbar ist, können Sie sie auf -Core-Geräten bereitstellen. Sie verwenden die GDK-CLI, um die Komponente von Ihrem Entwicklungscomputer im AWS IoT Greengrass Cloud-Service zu veröffentlichen. Die GDK-CLI lädt das Rezept und die Artefakte der Komponente für Sie hoch.

So veröffentlichen Sie die Hello-World-Komponente im AWS IoT Greengrass Service

1. Führen Sie den folgenden Befehl aus, um die Komponente mit der GDK-CLI zu erstellen. Der Komponentenerstellungsbefehl erstellt ein Rezept und Artefakte basierend auf der GDK-CLI-Konfigurationsdatei. In diesem Prozess erstellt die GDK-CLI eine ZIP-Datei, die den Quellcode der Komponente enthält.

   ```
   gdk component build
   ```

   Sie sollten Meldungen ähnlich dem folgenden Beispiel sehen.

   ```
   [2022-04-28 11:20:16] INFO - Getting project configuration from gdk-config.json
   [2022-04-28 11:20:16] INFO - Found component recipe file 'recipe.yaml' in the
    project directory.
   [2022-04-28 11:20:16] INFO - Building the component
    'com.example.BatteryAwareHelloWorld' with the given project configuration.
   [2022-04-28 11:20:16] INFO - Using 'zip' build system to build the component.
   [2022-04-28 11:20:16] WARNING - This component is identified as using 'zip' build
    system. If this is incorrect, please exit and specify custom build command in the
    'gdk-config.json'.
   [2022-04-28 11:20:16] INFO - Zipping source code files of the component.
   [2022-04-28 11:20:16] INFO - Copying over the build artifacts to the greengrass
    component artifacts build folder.
   [2022-04-28 11:20:16] INFO - Updating artifact URIs in the recipe.
   [2022-04-28 11:20:16] INFO - Creating component recipe in 'C:\Users\finthomp
   \greengrassv2\com.example.BatteryAwareHelloWorld\greengrass-build\recipes'.
   ```

2. Führen Sie den folgenden Befehl aus, um die Komponente im AWS IoT Greengrass Cloud-Service zu veröffentlichen. Der Befehl zum Veröffentlichen der Komponente lädt das ZIP-Dateiartefakt der Komponente in einen S3-Bucket hoch. Anschließend wird der S3-URI der ZIP-Datei im Komponentenrezept aktualisiert und das Rezept wird in den AWS IoT Greengrass Service hochgeladen. In diesem Prozess überprüft die GDK-CLI, welche Version der Hello World-Komponente bereits im AWS IoT Greengrass Cloud-Service verfügbar ist, sodass sie die nächste Patch-Version nach dieser Version auswählen kann. Wenn die Komponente noch nicht vorhanden ist, verwendet die GDK-CLI Version `1.0.0`.

```
gdk component publish
```

Sie sollten Meldungen ähnlich dem folgenden Beispiel sehen. Die Ausgabe teilt Ihnen die Version der Komponente mit, die die GDK-CLI erstellt hat.

```
[2022-04-28 11:20:29] INFO - Getting project configuration from gdk-config.json
[2022-04-28 11:20:29] INFO - Found component recipe file 'recipe.yaml' in the
 project directory.
[2022-04-28 11:20:29] INFO - Found credentials in shared credentials file: ~/.aws/
credentials
[2022-04-28 11:20:30] INFO - No private version of the component
 'com.example.BatteryAwareHelloWorld' exist in the account. Using '1.0.0' as the
 next version to create.
[2022-04-28 11:20:30] INFO - Publishing the component
 'com.example.BatteryAwareHelloWorld' with the given project configuration.
[2022-04-28 11:20:30] INFO - Uploading the component built artifacts to s3 bucket.
[2022-04-28 11:20:30] INFO - Uploading component artifacts to S3
 bucket: greengrass-component-artifacts-us-west-2-123456789012. If this is your
 first time using this bucket, add the 's3:GetObject' permission to each core
 device's token exchange role to allow it to download the component artifacts. For
 more information, see https://docs.aws.amazon.com/greengrass/v2/developerguide/
device-service-role.html.
[2022-04-28 11:20:30] INFO - Not creating an artifacts bucket as it already exists.
[2022-04-28 11:20:30] INFO - Updating the component recipe
 com.example.BatteryAwareHelloWorld-1.0.0.
[2022-04-28 11:20:31] INFO - Creating a new greengrass component
 com.example.BatteryAwareHelloWorld-1.0.0
[2022-04-28 11:20:31] INFO - Created private version '1.0.0' of the component in
 the account.'com.example.BatteryAwareHelloWorld'.
```

3. Kopieren Sie den Namen des S3-Buckets aus der Ausgabe. Sie verwenden den Bucket-Namen später, damit das Core-Gerät Komponentenartefakte aus diesem Bucket herunterladen kann.

4. (Optional) Zeigen Sie die Komponente in der AWS IoT Greengrass Konsole an, um zu überprüfen, ob sie erfolgreich hochgeladen wurde. Gehen Sie wie folgt vor:

   a. Wählen Sie im Navigationsmenü der AWS IoT GreengrassKonsole Komponenten aus.

   b. Wählen Sie auf der Seite Komponenten die Registerkarte Meine Komponenten und dann auscom.example.BatteryAwareHelloWorld.

      Auf dieser Seite sehen Sie das Rezept der Komponente und andere Informationen über die Komponente.

5. Erlauben Sie dem Core-Gerät den Zugriff auf Komponentenartefakte im S3-Bucket.

   Jedes Core-Gerät verfügt über eine IAM-Rolle für das Core-Gerät, die es ihm ermöglicht, mit der AWS Cloud zu interagieren AWS IoT und Protokolle an diese zu senden. Diese Geräterolle erlaubt standardmäßig keinen Zugriff auf S3-Buckets. Daher müssen Sie eine Richtlinie erstellen und anfügen, die es dem Core-Gerät ermöglicht, Komponentenartefakte aus dem S3-Bucket abzurufen.

   Wenn die Rolle Ihres Geräts bereits den Zugriff auf den S3-Bucket zulässt, können Sie diesen Schritt überspringen. Erstellen Sie andernfalls eine IAM-Richtlinie, die den Zugriff zulässt, und fügen Sie sie wie folgt an die Rolle an:

   a. Wählen Sie im Navigationsmenü der IAM-Konsole Richtlinien und dann Richtlinie erstellen aus.

   b. Ersetzen Sie auf der Registerkarte JSON den Platzhalterinhalt durch die folgende Richtlinie. Ersetzen Sie *greengrass-component-artifacts-us-west-2-123456789012* durch den Namen des S3-Buckets, in den die GDK-CLI die Artefakte der Komponente hochgeladen hat.

      Wenn Sie beispielsweise **greengrass-component-artifacts** und **us-west-2** in der GDK-CLI-Konfigurationsdatei angegeben haben und Ihre AWS-Konto ID lautet**123456789012**, verwendet die GDK-CLI den S3-Bucket mit dem Namen `greengrass-component-artifacts-us-west-2-123456789012`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
```

```
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::greengrass-component-artifacts-us-
west-2-123456789012/*"
    }
  ]
}
```

c. Wählen Sie Weiter aus.

d. Geben Sie im Abschnitt Richtliniendetails für Name ein**`MyGreengrassV2ComponentArtifactPolicy`**.

e. Wählen Sie Richtlinie erstellen aus.

f. Wählen Sie im Navigationsmenü der IAM-Konsole Rolle und dann den Namen der Rolle für das Core-Gerät aus. Sie haben diesen Rollennamen bei der Installation der AWS IoT Greengrass Core-Software angegeben. Wenn Sie keinen Namen angegeben haben, ist der Standardwert `GreengrassV2TokenExchangeRole`.

g. Wählen Sie unter Berechtigungen die Option Berechtigungen hinzufügen und dann Richtlinien anfügen aus.

h. Aktivieren Sie auf der Seite Berechtigungen hinzufügen das Kontrollkästchen neben der von Ihnen erstellten `MyGreengrassV2ComponentArtifactPolicy` Richtlinie und wählen Sie dann Berechtigungen hinzufügen aus.

Schritt 4: Bereitstellen und Testen der Komponente auf einem Core-Gerät

In diesem Abschnitt stellen Sie die Komponente auf dem Core-Gerät bereit, um ihre Funktionalität zu testen. Auf dem Core-Gerät erstellen Sie die Datei mit dem virtuellen Batteriestand, um eine echte Batterie zu imitieren. Anschließend erstellen Sie zusätzliche Bereitstellungen und beobachten die Komponentenprotokolldateien auf dem Core-Gerät, um die Komponentenverzögerung zu sehen und Updates zu bestätigen.

So stellen Sie die Hello-World-Komponente bereit und testen sie, die Updates verzögert

1. Verwenden Sie einen Texteditor, um eine Datei mit virtuellem Batteriestand zu erstellen. Diese Datei imitiert eine echte Batterie.

   - Erstellen Sie auf Linux-Core-Geräten eine Datei mit dem Namen `/home/ggc_user/virtual_battery.json`. Führen Sie den Texteditor mit `-sudo`Berechtigungen aus.

- Erstellen Sie auf Windows-Core-Geräten eine Datei mit dem Namen `C:\Users\ggc_user\virtual_battery.json`. Führen Sie den Texteditor als Administrator aus.

Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen der Datei zu verwenden.

```
sudo nano /home/ggc_user/virtual_battery.json
```

Kopieren Sie den folgenden JSON-Code in die Datei .

```
{
  "battery_level": 50
}
```

2. Stellen Sie die Hello-World-Komponente auf dem Core-Gerät bereit. Gehen Sie wie folgt vor:

   a. Wählen Sie im Navigationsmenü der AWS IoT GreengrassKonsole Komponenten aus.

   b. Wählen Sie auf der Seite Komponenten die Registerkarte Meine Komponenten und dann auscom.example.BatteryAwareHelloWorld.

   c. Wählen Sie auf der com.example.BatteryAwareHelloWorld Seite Bereitstellen aus.

   d. Wählen Sie unter Zu Bereitstellung hinzufügen eine vorhandene Bereitstellung aus, die Sie überarbeiten möchten, oder wählen Sie aus, um eine neue Bereitstellung zu erstellen, und wählen Sie dann Weiter aus.

   e. Wenn Sie eine neue Bereitstellung erstellen möchten, wählen Sie das Ziel-Core-Gerät oder die Objektgruppe für die Bereitstellung aus. Wählen Sie auf der Seite Ziel angeben unter Bereitstellungsziel ein Core-Gerät oder eine Objektgruppe und dann Weiter aus.

   f. Überprüfen Sie auf der Seite Komponenten auswählen, ob die com.example.BatteryAwareHelloWorld Komponente ausgewählt ist, und wählen Sie Weiter aus.

   g. Wählen Sie auf der Seite Komponenten konfigurieren die Option aus com.example.BatteryAwareHelloWorldund gehen Sie dann wie folgt vor:

      i. Wählen Sie Komponente konfigurieren aus.

      ii. Geben Sie im Feld Modal konfigurieren com.example.BatteryAwareHelloWorld unter Konfigurationsaktualisierung unter Konfiguration zum Zusammenführen von das folgende Konfigurationsupdate ein.

```
{
  "BatteryThreshold": 70
}
```

iii. Wählen Sie Bestätigen, um das Modal zu schließen, und wählen Sie dann Weiter aus.

h. Vergewissern Sie sich auf der Seite Erweiterte Einstellungen bestätigen im Abschnitt Bereitstellungsrichtlinien unter Komponentenaktualisierungsrichtlinie, dass Komponenten benachrichtigen ausgewählt ist. Benachrichtigen von Komponenten ist standardmäßig ausgewählt, wenn Sie eine neue Bereitstellung erstellen.

i. Wählen Sie auf der Seite Review (Prüfen) die Option Deploy (Bereitstellen) aus.

Die Bereitstellung kann bis zu einer Minute dauern.

3. Die AWS IoT Greengrass Core-Software speichert stdout aus Komponentenprozessen in Protokolldateien im `logs` Ordner . Führen Sie den folgenden Befehl aus, um zu überprüfen, ob die Hello-World-Komponente ausgeführt wird und Statusmeldungen druckt.

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/com.example.BatteryAwareHelloWorld.log
```

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\logs\com.example.BatteryAwareHelloWorld.log
```

PowerShell

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

Sie sollten Meldungen ähnlich dem folgenden Beispiel sehen.

```
Hello, World! Battery level (50) is below threshold (70), so the component will
 defer updates.
```

> **Note**
>
> Wenn die Datei nicht vorhanden ist, ist die Bereitstellung möglicherweise noch nicht abgeschlossen. Wenn die Datei nicht innerhalb von 30 Sekunden existiert, ist die Bereitstellung wahrscheinlich fehlgeschlagen. Dies kann beispielsweise der Fall sein, wenn das Core-Gerät nicht über die Berechtigung verfügt, die Artefakte der Komponente aus dem S3-Bucket herunterzuladen. Führen Sie den folgenden Befehl aus, um die Protokolldatei der -AWS IoT GreengrassCore-Software anzuzeigen. Diese Datei enthält Protokolle aus dem Bereitstellungsservice des Greengrass-Core-Geräts.
>
> Linux or Unix
>
> ```
> sudo tail -f /greengrass/v2/logs/greengrass.log
> ```
>
> Windows Command Prompt (CMD)
>
> ```
> type C:\greengrass\v2\logs\greengrass.log
> ```
>
> Der `type` Befehl schreibt den Inhalt der Datei in das Terminal. Führen Sie diesen Befehl mehrmals aus, um Änderungen in der Datei zu beobachten.
>
> PowerShell
>
> ```
> gc C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
> ```

4. Erstellen Sie eine neue Bereitstellung auf dem Core-Gerät, um zu überprüfen, ob die Komponente das Update aufschiebt. Gehen Sie wie folgt vor:
   a. Wählen Sie im Navigationsmenü der AWS IoT Greengrass Konsole Bereitstellungen aus.
   b. Wählen Sie die Bereitstellung aus, die Sie zuvor erstellt oder überarbeitet haben.
   c. Wählen Sie auf der Bereitstellungsseite Überarbeiten aus.
   d. Wählen Sie im Modal Bereitstellung überarbeiten die Option Bereitstellung überarbeiten aus.
   e. Wählen Sie bei jedem Schritt Weiter und dann Bereitstellen aus.
5. Führen Sie den folgenden Befehl aus, um die Protokolle der Komponente erneut anzuzeigen und zu überprüfen, ob die Aktualisierung verzögert wird.

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/com.example.BatteryAwareHelloWorld.log
```

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\logs\com.example.BatteryAwareHelloWorld.log
```

PowerShell

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

Sie sollten Meldungen ähnlich dem folgenden Beispiel sehen. Die Komponente verschiebt die Aktualisierung um 30 Sekunden, sodass die Komponente diese Meldung wiederholt ausgibt.

```
Deferred update for deployment 50722a95-a05f-4e2a-9414-da80103269aa.
```

6. Verwenden Sie einen Texteditor, um die Datei mit dem virtuellen Batteriestand zu bearbeiten und den Batteriestand auf einen Wert über dem Schwellenwert zu ändern, damit die Bereitstellung fortgesetzt werden kann.

   - Bearbeiten Sie auf Linux-Core-Geräten die Datei mit dem Namen `/home/ggc_user/virtual_battery.json`. Führen Sie den Texteditor mit `-sudo`Berechtigungen aus.
   - Bearbeiten Sie auf Windows-Core-Geräten die Datei mit dem Namen `C:\Users\ggc_user\virtual_battery.json`. Führen Sie den Texteditor als Administrator aus.

   Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen der Datei zu verwenden.

   ```
   sudo nano /home/ggc_user/virtual_battery.json
   ```

   Ändern Sie den Batteriestand auf `80`.

   ```
   {
     "battery_level": 80
   }
   ```

7. Führen Sie den folgenden Befehl aus, um die Protokolle der Komponente erneut anzuzeigen und zu überprüfen, ob die Aktualisierung bestätigt wurde.

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/com.example.BatteryAwareHelloWorld.log
   ```

   Windows Command Prompt (CMD)

   ```
   type C:\greengrass\v2\logs\com.example.BatteryAwareHelloWorld.log
   ```

   PowerShell

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

   Sie sollten Meldungen ähnlich den folgenden Beispielen sehen.

   ```
   Hello, World! Battery level (80) is above threshold (70), so the component will
    acknowledge updates.
   Acknowledged update for deployment f9499eb2-4a40-40a7-86c1-c89887d859f1.
   ```

Sie haben dieses Tutorial abgeschlossen. Die Hello-World-Komponente verschiebt oder bestätigt Updates basierend auf dem Batteriestand des Core-Geräts. Weitere Informationen zu den Themen, die in diesem Tutorial behandelt werden, finden Sie unter:

- Entwickeln von AWS IoT Greengrass Komponenten
- Bereitstellen von AWS IoT Greengrass Komponenten auf Geräten
- Verwenden Sie den AWS IoT Device SDK , um mit dem Greengrass-Kern und anderen Komponenten zu kommunizieren und AWS IoT Core
- AWS IoT Greengrass Befehlszeilenschnittstelle des Development Kit

Tutorial: Interagieren mit lokalen IoT-Geräten über MQTT

Sie können dieses Tutorial abschließen, um ein Core-Gerät für die Interaktion mit lokalen IoT-Geräten, den so genannten Client-Geräten, zu konfigurieren, die über MQTT eine Verbindung zum Core-Gerät herstellen. In diesem Tutorial konfigurieren Sie AWS IoT Objekte für die Verwendung von

Cloud Discovery, um eine Verbindung zum Core-Gerät als Client-Geräte herzustellen. Wenn Sie die Cloud-Erkennung konfigurieren, kann ein Client-Gerät eine Anforderung an den AWS IoT Greengrass Cloud-Service senden, um -Core-Geräte zu erkennen. Die Antwort von AWS IoT Greengrass enthält Konnektivitätsinformationen und Zertifikate für die Core-Geräte, für deren Erkennung Sie das Client-Gerät konfigurieren. Anschließend kann das Client-Gerät diese Informationen verwenden, um eine Verbindung zu einem verfügbaren Core-Gerät herzustellen, auf dem es über MQTT kommunizieren kann.

In diesem Tutorial führen Sie folgende Aufgaben aus:

1. Überprüfen und aktualisieren Sie bei Bedarf die Berechtigungen des Core-Geräts.
2. Ordnen Sie dem Core-Gerät Client-Geräte zu, damit sie das Core-Gerät mithilfe der Cloud-Erkennung erkennen können.
3. Stellen Sie Greengrass-Komponenten auf dem Core-Gerät bereit, um Client-Geräteunterstützung zu ermöglichen.
4. Verbinden Sie Client-Geräte mit dem Core-Gerät und testen Sie die Kommunikation mit dem AWS IoT Core Cloud-Service.
5. Entwickeln Sie eine benutzerdefinierte Greengrass-Komponente, die mit den Client-Geräten kommuniziert.
6. Entwickeln Sie eine benutzerdefinierte Komponente, die mit den AWS IoT Geräteschatten der Client-Geräte interagiert.

In diesem Tutorial werden ein einzelnes Core-Gerät und ein einzelnes Client-Gerät verwendet. Sie können auch dem Tutorial folgen, um mehrere Client-Geräte zu verbinden und zu testen.

Sie können damit rechnen, 30–60 Minuten für dieses Tutorial zu verbringen.

Voraussetzungen

Zum Durcharbeiten dieses Tutorials ist Folgendes erforderlich:

- Ein(e) AWS-Konto. Falls Sie noch keines haben, beachten Sie die Informationen unter Richten Sie eine ein AWS-Konto.
- Ein AWS Identity and Access Management (IAM)-Benutzer mit Administratorberechtigungen.
- Ein Greengrass-Core-Gerät. Weitere Informationen zum Einrichten eines Core-Geräts finden Sie unter Einrichtung von AWS IoT Greengrass Kerngeräten.

- Das Core-Gerät muss Greengrass-Kern v2.6.0 oder höher ausführen. Diese Version bietet Unterstützung für Platzhalter bei der lokalen Veröffentlichungs-/Abonnementkommunikation und Unterstützung für Client-Geräteschatten.

  > **Note**
  >
  > Die Unterstützung von Client-Geräten erfordert Greengrass-Kern v2.2.0 oder höher. In diesem Tutorial werden jedoch neuere Funktionen untersucht, z. B. die Unterstützung für MQTT-Platzhalter bei lokaler Veröffentlichung/Abonnement und die Unterstützung für Client-Geräteschatten. Für diese Funktionen ist Greengrass-Kern v2.6.0 oder höher erforderlich.

- Das Core-Gerät muss sich im selben Netzwerk befinden wie die Client-Geräte, um eine Verbindung herzustellen.
- (Optional) Um die Module abzuschließen, in denen Sie benutzerdefinierte Greengrass-Komponenten entwickeln, muss das Core-Gerät die Greengrass-CLI ausführen. Weitere Informationen finden Sie unter Installieren Sie die Greengrass-CLI.

- Ein AWS IoT Objekt, das in diesem Tutorial als Client-Gerät verbunden werden soll. Weitere Informationen finden Sie unter Erstellen von -AWS IoTRessourcen im AWS IoT Core - Entwicklerhandbuch.
  - Die AWS IoT Richtlinie des Client-Geräts muss die -`greengrass:Discover`Berechtigung zulassen. Weitere Informationen finden Sie unter Minimale AWS IoT Richtlinie für Client-Geräte.
  - Das Client-Gerät muss sich im selben Netzwerk wie das Core-Gerät befinden.
  - Das Client-Gerät muss Python 3 ausführen.
  - Auf dem Client-Gerät muss Git ausgeführt werden.

Schritt 1: Überprüfen und Aktualisieren der AWS IoT Richtlinie für Kerngeräte

Um Client-Geräte zu unterstützen, muss die AWS IoT Richtlinie eines Core-Geräts die folgenden Berechtigungen zulassen:

- `greengrass:PutCertificateAuthorities`
- `greengrass:VerifyClientDeviceIdentity`
- `greengrass:VerifyClientDeviceIoTCertificateAssociation`

- `greengrass:GetConnectivityInfo`
- `greengrass:UpdateConnectivityInfo` – (Optional) Diese Berechtigung ist erforderlich, um die IP-Detektorkomponente zu verwenden, die die Netzwerkkonnektivitätsinformationen des Core-Geräts an den AWS IoT Greengrass Cloud-Service meldet.

Weitere Informationen zu diesen Berechtigungen und AWS IoT Richtlinien für -Core-Geräte finden Sie unter AWS IoT-Richtlinien für Operationen auf Datenebene und Minimale AWS IoT Richtlinie zur Unterstützung von Client-Geräten.

In diesem Abschnitt überprüfen Sie die AWS IoT Richtlinien für Ihr Core-Gerät und fügen alle erforderlichen Berechtigungen hinzu, die fehlen. Wenn Sie das AWS IoT Greengrass -Core-Softwareinstallationsprogramm zur Bereitstellung von Ressourcen verwendet haben, verfügt Ihr -Core-Gerät über eine -AWS IoTRichtlinie, die den Zugriff auf alle -AWS IoT GreengrassAktionen (`greengrass:*`) ermöglicht. In diesem Fall müssen Sie die AWS IoT Richtlinie nur aktualisieren, wenn Sie die Shadow-Manager-Komponente so konfigurieren möchten, dass Geräteschatten mit synchronisiert werdenAWS IoT Core. Andernfalls können Sie diesen Abschnitt überspringen.

So überprüfen und aktualisieren Sie die AWS IoT Richtlinie eines Core-Geräts

1. Wählen Sie im Navigationsmenü der AWS IoT Greengrass Konsole die Option Core-Geräte aus.
2. Wählen Sie auf der Seite Core-Geräte das zu aktualisierende Core-Gerät aus.
3. Wählen Sie auf der Detailseite des Core-Geräts den Link zum Objekt des Core-Geräts aus. Dieser Link öffnet die Seite mit den Objektdetails in der -AWS IoTKonsole.
4. Wählen Sie auf der Seite mit den Objektdetails die Option Zertifikate aus.
5. Wählen Sie auf der Registerkarte Zertifikate das aktive Zertifikat des Objekts aus.
6. Wählen Sie auf der Seite mit den Zertifikatsdetails Richtlinien aus.
7. Wählen Sie auf der Registerkarte Richtlinien die zu überprüfende und zu aktualisierende AWS IoT Richtlinie aus. Sie können die erforderlichen Berechtigungen zu jeder Richtlinie hinzufügen, die dem aktiven Zertifikat des Core-Geräts angefügt ist.

> **Note**
>
> Wenn Sie das AWS IoT Greengrass-Core-Softwareinstallationsprogramm zur Bereitstellung von Ressourcen verwendet haben, haben Sie zwei AWS IoT Richtlinien. Wir empfehlen Ihnen, die Richtlinie mit dem Namen auszuwählenGreengrassV2IoTThingPolicy, falls sie vorhanden ist. Core-Geräte, die Sie

mit dem Schnellinstallationsprogramm erstellen, verwenden diesen Richtliniennamen standardmäßig. Wenn Sie dieser Richtlinie Berechtigungen hinzufügen, erteilen Sie diese Berechtigungen auch anderen -Core-Geräten, die diese Richtlinie verwenden.

8. Wählen Sie in der Richtlinienübersicht die Option Aktive Version bearbeiten aus.
9. Überprüfen Sie die Richtlinie auf die erforderlichen Berechtigungen und fügen Sie alle erforderlichen Berechtigungen hinzu, die fehlen.
10. 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.
11. Wählen Sie Als neue Version speichern aus.

Schritt 2: Aktivieren der Client-Geräteunterstützung

Damit ein Client-Gerät Cloud Discovery verwenden kann, um eine Verbindung zu einem Core-Gerät herzustellen, müssen Sie die Geräte zuordnen. Wenn Sie ein Client-Gerät einem Core-Gerät zuordnen, ermöglichen Sie diesem Client-Gerät, die IP-Adressen und Zertifikate des Core-Geräts abzurufen, die für die Verbindung verwendet werden sollen.

Damit Client-Geräte eine sichere Verbindung zu einem Core-Gerät herstellen und mit Greengrass-Komponenten und kommunizieren könnenAWS IoT Core, stellen Sie die folgenden Greengrass-Komponenten auf dem Core-Gerät bereit:

- Authentifizierung auf Client-Geräten (`aws.greengrass.clientdevices.Auth`)

  Stellen Sie die Authentifizierungskomponente des Client-Geräts bereit, um Client-Geräte zu authentifizieren und Client-Geräteaktionen zu autorisieren. Diese Komponente ermöglicht es Ihren AWS IoT Objekten, eine Verbindung zu einem Core-Gerät herzustellen.

  Diese Komponente erfordert eine gewisse Konfiguration, um sie zu verwenden. Sie müssen Gruppen von Client-Geräten und die Operationen angeben, zu deren Ausführung jede Gruppe berechtigt ist, z. B. zum Herstellen einer Verbindung und Kommunikation über MQTT. Weitere Informationen finden Sie unter Konfiguration der Client-Geräte-Authentifizierungskomponente .
- MQTT 3.1.1-Broker (Moquette) (`aws.greengrass.clientdevices.mqtt.Moquette`)

  Stellen Sie die Moquette MQTT-Brokerkomponente bereit, um einen leichtgewichtigen MQTT-Broker auszuführen. Der Moquette MQTT-Broker ist mit MQTT 3.1.1 konform und bietet lokale

Unterstützung für QoS 0, QoS 1, QoS 2, beibehaltene Nachrichten, Last-Will-Nachrichten und persistente Abonnements.

Sie müssen diese Komponente nicht für ihre Verwendung konfigurieren. Sie können jedoch den Port konfigurieren, an dem diese Komponente den MQTT-Broker betreibt. Standardmäßig wird Port 8883 verwendet.

- MQTT-Brücke (`aws.greengrass.clientdevices.mqtt.Bridge`)

  (Optional) Stellen Sie die MQTT-Bridge-Komponente bereit, um Nachrichten zwischen Client-Geräten (lokales MQTT), lokaler Veröffentlichung/Abonnement und AWS IoT Core MQTT weiterzuleiten. Konfigurieren Sie diese Komponente, um Client-Geräte mit zu synchronisieren AWS IoT Core und von Greengrass-Komponenten aus mit Client-Geräten zu interagieren.

  Für diese Komponente ist eine Konfiguration erforderlich. Sie müssen die Themenzuordnungen angeben, an die diese Komponente Nachrichten weiterleitet. Weitere Informationen finden Sie unter Konfiguration der MQTT-Bridge-Komponente .

- IP-Detektor (`aws.greengrass.clientdevices.IPDetector`)

  (Optional) Stellen Sie die IP-Detektorkomponente bereit, um die MQTT-Broker-Endpunkte des Core-Geräts automatisch an den AWS IoT Greengrass Cloud-Service zu melden. Sie können diese Komponente nicht verwenden, wenn Sie über eine komplexe Netzwerkeinrichtung verfügen, z. B. eine, bei der ein Router den MQTT-Broker-Port an das Core-Gerät weiterleitet.

  Sie müssen diese Komponente nicht für ihre Verwendung konfigurieren.

In diesem Abschnitt verwenden Sie die AWS IoT Greengrass Konsole, um Client-Geräte zuzuordnen und Client-Gerätekomponenten auf einem Core-Gerät bereitzustellen.

So aktivieren Sie die Client-Geräteunterstützung

1. Navigieren Sie zur AWS IoT Greengrass-Konsole.
2. Wählen Sie im linken Navigationsmenü Core-Geräte aus.
3. Wählen Sie auf der Seite Core-Geräte das Core-Gerät aus, auf dem Sie die Client-Geräteunterstützung aktivieren möchten.
4. Wählen Sie auf der Detailseite des Core-Geräts die Registerkarte Client-Geräte aus.
5. Wählen Sie auf der Registerkarte Client-Geräte die Option Cloud-Erkennung konfigurieren aus.

Die Seite Core-Geräteerkennung konfigurieren wird geöffnet. Auf dieser Seite können Sie Client-Geräte einem Core-Gerät zuordnen und Client-Gerätekomponenten bereitstellen. Auf dieser Seite wird das Core-Gerät für Sie in Schritt 1: Ziel-Core-Geräte auswählen ausgewählt.

> **Note**
>
> Sie können diese Seite auch verwenden, um die Erkennung von Kerngeräten für eine Objektgruppe zu konfigurieren. Wenn Sie diese Option wählen, können Sie Client-Gerätekomponenten auf allen Core-Geräten in einer Objektgruppe bereitstellen. Wenn Sie diese Option wählen, müssen Sie Client-Geräte jedoch später nach dem Erstellen der Bereitstellung manuell jedem Core-Gerät zuordnen. In diesem Tutorial konfigurieren Sie ein einzelnes Core-Gerät.

6. Ordnen Sie in Schritt 2: Zuordnen von Client-Geräten das AWS IoT Objekt des Client-Geräts dem Core-Gerät zu. Auf diese Weise kann das Client-Gerät die Cloud-Erkennung verwenden, um die Konnektivitätsinformationen und Zertifikate des Core-Geräts abzurufen. Gehen Sie wie folgt vor:

   a. Wählen Sie Client-Geräte zuordnen aus.

   b. Geben Sie im Modal Client-Geräte dem Core-Gerät zuordnen den Namen des AWS IoT Objekts ein, das zugeordnet werden soll.

   c. Wählen Sie Hinzufügen aus.

   d. Wählen Sie Associate aus.

7. Stellen Sie in Schritt 3: Konfigurieren und Bereitstellen von Greengrass-Komponenten Komponenten bereit, um Client-Geräteunterstützung zu ermöglichen. Wenn das Ziel-Core-Gerät über eine frühere Bereitstellung verfügt, wird diese Bereitstellung auf dieser Seite geändert. Andernfalls erstellt diese Seite eine neue Bereitstellung für das Core-Gerät. Gehen Sie wie folgt vor, um die Komponenten des Client-Geräts zu konfigurieren und bereitzustellen:

   a. Das Core-Gerät muss Greengrass-Kern v2.6.0 oder höher ausführen, um dieses Tutorial abzuschließen. Wenn auf dem Core-Gerät eine frühere Version ausgeführt wird, gehen Sie wie folgt vor:

      i. Aktivieren Sie das Kontrollkästchen, um die aws.greengrass.Nucleus Komponente bereitzustellen.

      ii. Wählen Sie für die aws.greengrass.Nucleus Komponente Konfiguration bearbeiten aus.

iii. Wählen Sie für Komponentenversion Version 2.6.0 oder höher aus.

iv. Wählen Sie Bestätigen aus.

> **Note**
>
> Wenn Sie den Greengrass-Kern von einer früheren Nebenversion aktualisieren und das CoreAWS-Gerät von bereitgestellte Komponenten ausführt, die vom Kern abhängen, müssen Sie auch die von AWSbereitgestellten Komponenten auf neuere Versionen aktualisieren. Sie können die Version dieser Komponenten konfigurieren, wenn Sie die Bereitstellung später in diesem Tutorial überprüfen. Weitere Informationen finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

b. Wählen Sie für die aws.greengrass.clientdevices.Auth Komponente Konfiguration bearbeiten aus.

c. Konfigurieren Sie im Modal Konfiguration bearbeiten für die Authentifizierungskomponente des Client-Geräts eine Autorisierungsrichtlinie, die es Client-Geräten ermöglicht, den MQTT-Broker auf dem Core-Gerät zu veröffentlichen und zu abonnieren. Gehen Sie wie folgt vor:

i. Geben Sie unter Konfiguration im Block Konfiguration zum Zusammenführen von Code die folgende Konfiguration ein, die eine Client-Geräteautorisierungsrichtlinie enthält. Jede Gerätegruppenautorisierungsrichtlinie gibt eine Reihe von Aktionen und die Ressourcen an, auf denen ein Client-Gerät diese Aktionen ausführen kann.

- Diese Richtlinie ermöglicht Client-Geräten, deren Namen mit beginnen`MyClientDevice`, eine Verbindung herzustellen und zu allen MQTT-Themen zu kommunizieren. Ersetzen Sie *`MyClientDevice*`* durch den Namen des AWS IoT Objekts, das als Client-Gerät verbunden werden soll. Sie können auch einen Namen mit dem `*` Platzhalter angeben, der mit dem Namen des Client-Geräts übereinstimmt. Der `*` Platzhalter muss am Ende des Namens stehen.

  Wenn Sie ein zweites Client-Gerät zum Herstellen einer Verbindung haben, ersetzen Sie *`MyOtherClientDevice*`* durch den Namen dieses Client-Geräts oder ein Platzhaltermuster, das mit dem Namen dieses Client-Geräts übereinstimmt. Andernfalls können Sie diesen Abschnitt der Auswahlregel entfernen oder beibehalten, die es Client-Geräten mit übereinstimmenden Namen ermöglicht, eine Verbindung herzustellen und `MyOtherClientDevice*` zu kommunizieren.

- Diese Richtlinie verwendet einen `OR` Operator, um Client-Geräte, deren Namen mit beginnen, auch `MyOtherClientDevice` die Verbindung und Kommunikation zu allen MQTT-Themen zu ermöglichen. Sie können diese Klausel in der Auswahlregel entfernen oder sie so ändern, dass sie den Client-Geräten entspricht, die eine Verbindung herstellen möchten.
- Diese Richtlinie ermöglicht es den Client-Geräten, alle MQTT-Themen zu veröffentlichen und zu abonnieren. Um bewährte Sicherheitsmethoden zu befolgen, beschränken Sie die - `mqtt:publish` und -`mqtt:subscribe`Operationen auf den minimalen Satz von Themen, die die Client-Geräte für die Kommunikation verwenden.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice* OR
 thingName: MyOtherClientDevice*",
        "policyName": "MyClientDevicePolicy"
      }
    },
    "policies": {
      "MyClientDevicePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish to all
 topics.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "*"
```

```
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to all
 topics.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

Weitere Informationen finden Sie unter Konfiguration der Client-Geräte-Authentifizierungskomponente .

ii. Wählen Sie Bestätigen aus.

d. Wählen Sie für die aws.greengrass.clientdevices.mqtt.Bridge Komponente Konfiguration bearbeiten aus.

e. Konfigurieren Sie im Modal Konfiguration bearbeiten für die MQTT-Bridge-Komponente eine Themenzuordnung, die MQTT-Nachrichten von Client-Geräten an weiterleitetAWS IoT Core. Gehen Sie wie folgt vor:

i. Geben Sie unter Konfiguration im Block Konfiguration zum Zusammenführen von Code die folgende Konfiguration ein. Diese Konfiguration gibt an, MQTT-Nachrichten auf dem `clients/+/hello/world` Themenfilter von Clientgeräten an den AWS IoT Core Cloud-Service weiterzuleiten. Dieser Themenfilter entspricht beispielsweise dem `clients/MyClientDevice1/hello/world` Thema.

```
{
  "mqttTopicMapping": {
    "HelloWorldIotCoreMapping": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
```

```
}
```

Weitere Informationen finden Sie unter Konfiguration der MQTT-Bridge-Komponente .

ii. Wählen Sie Bestätigen aus.

8. Wählen Sie Überprüfen und Bereitstellen, um die Bereitstellung zu überprüfen, die diese Seite für Sie erstellt.
9. Wenn Sie die Greengrass-Servicerolle noch nicht in dieser Region eingerichtet haben, öffnet die Konsole ein Modal, um die Servicerolle für Sie einzurichten. Die Authentifizierungskomponente des Client-Geräts verwendet diese Servicerolle, um die Identität von Client-Geräten zu überprüfen, und die IP-Detektorkomponente verwendet diese Servicerolle, um die Konnektivitätsinformationen des Core-Geräts zu verwalten. Klicken Sie aufGewähren von Berechtigungenaus.
10. Wählen Sie auf der Seite Überprüfen die Option Bereitstellen aus, um die Bereitstellung auf dem Core-Gerät zu starten.
11. Um zu überprüfen, ob die Bereitstellung erfolgreich ist, überprüfen Sie den Status der Bereitstellung und überprüfen Sie die Protokolle auf dem Core-Gerät. Um den Status der Bereitstellung auf dem Core-Gerät zu überprüfen, können Sie in der Bereitstellungsübersicht Ziel auswählen. Weitere Informationen finden Sie hier:

    - Prüfen des Bereitstellungsstatus
    - Überwachen von AWS IoT Greengrass Protokollen

Schritt 3: Verbinden von Client-Geräten

Client-Geräte können die verwenden, AWS IoT Device SDK um ein Core-Gerät zu erkennen, eine Verbindung herzustellen und mit ihm zu kommunizieren. Das Client-Gerät muss ein -AWS IoTObjekt sein. Weitere Informationen finden Sie unter Erstellen eines Objekts im AWS IoT Core - Entwicklerhandbuch.

In diesem Abschnitt installieren Sie AWS IoT Device SDK v2 für Python und führen die Greengrass-Discovery-Beispielanwendung aus der ausAWS IoT Device SDK.

> **Note**
>
> Die AWS IoT Device SDK ist auch in anderen Programmiersprachen verfügbar. In diesem Tutorial wird AWS IoT Device SDK v2 für Python verwendet, aber Sie können die anderen

SDKs für Ihren Anwendungsfall erkunden. Weitere Informationen finden Sie unter AWS IoT Geräte-SDKs im AWS IoT Core -Entwicklerhandbuch.

So verbinden Sie ein Client-Gerät mit einem Core-Gerät

1. Laden Sie AWS IoT Device SDK v2 für Python herunter und installieren Sie es auf dem AWS IoT Objekt, um eine Verbindung als Client-Gerät herzustellen.

   Gehen Sie auf dem Client-Gerät wie folgt vor:

   a. Klonen Sie das Repository AWS IoT Device SDK v2 für Python, um es herunterzuladen.

   ```
   git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
   ```

   b. Installieren Sie AWS IoT Device SDK v2 für Python.

   ```
   python3 -m pip install --user ./aws-iot-device-sdk-python-v2
   ```

2. Wechseln Sie in den Ordner Beispiele in AWS IoT Device SDK v2 für Python.

   ```
   cd aws-iot-device-sdk-python-v2/samples
   ```

3. Führen Sie die Greengrass-Erkennungs-Beispielanwendung aus. Diese Anwendung erwartet Argumente, die den Objektnamen des Client-Geräts, das zu verwendende MQTT-Thema und die Nachricht sowie die Zertifikate angeben, die die Verbindung authentifizieren und sichern. Im folgenden Beispiel wird eine Hello World-Nachricht an das `clients/`*`MyClientDevice1`*`/hello/world` Thema gesendet.

   > **Note**
   >
   > Dieses Thema entspricht dem Thema, in dem Sie die MQTT-Brücke so konfiguriert haben, dass Nachrichten AWS IoT Core an weiterleitet werden.

   - Ersetzen Sie *`MyClientDevice1`* durch den Objektnamen des Client-Geräts.
   - Ersetzen Sie *`~/certs/AmazonRootCA1.pem`* durch den Pfad zum Amazon-Root-CA-Zertifikat auf dem Client-Gerät.

- Ersetzen Sie *~/certs/device.pem.crt* durch den Pfad zum Gerätezertifikat auf dem Client-Gerät.
- Ersetzen Sie *~/certs/private.pem.key* durch den Pfad zur Datei mit dem privaten Schlüssel auf dem Client-Gerät.
- Ersetzen Sie *us-east-1* durch die AWS Region, in der Ihr Client-Gerät und Ihr Core-Gerät betrieben werden.

```
python3 basic_discovery.py \\
  --thing_name MyClientDevice1 \\
  --topic 'clients/MyClientDevice1/hello/world' \\
  --message 'Hello World!' \\
  --ca_file ~/certs/AmazonRootCA1.pem \\
  --cert ~/certs/device.pem.crt \\
  --key ~/certs/private.pem.key \\
  --region us-east-1 \\
  --verbosity Warn
```

Die Discovery-Beispielanwendung sendet die Nachricht 10 Mal und trennt die Verbindung. Es abonniert auch dasselbe Thema, in dem es Nachrichten veröffentlicht. Wenn die Ausgabe angibt, dass die Anwendung MQTT-Nachrichten zum Thema erhalten hat, kann das Client-Gerät erfolgreich mit dem Core-Gerät kommunizieren.

```
Performing greengrass discovery...
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup
coreDevice-MyGreengrassCore',
 cores=[awsiot.greengrass_discovery.GGCore(thing_arn='arn:aws:iot:us-
east-1:123456789012:thing/MyGreengrassCore',
 connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='203.0.113.0',
 host_address='203.0.113.0', metadata='', port=8883)])],
 certificate_authorities=['-----BEGIN CERTIFICATE-----\
MIICiT...EXAMPLE=\
-----END CERTIFICATE-----\
'])])
Trying core arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore at host
 203.0.113.0 port 8883
Connected!
Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 0}
```

```
Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 0}'
Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 1}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 1}'

...

Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 9}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 9}'
```

Wenn die Anwendung stattdessen einen Fehler ausgibt, finden Sie weitere Informationen unter Fehlerbehebung bei Greengrass-Erkennungsproblemen.

Sie können auch die Greengrass-Protokolle auf dem Core-Gerät anzeigen, um zu überprüfen, ob das Client-Gerät erfolgreich eine Verbindung herstellt und Nachrichten sendet. Weitere Informationen finden Sie unter Überwachen von AWS IoT Greengrass Protokollen.

4. Stellen Sie sicher, dass die MQTT-Brücke die Nachrichten vom Client-Gerät an weiterleitetAWS IoT Core. Sie können den MQTT-Testclient in der AWS IoT Core Konsole verwenden, um einen MQTT-Themenfilter zu abonnieren. Gehen Sie wie folgt vor:

   a. Navigieren Sie zur AWS IoT-Konsole.

   b. Wählen Sie im linken Navigationsmenü unter Test die Option MQTT-Testclient aus.

   c. Geben Sie auf der Registerkarte Thema abonnieren für Themenfilter ein, `clients/+/hello/world` um Client-Gerätenachrichten vom Core-Gerät zu abonnieren.

   d. Wählen Sie Subscribe (Abonnieren) aus.

   e. Führen Sie die Anwendung zum Veröffentlichen/Abonnieren erneut auf dem Client-Gerät aus.

      Der MQTT-Testclient zeigt die Nachrichten, die Sie vom Client-Gerät senden, zu Themen an, die diesem Themenfilter entsprechen.

Schritt 4: Entwickeln einer Komponente, die mit Client-Geräten kommuniziert

Sie können Greengrass-Komponenten entwickeln, die mit Client-Geräten kommunizieren. Komponenten verwenden Interprocess Communication (IPC) und die lokale Publish/Subscribe-Schnittstelle, um auf einem Core-Gerät zu kommunizieren. Um mit Client-Geräten zu interagieren, konfigurieren Sie die MQTT-Bridge-Komponente so, dass Nachrichten zwischen Client-Geräten und der lokalen Publish/Subscribe-Schnittstelle weitergeleitet werden.

In diesem Abschnitt aktualisieren Sie die MQTT-Bridge-Komponente, um Nachrichten von Client-Geräten an die lokale Veröffentlichungs-/Abonnementschnittstelle weiterzuleiten. Anschließend entwickeln Sie eine Komponente, die diese Nachrichten abonniert und die Nachrichten druckt, wenn sie empfangen werden.

So entwickeln Sie eine Komponente, die mit Clientgeräten kommuniziert

1. Überarbeiten Sie die Bereitstellung auf dem Core-Gerät und konfigurieren Sie die MQTT-Bridge-Komponente so, dass Nachrichten von Client-Geräten an das lokale Veröffentlichen/Abonnement weitergeleitet werden. Gehen Sie wie folgt vor:

   a. Navigieren Sie zur AWS IoT Greengrass-Konsole.

   b. Wählen Sie im linken Navigationsmenü Core-Geräte aus.

   c. Wählen Sie auf der Seite Core-Geräte das Core-Gerät aus, das Sie für dieses Tutorial verwenden.

   d. Wählen Sie auf der Detailseite des Core-Geräts die Registerkarte Client-Geräte aus.

   e. Wählen Sie auf der Registerkarte Client-Geräte die Option Cloud-Erkennung konfigurieren aus.

      Die Seite Core-Geräteerkennung konfigurieren wird geöffnet. Auf dieser Seite können Sie ändern oder konfigurieren, welche Client-Gerätekomponenten auf dem Core-Gerät bereitgestellt werden.

   f. Wählen Sie in Schritt 3 für die aws.greengrass.clientdevices.mqtt.Bridge Komponente die Option Konfiguration bearbeiten aus.

   g. Konfigurieren Sie im Modal Konfiguration bearbeiten für die MQTT-Bridge-Komponente eine Themenzuordnung, die MQTT-Nachrichten von Client-Geräten an die lokale Veröffentlichungs-/Abonnementschnittstelle weiterleitet. Gehen Sie wie folgt vor:

i. Geben Sie unter Konfiguration im Block Konfiguration zum Zusammenführen von Code die folgende Konfiguration ein. Diese Konfiguration gibt an, MQTT-Nachrichten zu Themen weiterzuleiten, die dem `clients/+/hello/world` Themenfilter von Clientgeräten an den AWS IoT Core Cloud-Service und den lokalen Greengrass-Publish/Subscribe-Broker entsprechen.

```
{
  "mqttTopicMapping": {
    "HelloWorldIotCoreMapping": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "HelloWorldPubsubMapping": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "Pubsub"
    }
  }
}
```

Weitere Informationen finden Sie unter Konfiguration der MQTT-Bridge-Komponente .

ii. Wählen Sie Bestätigen aus.

h. Wählen Sie Überprüfen und Bereitstellen, um die Bereitstellung zu überprüfen, die diese Seite für Sie erstellt.

i. Wählen Sie auf der Seite Überprüfen die Option Bereitstellen aus, um die Bereitstellung auf dem Core-Gerät zu starten.

j. Um zu überprüfen, ob die Bereitstellung erfolgreich ist, überprüfen Sie den Status der Bereitstellung und überprüfen Sie die Protokolle auf dem Core-Gerät. Um den Status der Bereitstellung auf dem Core-Gerät zu überprüfen, können Sie in der Bereitstellungsübersicht Ziel auswählen. Weitere Informationen finden Sie hier:

- Prüfen des Bereitstellungsstatus
- Überwachen von AWS IoT Greengrass Protokollen

2. Entwickeln und stellen Sie eine Greengrass-Komponente bereit, die Hello-World-Nachrichten von Client-Geräten abonniert. Gehen Sie wie folgt vor:

a. Erstellen Sie Ordner für Rezepte und Artefakte auf dem Core-Gerät.

Linux or Unix

```
mkdir recipes
mkdir -p artifacts/com.example.clientdevices.MyHelloWorldSubscriber/1.0.0
```

Windows Command Prompt (CMD)

```
mkdir recipes
mkdir artifacts\com.example.clientdevices.MyHelloWorldSubscriber\1.0.0
```

PowerShell

```
mkdir recipes
mkdir artifacts\com.example.clientdevices.MyHelloWorldSubscriber\1.0.0
```

> **Important**
>
> Sie müssen das folgende Format für den Pfad des Artefaktordners verwenden. Fügen Sie den Komponentennamen und die Version ein, die Sie im Rezept angeben.
>
> ```
> artifacts/componentName/componentVersion/
> ```

b. Verwenden Sie einen Texteditor, um ein Komponentenrezept mit dem folgenden Inhalt zu erstellen. Dieses Rezept gibt an, dass AWS IoT Device SDK v2 für Python installiert und ein Skript ausgeführt werden soll, das das Thema abonniert und Nachrichten druckt.

Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen der Datei zu verwenden.

```
nano recipes/com.example.clientdevices.MyHelloWorldSubscriber-1.0.0.json
```

Kopieren Sie das folgende Rezept in die Datei .

```
{
```

```
 "RecipeFormatVersion": "2020-01-25",
 "ComponentName": "com.example.clientdevices.MyHelloWorldSubscriber",
 "ComponentVersion": "1.0.0",
 "ComponentDescription": "A component that subscribes to Hello World messages
from client devices.",
 "ComponentPublisher": "Amazon",
 "ComponentConfiguration": {
   "DefaultConfiguration": {
     "accessControl": {
       "aws.greengrass.ipc.pubsub": {
         "com.example.clientdevices.MyHelloWorldSubscriber:pubsub:1": {
           "policyDescription": "Allows access to subscribe to all topics.",
           "operations": [
             "aws.greengrass#SubscribeToTopic"
           ],
           "resources": [
             "*"
           ]
         }
       }
     }
   }
 },
 "Manifests": [
   {
     "Platform": {
       "os": "linux"
     },
     "Lifecycle": {
       "install": "python3 -m pip install --user awsiotsdk",
       "run": "python3 -u {artifacts:path}/hello_world_subscriber.py"
     }
   },
   {
     "Platform": {
       "os": "windows"
     },
     "Lifecycle": {
       "install": "py -3 -m pip install --user awsiotsdk",
       "run": "py -3 -u {artifacts:path}/hello_world_subscriber.py"
     }
   }
 ]
```

```
}
```

c. Verwenden Sie einen Texteditor, um ein Python-Skriptartefakt mit dem Namen `hello_world_subscriber.py` mit dem folgenden Inhalt zu erstellen. Diese Anwendung verwendet den IPC-Service zum Veröffentlichen/Abonnieren, um das `clients/+/hello/world` Thema zu abonnieren und Nachrichten zu drucken, die sie empfängt.

Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen der Datei zu verwenden.

```
nano artifacts/com.example.clientdevices.MyHelloWorldSubscriber/1.0.0/
hello_world_subscriber.py
```

Kopieren Sie den folgenden Python-Code in die Datei .

```
import sys
import time
import traceback

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2

CLIENT_DEVICE_HELLO_WORLD_TOPIC = 'clients/+/hello/world'
TIMEOUT = 10


def on_hello_world_message(event):
    try:
        message = str(event.binary_message.message, 'utf-8')
        print('Received new message: %s' % message)
    except:
        traceback.print_exc()


try:
    ipc_client = GreengrassCoreIPCClientV2()

    # SubscribeToTopic returns a tuple with the response and the operation.
    _, operation = ipc_client.subscribe_to_topic(
        topic=CLIENT_DEVICE_HELLO_WORLD_TOPIC,
 on_stream_event=on_hello_world_message)
    print('Successfully subscribed to topic: %s' %
          CLIENT_DEVICE_HELLO_WORLD_TOPIC)
```

```
    # Keep the main thread alive, or the process will exit.
    try:
        while True:
            time.sleep(10)
    except InterruptedError:
        print('Subscribe interrupted.')

    operation.close()
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

> **Note**
>
> Diese Komponente verwendet den IPC-Client V2 in der AWS IoT Device SDK v2 für Python, um mit der AWS IoT Greengrass Core-Software zu kommunizieren. Im Vergleich zum ursprünglichen IPC-Client reduziert der IPC-Client V2 die Menge an Code, die Sie schreiben müssen, um IPC in benutzerdefinierten Komponenten zu verwenden.

d. Verwenden Sie die Greengrass-CLI, um die Komponente bereitzustellen.

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.clientdevices.MyHelloWorldSubscriber=1.0.0"
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2/bin/greengrass-cli deployment create ^
  --recipeDir recipes ^
  --artifactDir artifacts ^
  --merge "com.example.clientdevices.MyHelloWorldSubscriber=1.0.0"
```

PowerShell

```
C:\greengrass\v2/bin/greengrass-cli deployment create `
  --recipeDir recipes `
  --artifactDir artifacts `
  --merge "com.example.clientdevices.MyHelloWorldSubscriber=1.0.0"
```

3. Sehen Sie sich die Komponentenprotokolle an, um zu überprüfen, ob die Komponente erfolgreich installiert wurde und das Thema abonniert hat.

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/
   com.example.clientdevices.MyHelloWorldSubscriber.log
   ```

   PowerShell

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

   Sie können den Protokoll-Feed geöffnet lassen, um zu überprüfen, ob das Core-Gerät Nachrichten empfängt.

4. Führen Sie auf dem Client-Gerät die Greengrass-Erkennungsanwendung erneut aus, um Nachrichten an das Core-Gerät zu senden.

   ```
   python3 basic_discovery.py \\
     --thing_name MyClientDevice1 \\
     --topic 'clients/MyClientDevice1/hello/world' \\
     --message 'Hello World!' \\
     --ca_file ~/certs/AmazonRootCA1.pem \\
     --cert ~/certs/device.pem.crt \\
     --key ~/certs/private.pem.key \\
     --region us-east-1 \\
     --verbosity Warn
   ```

5. Zeigen Sie die Komponentenprotokolle erneut an, um zu überprüfen, ob die Komponente die Nachrichten vom Client-Gerät empfängt und druckt.

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/
com.example.clientdevices.MyHelloWorldSubscriber.log
```

PowerShell

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

Schritt 5: Entwickeln einer Komponente, die mit Schatten von Client-Geräten interagiert

Sie können Greengrass-Komponenten entwickeln, die mit AWS IoT den Geräteschatten des Client-Geräts interagieren. Ein Schatten ist ein JSON-Dokument, das die aktuellen oder gewünschten Statusinformationen für ein -AWS IoTObjekt speichert, z. B. ein Client-Gerät. Benutzerdefinierte Komponenten können auf die Schatten von Client-Geräten zugreifen, um ihren Status zu verwalten, auch wenn das Client-Gerät nicht mit verbunden istAWS IoT. Jedes AWS IoT Objekt hat einen unbenannten Schatten, und Sie können auch mehrere benannte Schatten für jedes Objekt erstellen.

In diesem Abschnitt stellen Sie die Shadow Manager-Komponente bereit, um Schatten auf dem Core-Gerät zu verwalten. Sie aktualisieren auch die MQTT-Bridge-Komponente, um Shadow-Nachrichten zwischen Client-Geräten und der Shadow-Manager-Komponente weiterzuleiten. Anschließend entwickeln Sie eine Komponente, die die Schatten der Client-Geräte aktualisiert, und führen eine Beispielanwendung auf den Client-Geräten aus, die auf Schattenaktualisierungen von der Komponente reagiert. Diese Komponente stellt eine intelligente Light-Management-Anwendung dar, bei der das Core-Gerät den Farbstatus von Smart-Lights verwaltet, die sich als Client-Geräte mit ihm verbinden.

So entwickeln Sie eine Komponente, die mit Client-Geräteschatten interagiert

1. Überarbeiten Sie die Bereitstellung auf dem Core-Gerät, um die Shadow-Manager-Komponente bereitzustellen, und konfigurieren Sie die MQTT-Bridge-Komponente so, dass Schattennachrichten zwischen Client-Geräten und lokaler Veröffentlichung/Abonnement weitergeleitet werden, wo der Shadow Manager kommuniziert. Gehen Sie wie folgt vor:

   a. Navigieren Sie zur AWS IoT Greengrass-Konsole.

b. Wählen Sie im linken Navigationsmenü Core-Geräte aus.

c. Wählen Sie auf der Seite Core-Geräte das Core-Gerät aus, das Sie für dieses Tutorial verwenden.

d. Wählen Sie auf der Detailseite des Core-Geräts die Registerkarte Client-Geräte aus.

e. Wählen Sie auf der Registerkarte Client-Geräte die Option Cloud-Erkennung konfigurieren aus.

   Die Seite Core-Geräteerkennung konfigurieren wird geöffnet. Auf dieser Seite können Sie ändern oder konfigurieren, welche Client-Gerätekomponenten auf dem Core-Gerät bereitgestellt werden.

f. Wählen Sie in Schritt 3 für die aws.greengrass.clientdevices.mqtt.Bridge Komponente die Option Konfiguration bearbeiten aus.

g. Konfigurieren Sie im Modal Konfiguration bearbeiten für die MQTT-Bridge-Komponente eine Themenzuordnung, die MQTT-Nachrichten an Geräteschattenthemen zwischen Client-Geräten und der lokalen Veröffentlichungs-/Abonnementschnittstelle weiterleitet. Sie bestätigen auch, dass die Bereitstellung eine kompatible MQTT-Bridge-Version angibt. Die Unterstützung des Client-Geräteschattens erfordert MQTT Bridge v2.2.0 oder höher. Gehen Sie wie folgt vor:

   i. Wählen Sie für Komponentenversion Version 2.2.0 oder höher aus.

   ii. Geben Sie unter Konfiguration im Block Konfiguration zum Zusammenführen von Code die folgende Konfiguration ein. Diese Konfiguration gibt an, dass MQTT-Nachrichten zu Schattenthemen weitergeleitet werden sollen.

```
{
  "mqttTopicMapping": {
    "HelloWorldIotCoreMapping": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "HelloWorldPubsubMapping": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/#",
```

```
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Weitere Informationen finden Sie unter Konfiguration der MQTT-Bridge-Komponente .

iii. Wählen Sie Bestätigen aus.

h. Wählen Sie in Schritt 3 die aws.greengrass.ShadowManager Komponente aus, die bereitgestellt werden soll.

i. Wählen Sie Überprüfen und Bereitstellen, um die Bereitstellung zu überprüfen, die diese Seite für Sie erstellt.

j. Wählen Sie auf der Seite Überprüfen die Option Bereitstellen aus, um die Bereitstellung auf dem Core-Gerät zu starten.

k. Um zu überprüfen, ob die Bereitstellung erfolgreich ist, überprüfen Sie den Status der Bereitstellung und überprüfen Sie die Protokolle auf dem Core-Gerät. Um den Status der Bereitstellung auf dem Core-Gerät zu überprüfen, können Sie in der Bereitstellungsübersicht Ziel auswählen. Weitere Informationen finden Sie hier:

- Prüfen des Bereitstellungsstatus
- Überwachen von AWS IoT Greengrass Protokollen

2. Entwickeln und stellen Sie eine Greengrass-Komponente bereit, die Smart Light Client-Geräte verwaltet. Gehen Sie wie folgt vor:

a. Erstellen Sie einen Ordner mit den Artefakten der Komponente auf dem Core-Gerät.

Linux or Unix

```
mkdir -p artifacts/com.example.clientdevices.MySmartLightManager/1.0.0
```

Windows Command Prompt (CMD)

```
mkdir artifacts\com.example.clientdevices.MySmartLightManager\1.0.0
```

PowerShell

```
mkdir artifacts\com.example.clientdevices.MySmartLightManager\1.0.0
```

> **Important**
>
> Sie müssen das folgende Format für den Pfad des Artefaktordners verwenden. Fügen Sie den Komponentennamen und die Version ein, die Sie im Rezept angeben.
>
> ```
> artifacts/componentName/componentVersion/
> ```

b. Verwenden Sie einen Texteditor, um ein Komponentenrezept mit dem folgenden Inhalt zu erstellen. Dieses Rezept legt fest, dass AWS IoT Device SDK v2 für Python installiert und ein Skript ausgeführt werden soll, das mit den Schatten von Smart Light Client-Geräten interagiert, um ihre Farben zu verwalten.

   Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen der Datei zu verwenden.

   ```
   nano recipes/com.example.clientdevices.MySmartLightManager-1.0.0.json
   ```

   Kopieren Sie das folgende Rezept in die Datei .

   ```
   {
     "RecipeFormatVersion": "2020-01-25",
     "ComponentName": "com.example.clientdevices.MySmartLightManager",
     "ComponentVersion": "1.0.0",
     "ComponentDescription": "A component that interacts with smart light client
    devices.",
     "ComponentPublisher": "Amazon",
     "ComponentDependencies": {
       "aws.greengrass.Nucleus": {
   ```

```
      "VersionRequirement": "^2.6.0"
    },
    "aws.greengrass.ShadowManager": {
      "VersionRequirement": "^2.2.0"
    },
    "aws.greengrass.clientdevices.mqtt.Bridge": {
      "VersionRequirement": "^2.2.0"
    }
  },
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "smartLightDeviceNames": [],
      "accessControl": {
        "aws.greengrass.ShadowManager": {
          "com.example.clientdevices.MySmartLightManager:shadow:1": {
            "policyDescription": "Allows access to client devices' unnamed
shadows",
            "operations": [
              "aws.greengrass#GetThingShadow",
              "aws.greengrass#UpdateThingShadow"
            ],
            "resources": [
              "$aws/things/MyClientDevice*/shadow"
            ]
          }
        },
        "aws.greengrass.ipc.pubsub": {
          "com.example.clientdevices.MySmartLightManager:pubsub:1": {
            "policyDescription": "Allows access to client devices' unnamed
shadow updates",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "$aws/things/+/shadow/update/accepted"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
```

```
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk",
        "run": "python3 -u {artifacts:path}/smart_light_manager.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk",
        "run": "py -3 -u {artifacts:path}/smart_light_manager.py"
      }
    }
  ]
}
```

c. Verwenden Sie einen Texteditor, um ein Python-Skriptartefakt mit dem Namen `smart_light_manager.py` mit dem folgenden Inhalt zu erstellen. Diese Anwendung verwendet den Schatten-IPK-Service, um Schatten von Client-Geräten abzurufen und zu aktualisieren, und den lokalen Veröffentlichungs-/Abonnement-IPK-Service, um gemeldete Schattenaktualisierungen zu erhalten.

Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen der Datei zu verwenden.

```
nano artifacts/com.example.clientdevices.MySmartLightManager/1.0.0/
smart_light_manager.py
```

Kopieren Sie den folgenden Python-Code in die Datei .

```
import json
import random
import sys
import time
import traceback
from uuid import uuid4

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import ResourceNotFoundError
```

```
SHADOW_COLOR_PROPERTY = 'color'
CONFIGURATION_CLIENT_DEVICE_NAMES = 'smartLightDeviceNames'
COLORS = ['red', 'orange', 'yellow', 'green', 'blue', 'purple']
SHADOW_UPDATE_TOPIC = '$aws/things/+/shadow/update/accepted'
SET_COLOR_INTERVAL = 15


class SmartLightDevice():
    def __init__(self, client_device_name: str, reported_color: str = None):
        self.name = client_device_name
        self.reported_color = reported_color
        self.desired_color = None


class SmartLightDeviceManager():
    def __init__(self, ipc_client: GreengrassCoreIPCClientV2):
        self.ipc_client = ipc_client
        self.devices = {}
        self.client_tokens = set()
        self.shadow_update_accepted_subscription_operation = None
        self.client_device_names_configuration_subscription_operation = None
        self.update_smart_light_device_list()

    def update_smart_light_device_list(self):
        # Update the device list from the component configuration.
        response = self.ipc_client.get_configuration(
            key_path=[CONFIGURATION_CLIENT_DEVICE_NAMES])
        # Identify the difference between the configuration and the currently
 tracked devices.
        current_device_names = self.devices.keys()
        updated_device_names =
 response.value[CONFIGURATION_CLIENT_DEVICE_NAMES]
        added_device_names = set(updated_device_names) -
 set(current_device_names)
        removed_device_names = set(current_device_names) -
 set(updated_device_names)
        # Stop tracking any smart light devices that are no longer in the
 configuration.
        for name in removed_device_names:
            print('Removing %s from smart light device manager' % name)
            self.devices.pop(name)
        # Start tracking any new smart light devices that are in the
 configuration.
```

```
        for name in added_device_names:
            print('Adding %s to smart light device manager' % name)
            device = SmartLightDevice(name)
            device.reported_color = self.get_device_reported_color(device)
            self.devices[name] = device
            print('Current color for %s is %s' % (name,
device.reported_color))

    def get_device_reported_color(self, smart_light_device):
        try:
            response = self.ipc_client.get_thing_shadow(
                thing_name=smart_light_device.name, shadow_name='')
            shadow = json.loads(str(response.payload, 'utf-8'))
            if 'reported' in shadow['state']:
                return shadow['state']['reported'].get(SHADOW_COLOR_PROPERTY)
            return None
        except ResourceNotFoundError:
            return None

    def request_device_color_change(self, smart_light_device, color):
        # Generate and track a client token for the request.
        client_token = str(uuid4())
        self.client_tokens.add(client_token)
        # Create a shadow payload, which must be a blob.
        payload_json = {
            'state': {
                'desired': {
                    SHADOW_COLOR_PROPERTY: color
                }
            },
            'clientToken': client_token
        }
        payload = bytes(json.dumps(payload_json), 'utf-8')
        self.ipc_client.update_thing_shadow(
            thing_name=smart_light_device.name, shadow_name='',
payload=payload)
        smart_light_device.desired_color = color

    def subscribe_to_shadow_update_accepted_events(self):
        if self.shadow_update_accepted_subscription_operation == None:
            # SubscribeToTopic returns a tuple with the response and the
operation.
            _, self.shadow_update_accepted_subscription_operation =
self.ipc_client.subscribe_to_topic(
```

```
                topic=SHADOW_UPDATE_TOPIC,
on_stream_event=self.on_shadow_update_accepted_event)
            print('Successfully subscribed to shadow update accepted topic')

    def close_shadow_update_accepted_subscription(self):
        if self.shadow_update_accepted_subscription_operation is not None:
            self.shadow_update_accepted_subscription_operation.close()

    def on_shadow_update_accepted_event(self, event):
        try:
            message = str(event.binary_message.message, 'utf-8')
            accepted_payload = json.loads(message)
            # Check for reported states from smart light devices and ignore
desired states from components.
            if 'reported' in accepted_payload['state']:
                # Process this update only if it uses a client token created by
this component.
                client_token = accepted_payload.get('clientToken')
                if client_token is not None and client_token in
self.client_tokens:
                    self.client_tokens.remove(client_token)
                    shadow_state = accepted_payload['state']['reported']
                    if SHADOW_COLOR_PROPERTY in shadow_state:
                        reported_color = shadow_state[SHADOW_COLOR_PROPERTY]
                        topic = event.binary_message.context.topic
                        client_device_name = topic.split('/')[2]
                        if client_device_name in self.devices:
                            # Set the reported color for the smart light
device.
                            self.devices[client_device_name].reported_color =
reported_color
                            print(
                                'Received shadow update confirmation from
client device: %s' % client_device_name)
                    else:
                        print("Shadow update doesn't specify color")
        except:
            traceback.print_exc()

    def subscribe_to_client_device_name_configuration_updates(self):
        if self.client_device_names_configuration_subscription_operation ==
None:
            # SubscribeToConfigurationUpdate returns a tuple with the response
and the operation.
```

```
            _, self.client_device_names_configuration_subscription_operation =
 self.ipc_client.subscribe_to_configuration_update(
                key_path=[CONFIGURATION_CLIENT_DEVICE_NAMES],
 on_stream_event=self.on_client_device_names_configuration_update_event)
            print(
                'Successfully subscribed to configuration updates for smart
 light device names')

    def close_client_device_names_configuration_subscription(self):
        if self.client_device_names_configuration_subscription_operation is not
 None:

 self.client_device_names_configuration_subscription_operation.close()

    def on_client_device_names_configuration_update_event(self, event):
        try:
            if CONFIGURATION_CLIENT_DEVICE_NAMES in
 event.configuration_update_event.key_path:
                print('Received configuration update for list of client
 devices')
                self.update_smart_light_device_list()
        except:
            traceback.print_exc()


def choose_random_color():
    return random.choice(COLORS)

def main():
    try:
        # Create an IPC client and a smart light device manager.
        ipc_client = GreengrassCoreIPCClientV2()
        smart_light_manager = SmartLightDeviceManager(ipc_client)
        smart_light_manager.subscribe_to_shadow_update_accepted_events()

 smart_light_manager.subscribe_to_client_device_name_configuration_updates()
        try:
            # Keep the main thread alive, or the process will exit.
            while True:
                # Set each smart light device to a random color at a regular
 interval.
                for device_name in smart_light_manager.devices:
                    device = smart_light_manager.devices[device_name]
                    desired_color = choose_random_color()
```

```
                    print('Chose random color (%s) for %s' %
                          (desired_color, device_name))
                    if desired_color == device.desired_color:
                        print('Desired color for %s is already %s' %
                              (device_name, desired_color))
                    elif desired_color == device.reported_color:
                        print('Reported color for %s is already %s' %
                              (device_name, desired_color))
                    else:
                        smart_light_manager.request_device_color_change(
                            device, desired_color)
                        print('Requested color change for %s to %s' %
                              (device_name, desired_color))
                time.sleep(SET_COLOR_INTERVAL)
        except InterruptedError:
            print('Application interrupted')
        smart_light_manager.close_shadow_update_accepted_subscription()

 smart_light_manager.close_client_device_names_configuration_subscription()
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)


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

Diese Python-Anwendung führt Folgendes aus:

- Liest die Konfiguration der Komponente, um die Liste der zu verwaltenden Smart Light Client-Geräte abzurufen.
- Abonniert Konfigurationsaktualisierungsbenachrichtigungen mithilfe der SubscribeToConfigurationUpdate IPC-Operation. Die -AWS IoT GreengrassCore-Software sendet jedes Mal Benachrichtigungen, wenn sich die Konfiguration der Komponente ändert. Wenn die Komponente eine Benachrichtigung über die Konfigurationsaktualisierung erhält, aktualisiert sie die Liste der von ihr verwalteten Smart-Light-Client-Geräte.
- Ruft den Schatten jedes Smart Light Client-Geräts ab, um seinen ursprünglichen Farbstatus zu erhalten.

- Legt die Farbe jedes Smart Light Client-Geräts alle 15 Sekunden auf eine zufällige Farbe fest. Die Komponente aktualisiert den Objektschatten des Client-Geräts, um seine Farbe zu ändern. Dieser Vorgang sendet ein Schattendelta-Ereignis über MQTT an das Client-Gerät.
- Abonniert die Shadow-Aktualisierung akzeptierter Nachrichten auf der lokalen Veröffentlichungs-/Abonnementschnittstelle mithilfe des SubscribeToTopic IPC-Vorgangs. Diese Komponente empfängt diese Nachrichten, um die Farbe jedes Smart-Light-Client-Geräts zu verfolgen. Wenn ein Smart Light Client-Gerät eine Schattenaktualisierung erhält, sendet es eine MQTT-Nachricht, um zu bestätigen, dass es die Aktualisierung erhalten hat. Die MQTT-Brücke leitet diese Nachricht an die lokale Veröffentlichungs-/Abonnementschnittstelle weiter.

d. Verwenden Sie die Greengrass-CLI, um die Komponente bereitzustellen. Wenn Sie diese Komponente bereitstellen, geben Sie die Liste der Client-Geräte an, `smartLightDeviceNames`, deren Schatten sie verwaltet. Ersetzen Sie *`MyClientDevice1`* durch den Objektnamen des Clientgeräts.

   Linux or Unix

   ```
   sudo /greengrass/v2/bin/greengrass-cli deployment create \
     --recipeDir recipes \
     --artifactDir artifacts \
     --merge "com.example.clientdevices.MySmartLightManager=1.0.0" \
     --update-config '{
       "com.example.clientdevices.MySmartLightManager": {
         "MERGE": {
           "smartLightDeviceNames": [
             "MyClientDevice1"
           ]
         }
       }
     }'
   ```

   Windows Command Prompt (CMD)

   ```
   C:\greengrass\v2/bin/greengrass-cli deployment create ^
     --recipeDir recipes ^
     --artifactDir artifacts ^
     --merge "com.example.clientdevices.MySmartLightManager=1.0.0" ^
   ```

```
  --update-config '{"com.example.clientdevices.MySmartLightManager":
{"MERGE":{"smartLightDeviceNames":["MyClientDevice1"]}}}'
```

PowerShell

```
C:\greengrass\v2/bin/greengrass-cli deployment create `
  --recipeDir recipes `
  --artifactDir artifacts `
  --merge "com.example.clientdevices.MySmartLightManager=1.0.0" `
  --update-config '{
    "com.example.clientdevices.MySmartLightManager": {
      "MERGE": {
        "smartLightDeviceNames": [
          "MyClientDevice1"
        ]
      }
    }
  }'
```

3. Zeigen Sie die Komponentenprotokolle an, um zu überprüfen, ob die Komponente erfolgreich installiert wurde und ausgeführt wird.

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/
   com.example.clientdevices.MySmartLightManager.log
   ```

   PowerShell

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

   Die Komponente sendet Anfragen zum Ändern der Farbe des Smart Light Client-Geräts. Der Schattenmanager empfängt die Anforderung und legt den `desired` Status des Schattens fest. Das Smart Light Client-Gerät wird jedoch noch nicht ausgeführt, sodass sich der `reported` Status des Schattens nicht ändert. Die Protokolle der Komponente enthalten die folgenden Meldungen.

   ```
   2022-07-07T03:49:24.908Z [INFO] (Copier)
    com.example.clientdevices.MySmartLightManager: stdout. Chose random color (blue)
   ```

```
 for MyClientDevice1.
 {scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
 serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
2022-07-07T03:49:24.912Z [INFO] (Copier)
 com.example.clientdevices.MySmartLightManager: stdout.
 Requested color change for MyClientDevice1 to blue.
 {scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
 serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
```

Sie können den Protokoll-Feed geöffnet lassen, um zu sehen, wann die Komponente Nachrichten druckt.

4. Laden Sie eine Beispielanwendung herunter und führen Sie sie aus, die Greengrass Discovery verwendet und Geräteschattenaktualisierungen abonniert. Gehen Sie auf dem Client-Gerät wie folgt vor:

   a. Wechseln Sie in den Ordner Beispiele in AWS IoT Device SDK v2 für Python. Diese Beispielanwendung verwendet ein Befehlszeilen-Parsing-Modul im Ordner „Samples“.

   ```
   cd aws-iot-device-sdk-python-v2/samples
   ```

   b. Verwenden Sie einen Texteditor, um ein Python-Skript namens `basic_discovery_shadow.py` mit dem folgenden Inhalt zu erstellen. Diese Anwendung verwendet Greengrass-Erkennung und Schatten, um eine Eigenschaft zwischen dem Client-Gerät und dem Core-Gerät synchron zu halten.

   Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen der Datei zu verwenden.

   ```
   nano basic_discovery_shadow.py
   ```

   Kopieren Sie den folgenden Python-Code in die Datei .

   ```
   # Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
   # SPDX-License-Identifier: Apache-2.0.

   from awscrt import io
   from awscrt import mqtt
   from awsiot import iotshadow
   from awsiot.greengrass_discovery import DiscoveryClient
   from awsiot import mqtt_connection_builder
   ```

```
from concurrent.futures import Future
import sys
import threading
import traceback
from uuid import uuid4

# Parse arguments
import utils.command_line_utils;
cmdUtils = utils.command_line_utils.CommandLineUtils("Basic Discovery -
 Greengrass discovery example with device shadows.")
cmdUtils.add_common_mqtt_commands()
cmdUtils.add_common_topic_message_commands()
cmdUtils.add_common_logging_commands()
cmdUtils.register_command("key", "<path>", "Path to your key in PEM format.",
 True, str)
cmdUtils.register_command("cert", "<path>", "Path to your client certificate in
 PEM format.", True, str)
cmdUtils.remove_command("endpoint")
cmdUtils.register_command("thing_name", "<str>", "The name assigned to your IoT
 Thing", required=True)
cmdUtils.register_command("region", "<str>", "The region to connect through.",
 required=True)
cmdUtils.register_command("shadow_property", "<str>", "The name of the shadow
 property you want to change (optional, default='color'", default="color")
# Needs to be called so the command utils parse the commands
cmdUtils.get_args()

# Using globals to simplify sample code
is_sample_done = threading.Event()
mqtt_connection = None
shadow_thing_name = cmdUtils.get_command_required("thing_name")
shadow_property = cmdUtils.get_command("shadow_property")

SHADOW_VALUE_DEFAULT = "off"

class LockedData:
    def __init__(self):
        self.lock = threading.Lock()
        self.shadow_value = None
        self.disconnect_called = False
        self.request_tokens = set()

locked_data = LockedData()
```

```
def on_connection_interupted(connection, error, **kwargs):
    print('connection interrupted with error {}'.format(error))


def on_connection_resumed(connection, return_code, session_present, **kwargs):
    print('connection resumed with return code {}, session present
 {}'.format(return_code, session_present))

# Try IoT endpoints until we find one that works
def try_iot_endpoints():
    for gg_group in discover_response.gg_groups:
        for gg_core in gg_group.cores:
            for connectivity_info in gg_core.connectivity:
                try:
                    print('Trying core {} at host {} port
 {}'.format(gg_core.thing_arn, connectivity_info.host_address,
 connectivity_info.port))
                    mqtt_connection = mqtt_connection_builder.mtls_from_path(
                        endpoint=connectivity_info.host_address,
                        port=connectivity_info.port,
                        cert_filepath=cmdUtils.get_command_required("cert"),
                        pri_key_filepath=cmdUtils.get_command_required("key"),

 ca_bytes=gg_group.certificate_authorities[0].encode('utf-8'),
                        on_connection_interrupted=on_connection_interupted,
                        on_connection_resumed=on_connection_resumed,
                        client_id=cmdUtils.get_command_required("thing_name"),
                        clean_session=False,
                        keep_alive_secs=30)

                    connect_future = mqtt_connection.connect()
                    connect_future.result()
                    print('Connected!')
                    return mqtt_connection

                except Exception as e:
                    print('Connection failed with exception {}'.format(e))
                    continue

    exit('All connection attempts failed')

# Function for gracefully quitting this sample
def exit(msg_or_exception):
```

```
    if isinstance(msg_or_exception, Exception):
        print("Exiting sample due to exception.")
        traceback.print_exception(msg_or_exception.__class__, msg_or_exception,
 sys.exc_info()[2])
    else:
        print("Exiting sample:", msg_or_exception)

    with locked_data.lock:
        if not locked_data.disconnect_called:
            print("Disconnecting...")
            locked_data.disconnect_called = True
            future = mqtt_connection.disconnect()
            future.add_done_callback(on_disconnected)

def on_disconnected(disconnect_future):
    # type: (Future) -> None
    print("Disconnected.")

    # Signal that sample is finished
    is_sample_done.set()

def on_get_shadow_accepted(response):
    # type: (iotshadow.GetShadowResponse) -> None
    try:
        with locked_data.lock:
            # check that this is a response to a request from this session
            try:
                locked_data.request_tokens.remove(response.client_token)
            except KeyError:
                return

            print("Finished getting initial shadow state.")
            if locked_data.shadow_value is not None:
                print("  Ignoring initial query because a delta event has
 already been received.")
                return

        if response.state:
            if response.state.delta:
                value = response.state.delta.get(shadow_property)
                if value:
                    print("  Shadow contains delta value '{}'.".format(value))
                    change_shadow_value(value)
                    return
```

```
            if response.state.reported:
                value = response.state.reported.get(shadow_property)
                if value:
                    print("  Shadow contains reported value
 '{}'.".format(value))

 set_local_value_due_to_initial_query(response.state.reported[shadow_property])
                    return

        print("  Shadow document lacks '{}' property. Setting
 defaults...".format(shadow_property))
        change_shadow_value(SHADOW_VALUE_DEFAULT)
        return

    except Exception as e:
        exit(e)

def on_get_shadow_rejected(error):
    # type: (iotshadow.ErrorResponse) -> None
    try:
        # check that this is a response to a request from this session
        with locked_data.lock:
            try:
                locked_data.request_tokens.remove(error.client_token)
            except KeyError:
                return

        if error.code == 404:
            print("Thing has no shadow document. Creating with defaults...")
            change_shadow_value(SHADOW_VALUE_DEFAULT)
        else:
            exit("Get request was rejected. code:{} message:'{}'".format(
                error.code, error.message))

    except Exception as e:
        exit(e)

def on_shadow_delta_updated(delta):
    # type: (iotshadow.ShadowDeltaUpdatedEvent) -> None
    try:
        print("Received shadow delta event.")
        if delta.state and (shadow_property in delta.state):
            value = delta.state[shadow_property]
```

```
            if value is None:
                print("  Delta reports that '{}' was deleted. Resetting
 defaults...".format(shadow_property))
                change_shadow_value(SHADOW_VALUE_DEFAULT)
                return
            else:
                print("  Delta reports that desired value is '{}'. Changing
 local value...".format(value))
                if (delta.client_token is not None):
                    print ("  ClientToken is: " + delta.client_token)
                change_shadow_value(value, delta.client_token)
        else:
            print("  Delta did not report a change in
 '{}'".format(shadow_property))

    except Exception as e:
        exit(e)

def on_publish_update_shadow(future):
    #type: (Future) -> None
    try:
        future.result()
        print("Update request published.")
    except Exception as e:
        print("Failed to publish update request.")
        exit(e)

def on_update_shadow_accepted(response):
    # type: (iotshadow.UpdateShadowResponse) -> None
    try:
        # check that this is a response to a request from this session
        with locked_data.lock:
            try:
                locked_data.request_tokens.remove(response.client_token)
            except KeyError:
                return

        try:
            if response.state.reported != None:
                if shadow_property in response.state.reported:
                    print("Finished updating reported shadow value to
 '{}'.".format(response.state.reported[shadow_property])) # type: ignore
                else:
```

```
                    print ("Could not find shadow property with name:
 '{}'.".format(shadow_property)) # type: ignore
            else:
                print("Shadow states cleared.") # when the shadow states are
 cleared, reported and desired are set to None
        except:
            exit("Updated shadow is missing the target property")

    except Exception as e:
        exit(e)

def on_update_shadow_rejected(error):
    # type: (iotshadow.ErrorResponse) -> None
    try:
        # check that this is a response to a request from this session
        with locked_data.lock:
            try:
                locked_data.request_tokens.remove(error.client_token)
            except KeyError:
                return

        exit("Update request was rejected. code:{} message:'{}'".format(
            error.code, error.message))

    except Exception as e:
        exit(e)

def set_local_value_due_to_initial_query(reported_value):
    with locked_data.lock:
        locked_data.shadow_value = reported_value

def change_shadow_value(value, token=None):
    with locked_data.lock:
        if locked_data.shadow_value == value:
            print("Local value is already '{}'.".format(value))
            return

        print("Changed local shadow value to '{}'.".format(value))
        locked_data.shadow_value = value

        print("Updating reported shadow value to '{}'...".format(value))

        reuse_token = token is not None
        # use a unique token so we can correlate this "request" message to
```

```
        # any "response" messages received on the /accepted and /rejected
 topics
        if not reuse_token:
            token = str(uuid4())

        # if the value is "clear shadow" then send a UpdateShadowRequest with
 None
        # for both reported and desired to clear the shadow document
 completely.
        if value == "clear_shadow":
            tmp_state = iotshadow.ShadowState(reported=None, desired=None,
 reported_is_nullable=True, desired_is_nullable=True)
            request = iotshadow.UpdateShadowRequest(
                thing_name=shadow_thing_name,
                state=tmp_state,
                client_token=token,
            )
        # Otherwise, send a normal update request
        else:
            # if the value is "none" then set it to a Python none object to
            # clear the individual shadow property
            if value == "none":
                value = None

            request = iotshadow.UpdateShadowRequest(
            thing_name=shadow_thing_name,
            state=iotshadow.ShadowState(
                reported={ shadow_property: value }
                ),
                client_token=token,
            )

        future = shadow_client.publish_update_shadow(request,
 mqtt.QoS.AT_LEAST_ONCE)

        if not reuse_token:
            locked_data.request_tokens.add(token)

        future.add_done_callback(on_publish_update_shadow)


if __name__ == '__main__':
```

```
    tls_options =
io.TlsContextOptions.create_client_with_mtls_from_path(cmdUtils.get_command_required("
cmdUtils.get_command_required("key"))
    if cmdUtils.get_command(cmdUtils.m_cmd_ca_file):
        tls_options.override_default_trust_store_from_path(None,
cmdUtils.get_command(cmdUtils.m_cmd_ca_file))
    tls_context = io.ClientTlsContext(tls_options)

    socket_options = io.SocketOptions()

    print('Performing greengrass discovery...')
    discovery_client =
DiscoveryClient(io.ClientBootstrap.get_or_create_static_default(),
socket_options, tls_context, cmdUtils.get_command_required("region"))
    resp_future =
discovery_client.discover(cmdUtils.get_command_required("thing_name"))
    discover_response = resp_future.result()

    print(discover_response)
    if cmdUtils.get_command("print_discover_resp_only"):
        exit(0)

    mqtt_connection = try_iot_endpoints()
    shadow_client = iotshadow.IotShadowClient(mqtt_connection)

    try:
        # Subscribe to necessary topics.
        # Note that is **is** important to wait for "accepted/rejected"
subscriptions
        # to succeed before publishing the corresponding "request".
        print("Subscribing to Update responses...")
        update_accepted_subscribed_future, _ =
shadow_client.subscribe_to_update_shadow_accepted(

request=iotshadow.UpdateShadowSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_update_shadow_accepted)

        update_rejected_subscribed_future, _ =
shadow_client.subscribe_to_update_shadow_rejected(

request=iotshadow.UpdateShadowSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_update_shadow_rejected)
```

```
        # Wait for subscriptions to succeed
        update_accepted_subscribed_future.result()
        update_rejected_subscribed_future.result()

        print("Subscribing to Get responses...")
        get_accepted_subscribed_future, _ =
shadow_client.subscribe_to_get_shadow_accepted(

request=iotshadow.GetShadowSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_get_shadow_accepted)

        get_rejected_subscribed_future, _ =
shadow_client.subscribe_to_get_shadow_rejected(

request=iotshadow.GetShadowSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_get_shadow_rejected)

        # Wait for subscriptions to succeed
        get_accepted_subscribed_future.result()
        get_rejected_subscribed_future.result()

        print("Subscribing to Delta events...")
        delta_subscribed_future, _ =
shadow_client.subscribe_to_shadow_delta_updated_events(

request=iotshadow.ShadowDeltaUpdatedSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_shadow_delta_updated)

        # Wait for subscription to succeed
        delta_subscribed_future.result()

        # The rest of the sample runs asynchronously.

        # Issue request for shadow's current state.
        # The response will be received by the on_get_accepted() callback
        print("Requesting current shadow state...")

        with locked_data.lock:
            # use a unique token so we can correlate this "request" message to
```

```
            # any "response" messages received on the /accepted and /rejected
topics
            token = str(uuid4())

            publish_get_future = shadow_client.publish_get_shadow(

request=iotshadow.GetShadowRequest(thing_name=shadow_thing_name,
client_token=token),
                qos=mqtt.QoS.AT_LEAST_ONCE)

            locked_data.request_tokens.add(token)

        # Ensure that publish succeeds
        publish_get_future.result()

    except Exception as e:
        exit(e)

    # Wait for the sample to finish (user types 'quit', or an error occurs)
    is_sample_done.wait()
```

Diese Python-Anwendung führt Folgendes aus:

- Verwendet Greengrass Discovery, um das Core-Gerät zu erkennen und eine Verbindung mit ihm herzustellen.
- Fordert das Schattendokument vom Core-Gerät an, den Anfangsstatus der Eigenschaft abzurufen.
- Abonniert Shadow-Delta-Ereignisse, die das Core-Gerät sendet, wenn sich der `desired` Wert der Eigenschaft von seinem `reported` Wert unterscheidet. Wenn die Anwendung ein Schattendelta-Ereignis empfängt, ändert sie den Wert der -Eigenschaft und sendet eine Aktualisierung an das Core-Gerät, um den neuen Wert als `reported` Wert festzulegen.

Diese Anwendung kombiniert die Greengrass-Erkennungs- und Schattenbeispiele aus vAWS IoT Device SDK2.

c. Führen Sie die Beispielanwendung aus. Diese Anwendung erwartet Argumente, die den Objektnamen des Client-Geräts, die zu verwendende Schatteneigenschaft und die Zertifikate angeben, die die Verbindung authentifizieren und sichern.

- Ersetzen Sie *MyClientDevice1* durch den Objektnamen des Client-Geräts.
- Ersetzen Sie *~/certs/AmazonRootCA1.pem* durch den Pfad zum Amazon-Root-CA-Zertifikat auf dem Client-Gerät.
- Ersetzen Sie *~/certs/device.pem.crt* durch den Pfad zum Gerätezertifikat auf dem Client-Gerät.
- Ersetzen Sie *~/certs/private.pem.key* durch den Pfad zur Datei mit dem privaten Schlüssel auf dem Client-Gerät.
- Ersetzen Sie *us-east-1* durch die AWS Region, in der Ihr Client-Gerät und Ihr Core-Gerät betrieben werden.

```
python3 basic_discovery_shadow.py \
  --thing_name MyClientDevice1 \
  --shadow_property color \
  --ca_file ~/certs/AmazonRootCA1.pem \
  --cert ~/certs/device.pem.crt \
  --key ~/certs/private.pem.key \
  --region us-east-1 \
  --verbosity Warn
```

Die Beispielanwendung abonniert die Schattenthemen und wartet darauf, Schattendelta-Ereignisse vom Core-Gerät zu empfangen. Wenn die Ausgabe angibt, dass die Anwendung Schattendelta-Ereignisse empfängt und darauf reagiert, kann das Client-Gerät erfolgreich mit seinem Schatten auf dem Core-Gerät interagieren.

```
Performing greengrass discovery...
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGG
coreDevice-MyGreengrassCore',
 cores=[awsiot.greengrass_discovery.GGCore(thing_arn='arn:aws:iot:us-
east-1:123456789012:thing/MyGreengrassCore',
 connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='203.0.113.0',
 host_address='203.0.113.0', metadata='', port=8883)])],
 certificate_authorities=['-----BEGIN CERTIFICATE-----
\nMIICiT...EXAMPLE=\n-----END CERTIFICATE-----\n'])])
Trying core arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore at host
 203.0.113.0 port 8883
Connected!
Subscribing to Update responses...
Subscribing to Get responses...
```

```
Subscribing to Delta events...
Requesting current shadow state...
Received shadow delta event.
  Delta reports that desired value is 'purple'. Changing local value...
  ClientToken is: 3dce4d3f-e336-41ac-aa4f-7882725f0033
Changed local shadow value to 'purple'.
Updating reported shadow value to 'purple'...
Update request published.
```

Wenn die Anwendung stattdessen einen Fehler ausgibt, finden Sie weitere Informationen unter Fehlerbehebung bei Greengrass-Erkennungsproblemen.

Sie können auch die Greengrass-Protokolle auf dem Core-Gerät anzeigen, um zu überprüfen, ob das Client-Gerät erfolgreich eine Verbindung herstellt und Nachrichten sendet. Weitere Informationen finden Sie unter Überwachen von AWS IoT Greengrass Protokollen.

5. Zeigen Sie die Komponentenprotokolle erneut an, um zu überprüfen, ob die Komponente Bestätigungen zur Schattenaktualisierung vom Smart Light Client-Gerät erhält.

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/
   com.example.clientdevices.MySmartLightManager.log
   ```

   PowerShell

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

   Die Komponente protokolliert Nachrichten, um zu bestätigen, dass das Smart Light Client-Gerät seine Farbe geändert hat.

   ```
   2022-07-07T03:49:24.908Z [INFO] (Copier)
    com.example.clientdevices.MySmartLightManager: stdout. Chose random color (blue)
    for MyClientDevice1.
    {scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
    serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
   2022-07-07T03:49:24.912Z [INFO] (Copier)
    com.example.clientdevices.MySmartLightManager: stdout.
   ```

```
 Requested color change for MyClientDevice1 to blue.
 {scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
 serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
2022-07-07T03:49:24.959Z [INFO] (Copier)
 com.example.clientdevices.MySmartLightManager: stdout. Received
 shadow update confirmation from client device: MyClientDevice1.
 {scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
 serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
```

Note

Der Schatten des Client-Geräts ist zwischen dem Core-Gerät und dem Client-Gerät synchronisiert. Das Core-Gerät synchronisiert den Schatten des Client-Geräts jedoch nicht mit AWS IoT Core. Sie können einen Schatten mit synchronisierenAWS IoT Core, um beispielsweise den Status aller Geräte in Ihrer Flotte anzuzeigen oder zu ändern. Weitere Informationen zum Konfigurieren der Shadow-Manager-Komponente zum Synchronisieren von Schatten mit AWS IoT Corefinden Sie unter Lokale Geräteschatten mit synchronisieren AWS IoT Core.

Sie haben dieses Tutorial abgeschlossen. Das Client-Gerät stellt eine Verbindung zum Core-Gerät her, sendet MQTT-Nachrichten an - AWS IoT Core und Greengrass-Komponenten und empfängt Schattenaktualisierungen vom Core-Gerät. Weitere Informationen zu den in diesem Tutorial behandelten Themen finden Sie unter:

- Zuordnen von Client-Geräten
- Verwalten von -Core-Geräteendpunkten
- Testen der Kommunikation von Client-Geräten
- RESTful-API zur Greengrass-Erkennung
- Weiterleiten von MQTT-Nachrichten zwischen Client-Geräten und AWS IoT Core
- Interagieren mit Client-Geräten in Komponenten
- Interagieren mit Geräteschatten
- Interagieren und Synchronisieren von Client-Geräteschatten

Tutorial: Erste Schritte mit SageMaker Edge Manager

> **Important**
>
> SageMaker Edge Manager wird am 26. April 2024 eingestellt. Weitere Informationen zur weiteren Bereitstellung Ihrer Modelle auf Edge-Geräten finden Sie unter Ende der Nutzungsdauer von SageMaker Edge Manager.

Amazon SageMaker Edge Manager ist ein Softwareagent, der auf Edge-Geräten ausgeführt wird. SageMaker Edge Manager bietet Modellverwaltung für Edge-Geräte, sodass Sie mit Amazon SageMaker Neo kompilierte Modelle direkt auf Greengrass-Core-Geräten verpacken und verwenden können. Mithilfe von SageMaker Edge Manager können Sie auch Modelleingabe- und -ausgabedaten von Ihren Kerngeräten abfragen und diese Daten AWS Cloud zur Überwachung und Analyse an die senden. Weitere Informationen zur Funktionsweise von SageMaker Edge Manager auf Greengrass-Core-Geräten finden Sie unterVerwenden von Amazon SageMaker Edge Manager auf Greengrass-Core-Geräten.

In diesem Tutorial erfahren Sie, wie Sie mit der Verwendung von SageMaker Edge Manager beginnen können. Die AWS bereitgestellten Beispielkomponenten werden auf einem vorhandenen Core-Gerät verwendet. Diese Beispielkomponenten verwenden die SageMaker Edge Manager-Komponente als Abhängigkeit, um den Edge Manager-Agenten bereitzustellen und Inferenzen mithilfe vortrainierter Modelle durchzuführen, die mit Neo kompiliert wurden. SageMaker Weitere Informationen zum SageMaker Edge Manager-Agenten finden Sie unter SageMaker Edge Manager im Amazon SageMaker Developer Guide.

Um den SageMaker Edge Manager-Agenten auf einem vorhandenen Greengrass-Core-Gerät einzurichten und zu verwenden, AWS enthält Beispielcode, mit dem Sie die folgenden Beispiel-Inferenz- und Modellkomponenten erstellen können.

- Klassifizierung von Bildern
  - `com.greengrass.SageMakerEdgeManager.ImageClassification`
  - `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`
- Objekterkennung
  - `com.greengrass.SageMakerEdgeManager.ObjectDetection`
  - `com.greengrass.SageMakerEdgeManager.ObjectDetection.Model`

In diesem Tutorial erfahren Sie, wie Sie die Beispielkomponenten und den SageMaker Edge Manager-Agenten bereitstellen.

Themen

Voraussetzungen

Um dieses Tutorial abzuschließen, müssen Sie die folgenden Voraussetzungen erfüllen:

- Ein Greengrass-Core-Gerät, das auf Amazon Linux 2, einer Debian-basierten Linux-Plattform (x86_64 oder Armv8) oder Windows (x86_64) läuft. Falls Sie noch keines haben, beachten Sie die Informationen unter Tutorial: Erste Schritte mit AWS IoT Greengrass V2.
- Python 3.6 oder höher, auch `pip` für Ihre Version von Python, die auf Ihrem Kerngerät installiert ist.
- Die OpenGL-API GLX Runtime (`libgl1-mesa-glx`) ist auf Ihrem Core-Gerät installiert.
- Ein AWS Identity and Access Management (IAM) -Benutzer mit Administratorrechten.
- Ein internetfähiger Windows-, Mac- oder UNIX-ähnlicher Entwicklungscomputer, der die folgenden Anforderungen erfüllt:
  - Python 3.6 oder höher installiert.
  - AWS CLI installiert und mit Ihren IAM-Administrator-Benutzeranmeldedaten konfiguriert. Weitere Informationen finden Sie unter Installation AWS CLI und Konfiguration von. AWS CLI
- Die folgenden S3-Buckets wurden in demselben AWS-Konto und AWS-Region wie Ihr Greengrass-Core-Gerät erstellt:
  - Ein S3-Bucket zum Speichern der Artefakte, die in den Inferenz- und Modellkomponenten der Stichprobe enthalten sind. In diesem Tutorial wird DOC-EXAMPLE-BUCKET1 verwendet, um auf diesen Bucket zu verweisen.
  - Ein S3-Bucket, den Sie Ihrer Edge-Geräteflotte zuordnen. SageMaker SageMaker Edge Manager benötigt einen S3-Bucket, um die Edge-Geräteflotte zu erstellen und Beispieldaten aus laufenden Inferenzen auf Ihrem Gerät zu speichern. In diesem Tutorial wird DOC-EXAMPLE-BUCKET2 verwendet, um auf diesen Bucket zu verweisen.

  Informationen zum Erstellen von S3-Buckets finden Sie unter Erste Schritte mit Amazon S3.

- Die Greengrass-Geräterolle wurde wie folgt konfiguriert:
  - Eine Vertrauensbeziehung, die es ermöglicht `credentials.iot.amazonaws.com` und `sagemaker.amazonaws.com` die Übernahme der Rolle ermöglicht, wie im folgenden Beispiel für eine IAM-Richtlinie dargestellt.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
       },
      "Action": "sts:AssumeRole"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sagemaker.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

  - Die von AmazonSageMakerEdgeDeviceFleetPolicyIAM verwaltete Richtlinie.
  - Die von AmazonSageMakerFullAccessIAM verwaltete Richtlinie.
  - Die `s3:GetObject` Aktion für den S3-Bucket, der Ihre Komponentenartefakte enthält, wie im folgenden Beispiel für eine IAM-Richtlinie dargestellt.

```
{
  "Version": "2012-10-17",
  "Statement": [
  {
    "Action": [
      "s3:GetObject"
    ],
    "Resource": [
      "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/*"
    ],
    "Effect": "Allow"
```

```
  }
  ]
}
```

Richten Sie Ihr Greengrass Core-Gerät in SageMaker Edge Manager ein

Edge-Geräteflotten in SageMaker Edge Manager sind Sammlungen logisch gruppierter Geräte. Um SageMaker Edge Manager mit zu verwenden AWS IoT Greengrass, müssen Sie eine Edge-Geräteflotte erstellen, die denselben AWS IoT Rollenalias wie das Greengrass-Core-Gerät verwendet, auf dem Sie den SageMaker Edge Manager-Agent bereitstellen. Anschließend müssen Sie das Core-Gerät als Teil dieser Flotte registrieren.

Themen

- Erstellen Sie eine Flotte von Edge-Geräten
- Registrieren Sie Ihr Greengrass Core-Gerät

Erstellen Sie eine Flotte von Edge-Geräten

Um eine Edge-Geräteflotte (Konsole) zu erstellen

1. Wählen Sie in der SageMaker Amazon-Konsole Edge Manager und dann Edge-Geräteflotten aus.
2. Wählen Sie auf der Seite Geräteflotten die Option Geräteflotte erstellen aus.
3. Gehen Sie unter Eigenschaften der Geräteflotte wie folgt vor:
   - Geben Sie unter Name der Geräteflotte einen Namen für Ihre Geräteflotte ein.
   - Geben Sie für die IAM-Rolle den Amazon-Ressourcennamen (ARN) des AWS IoT Rollenalias ein, den Sie bei der Einrichtung Ihres Greengrass-Core-Geräts angegeben haben.
   - Deaktivieren Sie den Schalter IAM-Rollenalias erstellen.
4. Wählen Sie Weiter aus.
5. Geben Sie unter Ausgabekonfiguration für S3-Bucket-URI den URI des S3-Buckets ein, den Sie der Geräteflotte zuordnen möchten.
6. Wählen Sie Absenden aus.

Registrieren Sie Ihr Greengrass Core-Gerät

Um Ihr Greengrass Core-Gerät als Edge-Gerät (Konsole) zu registrieren

1. Wählen Sie in der SageMaker Amazon-Konsole Edge Manager und dann Edge-Geräte aus.
2. Wählen Sie auf der Seite Geräte die Option Geräte registrieren aus.
3. Geben Sie unter Geräteeigenschaften für Name der Geräteflotte den Namen der Geräteflotte ein, die Sie erstellt haben, und wählen Sie dann Weiter aus.
4. Wählen Sie Weiter aus.
5. Geben Sie unter Gerätequelle für Gerätename den AWS IoT Dingnamen Ihres Greengrass-Core-Geräts ein.
6. Wählen Sie Absenden aus.

Erstellen Sie die Beispielkomponenten

Um Ihnen den Einstieg in die Verwendung der SageMaker Edge Manager-Komponente zu erleichtern, AWS stellt es ein Python-Skript bereit GitHub , das die Beispielinferenz- und Modellkomponenten erstellt und sie AWS Cloud für Sie in die hochlädt. Führen Sie die folgenden Schritte auf einem Entwicklungscomputer aus.

Um die Beispielkomponenten zu erstellen

1. Laden Sie das Repository mit den AWS IoT Greengrass Komponentenbeispielen GitHub auf Ihren Entwicklungscomputer herunter.
2. Navigieren Sie zum heruntergeladenen `/machine-learning/sagemaker-edge-manager` Ordner.

   ```
   cd download-directory/machine-learning/sagemaker-edge-manager
   ```
3. Führen Sie den folgenden Befehl aus, um die Beispielkomponenten zu erstellen und in den hochzuladen AWS Cloud.

   ```
   python3 create_components.py -r region -b DOC-EXAMPLE-BUCKET
   ```

   Ersetzen Sie *`Region`* durch den Ort, AWS-Region an dem Sie Ihr Greengrass-Core-Gerät erstellt haben, und ersetzen Sie DOC-EXAMPLE-BUCKET1 durch den Namen des S3-Buckets, um Ihre Komponentenartefakte zu speichern.

> **Note**
>
> Standardmäßig erstellt das Skript Beispielkomponenten sowohl für die Bildklassifizierung als auch für die Inferenz zur Objekterkennung. Um Komponenten nur für einen bestimmten Inferenztyp zu erstellen, geben Sie das `-i ImageClassification | ObjectDetection` Argument an.

Beispielinferenz- und Modellkomponenten für die Verwendung mit SageMaker Edge Manager werden jetzt in Ihrem erstellt. AWS-Konto Um die Beispielkomponenten in der AWS IoT Greengrass Konsole anzuzeigen, wählen Sie Komponenten und suchen Sie dann unter Meine Komponenten nach den folgenden Komponenten:

- `com.greengrass.SageMakerEdgeManager.ImageClassification`
- `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`
- `com.greengrass.SageMakerEdgeManager.ObjectDetection`
- `com.greengrass.SageMakerEdgeManager.ObjectDetection.Model`

Führen Sie ein Beispiel für die Inferenz zur Bildklassifizierung aus

Um die Inferenz zur Bildklassifizierung mithilfe der AWS bereitgestellten Beispielkomponenten und des SageMaker Edge Manager-Agenten auszuführen, müssen Sie diese Komponenten auf Ihrem Kerngerät bereitstellen. Durch die Bereitstellung dieser Komponenten wird ein SageMaker NEO-kompiliertes, vortrainiertes Resnet-50-Modell heruntergeladen und der SageMaker Edge Manager-Agent auf Ihrem Gerät installiert. Der SageMaker Edge Manager-Agent lädt das Modell und veröffentlicht Inferenzergebnisse zu diesem Thema. `gg/sageMakerEdgeManager/image-classification` Um diese Inferenzergebnisse anzuzeigen, verwenden Sie den AWS IoT MQTT-Client in der AWS IoT Konsole, um dieses Thema zu abonnieren.

Themen

- Abonnieren Sie das Thema Benachrichtigungen
- Stellen Sie die Beispielkomponenten bereit
- Inferenzergebnisse anzeigen

Abonnieren Sie das Thema Benachrichtigungen

In diesem Schritt konfigurieren Sie den AWS IoT MQTT-Client in der AWS IoT Konsole so, dass er MQTT-Nachrichten beobachtet, die von der Beispiel-Inferenzkomponente veröffentlicht wurden. Standardmäßig veröffentlicht die Komponente Inferenzergebnisse zu diesem Thema. `gg/sageMakerEdgeManager/image-classification` Abonnieren Sie dieses Thema, bevor Sie die Komponente auf Ihrem Greengrass-Core-Gerät bereitstellen, um die Inferenzergebnisse zu sehen, wenn die Komponente zum ersten Mal ausgeführt wird.

Um das Thema Standardbenachrichtigungen zu abonnieren

1. Wählen Sie im Navigationsmenü der AWS IoT Konsole Test, MQTT-Testclient aus.
2. Geben **`gg/sageMakerEdgeManager/image-classification`**Sie unter Thema abonnieren in das Feld Themenname den Text ein.
3. Wählen Sie Subscribe (Abonnieren) aus.

Stellen Sie die Beispielkomponenten bereit

In diesem Schritt konfigurieren und implementieren Sie die folgenden Komponenten auf Ihrem Kerngerät:

- `aws.greengrass.SageMakerEdgeManager`
- `com.greengrass.SageMakerEdgeManager.ImageClassification`
- `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`

Für den Einsatz Ihrer Komponenten (Konsole)

1. Wählen Sie im Navigationsmenü der AWS IoT Greengrass Konsole Bereitstellungen und dann die Bereitstellung für Ihr Zielgerät aus, die Sie überarbeiten möchten.
2. Wählen Sie auf der Seite „Bereitstellung“ die Option „Überarbeiten“ und anschließend „Bereitstellung überarbeiten“ aus.
3. Wählen Sie auf der Seite „Ziel angeben“ die Option Weiter aus.
4. Gehen Sie auf der Seite „Komponenten auswählen“ wie folgt vor:

    a. Wählen Sie unter Meine Komponenten die folgenden Komponenten aus:

- `com.greengrass.SageMakerEdgeManager.ImageClassification`
- `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`

b. Deaktivieren Sie unter Öffentliche Komponenten die Option Nur ausgewählte Komponenten anzeigen, und wählen Sie dann die `aws.greengrass.SageMakerEdgeManager` Komponente aus.

c. Wählen Sie Weiter aus.

5. Wählen Sie auf der Seite Komponenten konfigurieren die `aws.greengrass.SageMakerEdgeManager` Komponente aus und gehen Sie wie folgt vor.

a. Wählen Sie Komponente konfigurieren aus.

b. Geben Sie unter Konfigurationsupdate unter Zusammenzuführende Konfiguration die folgende Konfiguration ein.

```
{
    "DeviceFleetName": "device-fleet-name",
    "BucketName": "DOC-EXAMPLE-BUCKET"
}
```

*`device-fleet-name`*Ersetzen Sie es durch den Namen der Edge-Geräteflotte, die Sie erstellt haben, und ersetzen Sie *`DOC-EXAMPLE-BUCKET`* durch den Namen des S3-Buckets, der Ihrer Geräteflotte zugeordnet ist.

c. Wählen Sie Bestätigen aus, und wählen Sie dann Weiter.

6. Behalten Sie auf der Seite Erweiterte Einstellungen konfigurieren die Standardkonfigurationseinstellungen bei und wählen Sie Weiter.

7. Wählen Sie auf der Seite „Überprüfen“ die Option Bereitstellen

Zur Bereitstellung Ihrer Komponenten (AWS CLI)

1. Erstellen Sie auf Ihrem Entwicklungscomputer eine `deployment.json` Datei, um die Bereitstellungskonfiguration für Ihre SageMaker Edge Manager-Komponenten zu definieren. Diese Datei sollte wie im folgenden Beispiel aussehen.

```
{
  "targetArn":"targetArn",
  "components": {
    "aws.greengrass.SageMakerEdgeManager": {
```

```
      "componentVersion": "1.0.x",
      "configurationUpdate": {
        "merge": "{\"DeviceFleetName\":\"device-fleet-name\",\"BucketName\":\"DOC-
EXAMPLE-BUCKET2\"}"
      }
    },
    "com.greengrass.SageMakerEdgeManager.ImageClassification": {
      "componentVersion": "1.0.x",
      "configurationUpdate": {
      }
    },
    "com.greengrass.SageMakerEdgeManager.ImageClassification.Model": {
      "componentVersion": "1.0.x",
      "configurationUpdate": {
      }
    },
  }
}
```

- Ersetzen Sie im `targetArn` Feld *`targetArn`* durch den Amazon-Ressourcennamen (ARN) des Objekts oder der Objektgruppe, auf die die Bereitstellung ausgerichtet werden soll, und zwar im folgenden Format:
  - Objekt: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`thingName`*
  - Objektgruppe: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thinggroup/`*`thingGroupName`*
- Ersetzen Sie das `merge` Feld *`device-fleet-name`*durch den Namen der Edge-Geräteflotte, die Sie erstellt haben. Ersetzen Sie dann *`DOC-EXAMPLE-BUCKET2`* durch den Namen des S3-Buckets, der Ihrer Geräteflotte zugeordnet ist.
- Ersetzen Sie die Versionen aller Komponenten durch die neueste verfügbare Version.

2. Führen Sie den folgenden Befehl aus, um die Komponenten auf dem Gerät bereitzustellen:

```
aws greengrassv2 create-deployment \
    --cli-input-json file://path/to/deployment.json
```

Es kann einige Minuten dauern, bis die Bereitstellung abgeschlossen ist. Überprüfen Sie im nächsten Schritt im Komponentenprotokoll, ob die Bereitstellung erfolgreich abgeschlossen wurde, und schauen Sie sich die Inference-Ergebnisse an.

Inferenzergebnisse anzeigen

Nachdem Sie die Komponenten bereitgestellt haben, können Sie die Inferenzergebnisse im Komponentenprotokoll auf Ihrem Greengrass-Core-Gerät und im AWS IoT MQTT-Client in der Konsole einsehen. AWS IoT Informationen zum Abonnieren des Themas, zu dem die Komponente Inferenzergebnisse veröffentlicht, finden Sie unter. Abonnieren Sie das Thema Benachrichtigungen

- AWS IoT MQTT-Client — Gehen Sie wie folgt vor, um die Ergebnisse anzuzeigen, die die Inferenzkomponente zum Thema Standardbenachrichtigungen veröffentlicht:

  1. Wählen Sie im Navigationsmenü der AWS IoT Konsole Test, MQTT-Testclient aus.
  2. Wählen Sie unter Abonnements die Option**`gg/sageMakerEdgeManager/image-classification`**.

- Komponentenprotokoll — Um die Inferenzergebnisse im Komponentenprotokoll anzuzeigen, führen Sie den folgenden Befehl auf Ihrem Greengrass-Core-Gerät aus.

```
sudo tail -f /greengrass/v2/logs/
com.greengrass.SageMakerEdgeManager.ImageClassification.log
```

Wenn Sie keine Inferenzergebnisse im Komponentenprotokoll oder im MQTT-Client sehen können, ist die Bereitstellung fehlgeschlagen oder hat das Kerngerät nicht erreicht. Dies kann der Fall sein, wenn Ihr Kerngerät nicht mit dem Internet verbunden ist oder nicht über die richtigen Berechtigungen zum Ausführen der Komponente verfügt. Führen Sie den folgenden Befehl auf Ihrem Core-Gerät aus, um die AWS IoT Greengrass Core-Software-Protokolldatei anzuzeigen. Diese Datei enthält Protokolle vom Bereitstellungsdienst des Greengrass-Core-Geräts.

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

Weitere Informationen finden Sie unter Fehlerbehebung bei Machine Learning-Inferenzen.

Tutorial: Durchführen einer Inferenz bei der Bildklassifizierung mit TensorFlow Lite

Dieses Tutorial zeigt Ihnen, wie Sie die Inferenzkomponente der TensorFlow Lite-Bildklassifizierung verwenden, um eine Beispiel-Inferenz der Bildklassifizierung auf einem Greengrass-Kerngerät durchzuführen. Diese Komponente umfasst die folgenden Komponentenabhängigkeiten:

- TensorFlow Lite-Bildklassifizierungsmodellspeicherkomponente
- TensorFlow Lite-Laufzeitkomponente

Wenn Sie diese Komponente bereitstellen, lädt sie ein vortrainiertes MobileNet v1-Modell herunter und installiert die TensorFlow Lite-Laufzeit und ihre Abhängigkeiten. Diese Komponente veröffentlicht Inferenzergebnisse zum `ml/tflite/image-classification` Thema. Um diese Inferenzergebnisse anzuzeigen, verwenden Sie den AWS IoT MQTT-Client in der AWS IoT Konsole, um dieses Thema zu abonnieren.

In diesem Tutorial stellen Sie die Beispielinferenzkomponente bereit, um die Bildklassifizierung für das von bereitgestellte Beispielbild durchzuführenAWS IoT Greengrass. Nachdem Sie dieses Tutorial abgeschlossen haben, können Sie abschließen, das Ihnen zeigtTutorial: Durchführen einer Beispielbildklassifizierungsinferenz für Bilder von einer Kamera mit TensorFlow Lite, wie Sie die Beispiel-Inferenzkomponente ändern, um die Bildklassifizierung auf Bildern von einer Kamera lokal auf einem Greengrass-Kerngerät durchzuführen.

Weitere Informationen zum Machine Learning auf Greengrass-Geräten finden Sie unter Durchführen von Machine Learning-Inferenzen.

Themen

Voraussetzungen

Zum Durcharbeiten dieses Tutorials ist Folgendes erforderlich:

- Ein Linux-Greengrass-Core-Gerät. Falls Sie noch keines haben, beachten Sie die Informationen unter Tutorial: Erste Schritte mit AWS IoT Greengrass V2. Das Core-Gerät muss die folgenden Anforderungen erfüllen:
  - Auf Greengrass-Core-Geräten, auf denen Amazon Linux 2 oder Ubuntu 18.04 ausgeführt wird, ist GNU C Library (glibc) Version 2.27 oder höher auf dem Gerät installiert.
  - Auf Armv7l-Geräten wie Raspberry Pi sind Abhängigkeiten für OpenCV-Python auf dem Gerät installiert. Führen Sie den folgenden Befehl aus, um die Abhängigkeiten zu installieren.

    ```
    sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
     libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
    ```
  - Raspberry Pi-Geräte, auf denen Raspberry Pi OS Bullseye ausgeführt wird, müssen die folgenden Anforderungen erfüllen:
    - NumPy 1.22.4 oder höher ist auf dem Gerät installiert. Raspberry Pi OS Bullseye enthält eine frühere Version von NumPy, sodass Sie den folgenden Befehl ausführen können, um ein Upgrade NumPy auf dem Gerät durchzuführen.

      ```
      pip3 install --upgrade numpy
      ```
    - Der auf dem Gerät aktivierte Legacy-Kamera-Stack. Raspberry Pi OS Bullseye enthält einen neuen Kamera-Stack, der standardmäßig aktiviert und nicht kompatibel ist, daher müssen Sie den Legacy-Kamera-Stack aktivieren.

      So aktivieren Sie den Legacy-Kamera-Stack

      1. Führen Sie den folgenden Befehl aus, um das Raspberry Pi-Konfigurationstool zu öffnen.

         ```
         sudo raspi-config
         ```
      2. Wählen Sie Schnittstellenoptionen aus.
      3. Wählen Sie Legacy-Kamera aus, um den Legacy-Kamera-Stack zu aktivieren.
      4. Starten Sie den Raspberry Pi neu.

Schritt 1: Abonnieren des Themas mit Standardbenachrichtigungen

In diesem Schritt konfigurieren Sie den AWS IoT MQTT-Client in der AWS IoT Konsole so, dass er von der TensorFlow Lite-Bildklassifizierungskomponente veröffentlichte MQTT-Nachrichten überwacht. Standardmäßig veröffentlicht die Komponente Inferenzergebnisse zum `ml/tflite/`

`image-classification` Thema. Abonnieren Sie dieses Thema, bevor Sie die Komponente auf Ihrem Greengrass-Kerngerät bereitstellen, um die Inferenzergebnisse zu sehen, wenn die Komponente zum ersten Mal ausgeführt wird.

So abonnieren Sie das Standard-Benachrichtigungsthema

1. Wählen Sie im Navigationsmenü der AWS IoT Konsole Test, MQTT-Testclient aus.
2. Geben Sie unter Thema abonnieren im Feld Themenname ein**`ml/tflite/image-classification`**.
3. Wählen Sie Subscribe (Abonnieren) aus.

Schritt 2: Bereitstellen der TensorFlow Lite-Bildklassifizierungskomponente

In diesem Schritt stellen Sie die TensorFlow Lite-Bildklassifizierungskomponente auf Ihrem Core-Gerät bereit:

So stellen Sie die TensorFlow Lite-Bildklassifizierungskomponente bereit (Konsole)

1. Wählen Sie im Navigationsmenü der AWS IoT GreengrassKonsole Komponenten aus.
2. Wählen Sie auf der Seite Komponenten auf der Registerkarte Öffentliche Komponenten die Option `aws.greengrass.TensorFlowLiteImageClassification` aus.
3. Wählen Sie auf der aws.greengrass.TensorFlowLiteImageClassification Seite Bereitstellen aus.
4. Wählen Sie unter Zu Bereitstellung hinzufügen eine der folgenden Optionen aus:

   a. Um diese Komponente mit einer auf Ihrem Zielgerät vorhandenen Bereitstellung zusammenzuführen, wählen Sie Zu vorhandener Bereitstellung hinzufügen und wählen Sie dann die Bereitstellung aus, die Sie überarbeiten möchten.

   b. Um auf Ihrem Zielgerät eine neue Bereitstellung zu erstellen, wählen Sie Neue Bereitstellung erstellen aus. Wenn auf Ihrem Gerät bereits eine Bereitstellung vorhanden ist, ersetzt die Auswahl in diesem Schritt die vorhandene Bereitstellung.
5. Gehen Sie auf der Seite Ziel angeben wie folgt vor:

   a. Geben Sie unter Bereitstellungsinformationen den Anzeigenamen für Ihre Bereitstellung ein oder ändern Sie ihn.

b. Wählen Sie unter Bereitstellungsziele ein Ziel für Ihre Bereitstellung aus und klicken Sie auf Weiter. Wenn Sie eine vorhandene Bereitstellung überarbeiten, können Sie das Bereitstellungsziel nicht ändern.

6. Überprüfen Sie auf der Seite Komponenten auswählen unter Öffentliche Komponenten, ob die `aws.greengrass.TensorFlowLiteImageClassification` Komponente ausgewählt ist, und wählen Sie Weiter aus.
7. Behalten Sie auf der Seite Komponenten konfigurieren die Standardkonfigurationseinstellungen bei und wählen Sie Weiter aus.
8. Behalten Sie auf der Seite Erweiterte Einstellungen konfigurieren die Standardkonfigurationseinstellungen bei und wählen Sie Weiter.
9. Wählen Sie auf der Seite Review die Option Deploy aus.

So stellen Sie die TensorFlow Lite-Bildklassifizierungskomponente bereit (AWS CLI)

1. Erstellen Sie eine -`deployment.json`Datei, um die Bereitstellungskonfiguration für die TensorFlow Lite-Image-Klassifizierungskomponente zu definieren. Diese Datei sollte wie folgt aussehen:

```
{
  "targetArn":"targetArn",
  "components": {
    "aws.greengrass.TensorFlowLiteImageClassification": {
      "componentVersion": 2.1.0,
      "configurationUpdate": {
      }
    }
  }
}
```

   - Ersetzen Sie im `targetArn` Feld *`targetArn`* durch den Amazon-Ressourcennamen (ARN) des Objekts oder der Objektgruppe, auf die die Bereitstellung ausgerichtet werden soll, und zwar im folgenden Format:
     - Objekt: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`thingName`*
     - Objektgruppe: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thinggroup/`*`thingGroupName`*

- In diesem Tutorial wird die Komponentenversion 2.1.0 verwendet. Ersetzen Sie im `aws.greengrass.TensorFlowLiteObjectDetection` Komponentenobjekt *`2.1.0`*, um eine andere Version der TensorFlow Lite-Objekterkennungskomponente zu verwenden.

2. Führen Sie den folgenden Befehl aus, um die TensorFlow Lite-Bildklassifizierungskomponente auf dem Gerät bereitzustellen:

```
aws greengrassv2 create-deployment \
    --cli-input-json file://path/to/deployment.json
```

Es kann einige Minuten dauern, bis die Bereitstellung abgeschlossen ist. Überprüfen Sie im nächsten Schritt im Komponentenprotokoll, ob die Bereitstellung erfolgreich abgeschlossen wurde, und schauen Sie sich die Inference-Ergebnisse an.

Schritt 3: Anzeigen von Inferenzergebnissen

Nachdem Sie die Komponente bereitgestellt haben, können Sie die Inferenzergebnisse im Komponentenprotokoll auf Ihrem Greengrass-Kerngerät und im AWS IoT MQTT-Client in der AWS IoT Konsole anzeigen. Informationen zum Abonnieren des Themas, zu dem die Komponente Inferenzergebnisse veröffentlicht, finden Sie unter Schritt 1: Abonnieren des Themas mit Standardbenachrichtigungen.

- AWS IoT MQTT-Client – Führen Sie die folgenden Schritte aus, um die Ergebnisse anzuzeigen, die die Inferenzkomponente im Standardbenachrichtigungsthema veröffentlicht:

  1. Wählen Sie im Navigationsmenü der AWS IoT Konsole Test, MQTT-Testclient aus.
  2. Wählen Sie unter Abonnements die Option aus**`ml/tflite/image-classification`**.

     Sie sollten Meldungen ähnlich dem folgenden Beispiel sehen.

```
{
  "timestamp": "2021-01-01 00:00:00.000000",
  "inference-type": "image-classification",
  "inference-description": "Top 5 predictions with score 0.3 or above ",
  "inference-results": [
    {
      "Label": "cougar, puma, catamount, mountain lion, painter, panther, Felis
 concolor",
      "Score": "0.5882352941176471"
    },
```

```
    {
      "Label": "Persian cat",
      "Score": "0.5882352941176471"
    },
    {
      "Label": "tiger cat",
      "Score": "0.5882352941176471"
    },
    {
      "Label": "dalmatian, coach dog, carriage dog",
      "Score": "0.5607843137254902"
    },
    {
      "Label": "malamute, malemute, Alaskan malamute",
      "Score": "0.5450980392156862"
    }
  ]
}
```

- Komponentenprotokoll – Um die Inferenzergebnisse im Komponentenprotokoll anzuzeigen, führen Sie den folgenden Befehl auf Ihrem Greengrass-Kerngerät aus.

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

  Sie sollten Ergebnisse ähnlich dem folgenden Beispiel sehen.

  ```
  2021-01-01 00:00:00.000000 [INFO] (Copier)
   aws.greengrass.TensorFlowLiteImageClassification: stdout. Publishing results to the
   IoT core....
   {scriptName=services.aws.greengrass.TensorFlowLiteImageClassification.lifecycle.Run.script,
   serviceName=aws.greengrass.TensorFlowLiteImageClassification, currentState=RUNNING}

  2021-01-01 00:00:00.000000 [INFO] (Copier)
   aws.greengrass.TensorFlowLiteImageClassification: stdout. {"timestamp":
   "2021-01-01 00:00:00.000000", "inference-type": "image-classification", "inference-
  description": "Top 5 predictions with score 0.3 or above ", "inference-results":
   [{"Label": "cougar, puma, catamount, mountain lion, painter, panther, Felis
   concolor", "Score": "0.5882352941176471"}, {"Label": "Persian cat", "Score":
   "0.5882352941176471"}, {"Label": "tiger cat", "Score": "0.5882352941176471"},
   {"Label": "dalmatian, coach dog, carriage dog", "Score": "0.5607843137254902"},
   {"Label": "malamute, malemute, Alaskan malamute", "Score": "0.5450980392156862"}]}.
  ```

```
{scriptName=services.aws.greengrass.TensorFlowLiteImageClassification.lifecycle.Run.script,
serviceName=aws.greengrass.TensorFlowLiteImageClassification, currentState=RUNNING}
```

Wenn Sie keine Inferenzergebnisse im Komponentenprotokoll oder im MQTT-Client sehen können, ist die Bereitstellung fehlgeschlagen oder hat das Core-Gerät nicht erreicht. Dies kann passieren, wenn Ihr Core-Gerät nicht mit dem Internet verbunden ist oder nicht über die richtigen Berechtigungen zum Ausführen der Komponente verfügt. Führen Sie den folgenden Befehl auf Ihrem Core-Gerät aus, um die AWS IoT Greengrass Core-Softwareprotokolldatei anzuzeigen. Diese Datei enthält Protokolle aus dem Bereitstellungsservice des Greengrass-Core-Geräts.

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

Weitere Informationen finden Sie unter Fehlerbehebung bei Machine Learning-Inferenzen.

Nächste Schritte

Wenn Sie über ein Greengrass-Core-Gerät mit einer unterstützten Kameraschnittstelle verfügen, können Sie vervollständigen, das Ihnen zeigtTutorial: Durchführen einer Beispielbildklassifizierungsinferenz für Bilder von einer Kamera mit TensorFlow Lite, wie Sie die Beispiel-Inferenzkomponente ändern, um eine Bildklassifizierung für Bilder von einer Kamera durchzuführen.

Um die Konfiguration der Beispiel-Inferenzkomponente für die TensorFlow Lite-Bildklassifizierung weiter zu untersuchen, versuchen Sie Folgendes:

- Ändern Sie den `InferenceInterval` Konfigurationsparameter, um zu ändern, wie oft der Inferenzcode ausgeführt wird.
- Ändern Sie die `ImageDirectory` Konfigurationsparameter `ImageName` und in der Konfiguration der Inferenzkomponente, um ein benutzerdefiniertes Image anzugeben, das für die Inferenz verwendet werden soll.

Informationen zum Anpassen der Konfiguration öffentlicher Komponenten oder zum Erstellen benutzerdefinierter Machine-Learning-Komponenten finden Sie unter Anpassen Ihrer Machine-Learning-Komponenten.

Tutorial: Durchführen einer Beispielbildklassifizierungsinferenz für Bilder von einer Kamera mit TensorFlow Lite

Dieses Tutorial zeigt Ihnen, wie Sie die Inferenzkomponente der TensorFlow Lite-Bildklassifizierung verwenden, um eine Beispiel-Bildklassifizierungsinferenz auf Bildern von einer Kamera lokal auf einem Greengrass-Kerngerät durchzuführen. Diese Komponente umfasst die folgenden Komponentenabhängigkeiten:

- TensorFlow Lite-Bildklassifizierungsmodellspeicherkomponente
- TensorFlow Lite-Laufzeitkomponente

> **Note**
>
> Dieses Tutorial greift auf das Kameramodul für Raspberry Pi- oder NVIDIA Jetson Nano-Geräte zu, AWS IoT Greengrass unterstützt jedoch andere Geräte auf Armv7l, Armv8- oder x86_64-Plattformen. Informationen zum Einrichten einer Kamera für ein anderes Gerät finden Sie in der entsprechenden Dokumentation für Ihr Gerät.

Weitere Informationen zum Machine Learning auf Greengrass-Geräten finden Sie unter Durchführen von Machine Learning-Inferenzen.

Themen

Voraussetzungen

Um dieses Tutorial abzuschließen, müssen Sie zuerst abschließenTutorial: Durchführen einer Inferenz bei der Bildklassifizierung mit TensorFlow Lite.

Sie benötigen außerdem Folgendes:

- Ein Linux-Greengrass-Core-Gerät mit einer Kameraschnittstelle. Dieses Tutorial greift auf das Kameramodul auf einem der folgenden unterstützten Geräte zu:
  - Raspberry Pi mit Raspberry Pi OS (früher Raspbian genannt)
  - NVIDIA Jetson Nano

  Informationen zum Einrichten eines Greengrass-Core-Geräts finden Sie unter Tutorial: Erste Schritte mit AWS IoT Greengrass V2.

  Das Core-Gerät muss die folgenden Anforderungen erfüllen:

  - Auf Greengrass-Core-Geräten, auf denen Amazon Linux 2 oder Ubuntu 18.04 ausgeführt wird, ist GNU C Library (glibc) Version 2.27 oder höher auf dem Gerät installiert.
  - Auf Armv7l-Geräten wie Raspberry Pi sind Abhängigkeiten für OpenCV-Python auf dem Gerät installiert. Führen Sie den folgenden Befehl aus, um die Abhängigkeiten zu installieren.

    ```
    sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
     libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
    ```
  - Raspberry Pi-Geräte, auf denen Raspberry Pi OS Bullseye ausgeführt wird, müssen die folgenden Anforderungen erfüllen:
    - NumPy 1.22.4 oder höher ist auf dem Gerät installiert. Raspberry Pi OS Bullseye enthält eine frühere Version von NumPy, sodass Sie den folgenden Befehl ausführen können, um ein Upgrade NumPy auf dem Gerät durchzuführen.

      ```
      pip3 install --upgrade numpy
      ```
    - Der auf dem Gerät aktivierte Legacy-Kamera-Stack. Raspberry Pi OS Bullseye enthält einen neuen Kamera-Stack, der standardmäßig aktiviert und nicht kompatibel ist, daher müssen Sie den Legacy-Kamera-Stack aktivieren.

      So aktivieren Sie den Legacy-Kamera-Stack

      1. Führen Sie den folgenden Befehl aus, um das Raspberry Pi-Konfigurationstool zu öffnen.

         ```
         sudo raspi-config
         ```
      2. Wählen Sie Schnittstellenoptionen aus.
      3. Wählen Sie Legacy-Kamera aus, um den Legacy-Kamera-Stack zu aktivieren.

4. Starten Sie den Raspberry Pi neu.

- Für Raspberry Pi- oder NVIDIA Jetson Nano-Geräte Raspberry Pi Camera Module V2 – 8 Megapixel, 1080p. Weitere Informationen zum Einrichten der Kamera finden Sie unter Verbinden der Kamera in der Raspberry Pi-Dokumentation.

Schritt 1: Konfigurieren des Kameramoduls auf Ihrem Gerät

In diesem Schritt installieren und aktivieren Sie das Kameramodul für Ihr Gerät. Führen Sie die folgenden Befehle auf dem Gerät aus.

Raspberry Pi (Armv7l)

1. Installieren Sie die `picamera` Schnittstelle für das Kameramodul. Führen Sie den folgenden Befehl aus, um das Kameramodul und die anderen Python-Bibliotheken zu installieren, die für dieses Tutorial erforderlich sind.

   ```
   sudo apt-get install -y python3-picamera
   ```

2. Überprüfen Sie, ob Picamera erfolgreich installiert wurde.

   ```
   sudo -u ggc_user bash -c 'python3 -c "import picamera"'
   ```

   Wenn die Ausgabe keine Fehler enthält, ist die Überprüfung erfolgreich.

   > **Note**
   >
   > Wenn die ausführbare Python-Datei, die auf Ihrem Gerät installiert ist, ist`python3.7`, 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 das Gerät 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. Führen Sie den folgenden Befehl aus, um die Kameraeinrichtung 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.
7. Führen Sie den folgenden Befehl aus, um einen Symlink zu erstellen, damit die Inferenzkomponente aus der virtuellen Umgebung, die von der Laufzeitkomponente erstellt wird, auf Ihre Kamera zugreifen kann.

   ```
   sudo ln -s /usr/lib/python3/dist-packages/picamera "MLRootPath/
   greengrass_ml_tflite_venv/lib/python3.7/site-packages"
   ```

   Der Standardwert für *MLRootPath* für dieses Tutorial ist */greengrass/v2*`/work/variant.TensorFlowLite/greengrass_ml`. Der `greengrass_ml_tflite_venv` Ordner an diesem Speicherort wird erstellt, wenn Sie die Inferenzkomponente zum ersten Mal in bereitstellenTutorial: Durchführen einer Inferenz bei der Bildklassifizierung mit TensorFlow Lite.

Jetson Nano (Armv8)

1. Führen Sie den folgenden Befehl aus, um die Kameraeinrichtung zu testen.

   ```
   gst-launch-1.0 nvarguscamerasrc num-buffers=1 ! "video/x-raw(memory:NVMM),
    width=1920, height=1080, format=NV12, framerate=30/1" ! nvjpegenc ! filesink
    location=test.jpg
   ```

   Dadurch wird ein Image mit dem Namen erfasst und `test.jpg` in Ihrem aktuellen Verzeichnis gespeichert.
2. (Optional) Starten Sie das Gerät neu. Wenn beim Ausführen des `gst-launch` Befehls im vorherigen Schritt Probleme auftreten, kann ein Neustart Ihres Geräts diese Probleme möglicherweise beheben.

```
sudo reboot
```

> **Note**
>
> Für Armv8 (AArch64)-Geräte, wie z. B. Jetson Nano, müssen Sie keinen Symlink erstellen, damit die Inferenzkomponente aus der virtuellen Umgebung, die von der Laufzeitkomponente erstellt wird, auf die Kamera zugreifen kann.

Schritt 2: Überprüfen Ihres Abonnements für das Standardbenachrichtigungsthema

In haben Sie den AWS IoT MQTT-Client in der AWS IoTKonsole so konfiguriertTutorial: Durchführen einer Inferenz bei der Bildklassifizierung mit TensorFlow Lite, dass er MQTT-Nachrichten überwacht, die von der TensorFlow Lite-Bildklassifizierungskomponente zum `ml/tflite/image-classification` Thema veröffentlicht wurden. Überprüfen Sie in der -AWS IoTKonsole, ob dieses Abonnement vorhanden ist. Andernfalls führen Sie die Schritte unter aus, Schritt 1: Abonnieren des Themas mit Standardbenachrichtigungen um dieses Thema zu abonnieren, bevor Sie die Komponente auf Ihrem Greengrass-Kerngerät bereitstellen.

Schritt 3: Ändern der Konfiguration der TensorFlow Lite-Image-Klassifizierungskomponente und Bereitstellen

In diesem Schritt konfigurieren Sie die TensorFlow Lite-Bildklassifizierungskomponente und stellen sie auf Ihrem Core-Gerät bereit:

So konfigurieren und stellen Sie die TensorFlow Lite-Image-Klassifizierungskomponente bereit (Konsole)

1. Wählen Sie im Navigationsmenü der AWS IoT GreengrassKonsole Komponenten aus.
2. Wählen Sie auf der Seite Komponenten auf der Registerkarte Öffentliche Komponenten die Option `aws.greengrass.TensorFlowLiteImageClassification` aus.
3. Wählen Sie auf der aws.greengrass.TensorFlowLiteImageClassification Seite Bereitstellen aus.
4. Wählen Sie unter Zur Bereitstellung hinzufügen eine der folgenden Optionen aus:

a. Um diese Komponente mit einer auf Ihrem Zielgerät vorhandenen Bereitstellung zusammenzuführen, wählen Sie Zu vorhandener Bereitstellung hinzufügen und wählen Sie dann die Bereitstellung aus, die Sie überarbeiten möchten.

b. Um auf Ihrem Zielgerät eine neue Bereitstellung zu erstellen, wählen Sie Neue Bereitstellung erstellen aus. Wenn auf Ihrem Gerät bereits eine Bereitstellung vorhanden ist, ersetzt die Auswahl in diesem Schritt die vorhandene Bereitstellung.

5. Gehen Sie auf der Seite Ziel angeben wie folgt vor:

   a. Geben Sie unter Bereitstellungsinformationen den Anzeigenamen für Ihre Bereitstellung ein oder ändern Sie ihn.

   b. Wählen Sie unter Bereitstellungsziele ein Ziel für Ihre Bereitstellung aus und klicken Sie auf Weiter. Wenn Sie eine vorhandene Bereitstellung überarbeiten, können Sie das Bereitstellungsziel nicht ändern.

6. Überprüfen Sie auf der Seite Komponenten auswählen unter Öffentliche Komponenten, ob die `aws.greengrass.TensorFlowLiteImageClassification` Komponente ausgewählt ist, und wählen Sie Weiter aus.

7. Gehen Sie auf der Seite Komponenten konfigurieren wie folgt vor:

   a. Wählen Sie die Inferenzkomponente und dann Komponente konfigurieren aus.

   b. Geben Sie unter Konfigurationsaktualisierung das folgende Konfigurationsupdate in das Feld Zusammenzuführende Konfiguration ein.

   ```
   {
     "InferenceInterval": "60",
     "UseCamera": "true"
   }
   ```

   Mit diesem Konfigurationsupdate greift die Komponente auf das Kameramodul auf Ihrem Gerät zu und führt Inferenzen auf Bildern durch, die von der Kamera aufgenommen wurden. Der Inferenzcode wird alle 60 Sekunden ausgeführt.

   c. Wählen Sie Bestätigen aus, und wählen Sie dann Weiter.

8. Behalten Sie auf der Seite Erweiterte Einstellungen konfigurieren die Standardkonfigurationseinstellungen bei und wählen Sie Weiter.

9. Wählen Sie auf der Seite Überprüfen die Option Bereitstellen aus.

So konfigurieren und stellen Sie die TensorFlow Lite-Image-Klassifizierungskomponente (AWS CLI) bereit

1. Erstellen Sie eine -`deployment.json`Datei, um die Bereitstellungskonfiguration für die TensorFlow Lite-Image-Klassifizierungskomponente zu definieren. Diese Datei sollte wie folgt aussehen:

   ```
   {
     "targetArn":"targetArn",
     "components": {
       "aws.greengrass.TensorFlowLiteImageClassification": {
         "componentVersion": 2.1.0,
         "configurationUpdate": {
           "InferenceInterval": "60",
           "UseCamera": "true"
         }
       }
     }
   }
   ```

   - Ersetzen Sie im `targetArn` Feld *`targetArn`* durch den Amazon-Ressourcennamen (ARN) des Objekts oder der Objektgruppe, auf die die Bereitstellung ausgerichtet werden soll, und zwar im folgenden Format:
     - Objekt: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`thingName`*
     - Objektgruppe: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thinggroup/`*`thingGroupName`*
   - In diesem Tutorial wird die Komponentenversion 2.1.0 verwendet. Ersetzen Sie im `aws.greengrass.TensorFlowLiteImageClassification` Komponentenobjekt *`2.1.0`*, um eine andere Version der TensorFlow Lite-Bildklassifizierungskomponente zu verwenden.

   Mit diesem Konfigurationsupdate greift die Komponente auf das Kameramodul auf Ihrem Gerät zu und führt Inferenzen auf Bildern durch, die von der Kamera aufgenommen wurden. Der Inferenzcode wird alle 60 Sekunden ausgeführt. Ersetzen Sie die folgenden Werte

2. Führen Sie den folgenden Befehl aus, um die TensorFlow Lite-Bildklassifizierungskomponente auf dem Gerät bereitzustellen:

   ```
   aws greengrassv2 create-deployment \
       --cli-input-json file://path/to/deployment.json
   ```

Es kann einige Minuten dauern, bis die Bereitstellung abgeschlossen ist. Überprüfen Sie im nächsten Schritt im Komponentenprotokoll, ob die Bereitstellung erfolgreich abgeschlossen wurde, und schauen Sie sich die Inference-Ergebnisse an.

Schritt 4: Anzeigen von Inferenzergebnissen

Nachdem Sie die Komponente bereitgestellt haben, können Sie die Inferenzergebnisse im Komponentenprotokoll auf Ihrem Greengrass-Kerngerät und im AWS IoT MQTT-Client in der AWS IoT Konsole anzeigen. Informationen zum Abonnieren des Themas, zu dem die Komponente Inferenzergebnisse veröffentlicht, finden Sie unter Schritt 2: Überprüfen Ihres Abonnements für das Standardbenachrichtigungsthema.

- AWS IoT MQTT-Client – Führen Sie die folgenden Schritte aus, um die Ergebnisse anzuzeigen, die die Inferenzkomponente im Standardbenachrichtigungsthema veröffentlicht:

  1. Wählen Sie im Navigationsmenü der AWS IoT Konsole Test, MQTT-Testclient aus.
  2. Wählen Sie unter Abonnements die Option aus**`ml/tflite/image-classification`**.

- Komponentenprotokoll – Um die Inferenzergebnisse im Komponentenprotokoll anzuzeigen, führen Sie den folgenden Befehl auf Ihrem Greengrass-Kerngerät aus.

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

Wenn Sie keine Inferenzergebnisse im Komponentenprotokoll oder im MQTT-Client sehen können, ist die Bereitstellung fehlgeschlagen oder hat das Core-Gerät nicht erreicht. Dies kann passieren, wenn Ihr Core-Gerät nicht mit dem Internet verbunden ist oder nicht über die erforderlichen Berechtigungen zum Ausführen der Komponente verfügt. Führen Sie den folgenden Befehl auf Ihrem Core-Gerät aus, um die AWS IoT Greengrass Core-Softwareprotokolldatei anzuzeigen. Diese Datei enthält Protokolle aus dem Bereitstellungsservice des Greengrass-Core-Geräts.

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

Weitere Informationen finden Sie unter Fehlerbehebung bei Machine Learning-Inferenzen.

Nächste Schritte

In diesem Tutorial erfahren Sie, wie Sie die TensorFlow Lite-Bildklassifizierungskomponente mit benutzerdefinierten Konfigurationsoptionen verwenden, um eine Beispielbildklassifizierung für von einer Kamera aufgenommene Bilder durchzuführen.

Weitere Informationen zum Anpassen der Konfiguration öffentlicher Komponenten oder zum Erstellen benutzerdefinierter Machine-Learning-Komponenten finden Sie unter Anpassen Ihrer Machine-Learning-Komponenten.

Komponenten

AWS IoT Greengrass -Komponenten sind Softwaremodule, die Sie auf Greengrass-Core-Geräten bereitstellen. Komponenten können Anwendungen, Laufzeitinstallationsprogramme, Bibliotheken oder Code darstellen, den Sie auf einem Gerät ausführen würden. Sie können Komponenten definieren, die von anderen Komponenten abhängen. Sie können beispielsweise eine Komponente definieren, die Python installiert, und diese Komponente dann als Abhängigkeit Ihrer Komponenten definieren, die Python-Anwendungen ausführen. Wenn Sie Ihre Komponenten auf Ihren Geräteflotten bereitstellen, stellt Greengrass nur die Softwaremodule bereit, die Ihre Geräte benötigen.

Themen

AWSVon bereitgestellte Komponenten

AWS IoT Greengrass bietet und verwaltet vorgefertigte Komponenten, die Sie auf Ihren Geräten bereitstellen können. Zu diesen Komponenten gehören Funktionen (wie Stream Manager), AWS IoT Greengrass V1-Konnektoren (wie CloudWatch Metriken) und lokale Entwicklungstools (wie die AWS IoT Greengrass -CLI). Sie können diese Komponenten auf Ihren Geräten für ihre eigenständige Funktionalität bereitstellen oder sie als Abhängigkeiten in Ihren benutzerdefinierten Greengrass-Komponenten verwenden.

> **Note**
>
> Mehrere AWSvon bereitgestellte Komponenten hängen von bestimmten Nebenversionen des Greengrass-Kerns ab. Aufgrund dieser Abhängigkeit müssen Sie diese Komponenten aktualisieren, wenn Sie den Greengrass-Kern auf eine neue Nebenversion aktualisieren. Informationen zu den spezifischen Versionen des Kerns, von denen jede Komponente abhängt, finden Sie im entsprechenden Komponententhema. Weitere Informationen zum

Aktualisieren des Kerns finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Wenn eine Komponente sowohl den Komponententyp generisch als auch Lambda hat, ist die aktuelle Version der Komponente der generische Typ und eine frühere Version der Komponente der Lambda-Typ.

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
Grüngraskern	Der Kern der Core- AWS IoT Greengrass Software. Verwenden Sie diese Komponente, um die Software auf Ihren -Core-Geräten zu konfigurieren und zu aktualisieren.	Kabel	Linux, Windows	Ja
Authentifizierung auf Client-Geräten	Ermöglicht lokalen IoT-Geräten, so genannte Client-Geräte, eine Verbindung zum Core-Gerät herzustellen.	Plug-In	Linux, Windows	Ja

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
CloudWatch -Metriken	Veröffentlicht benutzerdefinierte Metriken in Amazon CloudWatch.	Generisch, Lambda	Linux, Windows	Ja
AWS IoT Device Defender	Benachrichtigt Administratoren über Änderungen des Zustands des Greengrass-Kerngeräts, um ungewöhnliches Verhalten zu identifizieren.	Generisch, Lambda	Linux, Windows	Ja

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
Festplatten-Spooler	Aktiviert eine persistente Speicheroption für Nachrichten, die von Greengrass-Core-Geräten nach gepoolt werden AWS IoT Core. Diese Komponente speichert diese ausgehenden Nachrichten auf der Festplatte.	Plug-In	Linux, Windows	Ja

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
Docker-Anwendungsmanager	Ermöglicht AWS IoT Greengrass das Herunterladen von Docker-Images von Docker Hub und Amazon Elastic Container Registry (Amazon ECR).	Generisch	Linux, Windows	Nein
Edge-Anschluss für Kinesis Video Streams	Liest Video-Feeds von lokalen Kameras, veröffentlicht die Streams in Kinesis Video Streams und zeigt die Streams in Grafana-Dashboards mit an AWS IoT TwinMaker.	Generisch	Linux	Nein

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
Greengrass CLI	Bietet eine Befehlszeilenschnittstelle, mit der Sie lokale Bereitstellungen erstellen und mit dem Greengrass-Kerngerät und seinen Komponenten interagieren können.	Plug-In	Linux, Windows	Ja
IP-Detektor	Meldet MQTT-Broker-Konnektivitätsinformationen an AWS IoT Greengrass, sodass Client-Geräte erkennen können, wie eine Verbindung hergestellt werden soll.	Plug-In	Linux, Windows	Ja

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
Firehose	Veröffentlicht Daten über Amazon-Data-Firehose-Bereitstellungsdatenströme an Ziele im AWS Cloud.	Lambda	Linux	Nein
Lambda-Launcher	Bearbeitet Prozesse und Umgebungskonfigurationen für Lambda-Funktionen.	Generisch	Linux	Nein
Lambda-Manager	Bewältigt die Kommunikation und Skalierung zwischen Prozessen für Lambda-Funktionen.	Plug-In	Linux	Nein
Lambda-Laufzeiten	Stellt Artefakte für jede Lambda-Laufzeit bereit.	Generisch	Linux	Nein

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
Legacy-Abonnement-Router	Verwaltet Abonnements für Lambda-Funktionen, die auf AWS IoT Greengrass V1 ausgeführt werden.	Generisch	Linux	Nein
Lokale Debug-Konsole	Bietet eine lokale Konsole, mit der Sie das Greengrass-Kernger ät und seine Komponenten debuggen und verwalten können.	Plug-In	Linux, Windows	Ja
Protokollmanager	Sammelt und lädt Protokolle auf das Greengrass-Core-Gerät hoch.	Plug-In	Linux, Windows	Ja

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
Machine-Learning-Komponenten	Stellt Machine-Learning-Modelle und Beispiel-Inferenzcode bereit, mit dem Sie Machine-Learning-Inferenzen auf Greengrass-Core-Geräten durchführen können.	Siehe Machine-Learning-Komponenten.		
Adapter für das Bol-RTU-Protokoll	Abfragen von Informationen von lokalen RTU-Geräten.	Lambda	Linux	Nein
Telemetrie-Emitter	Veröffentlicht die vom Kern gesammelten Telemetriedaten des Systemzustands in einem lokalen Thema oder in einem AWS IoT Core MQTT-Thema.	Plug-In	Linux, Windows	Ja

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
MQTT-Brücke	Weiterleitet MQTT-Nachrichten zwischen Client-Geräten, lokaler AWS IoT Greengrass Veröffentlichung/Abonnement und AWS IoT Core.	Plug-In	Linux, Windows	Ja
MQTT 3.1.1-Broker (Moquette)	Führt einen MQTT-3.1.1-Broker aus, der Nachrichten zwischen Client-Geräten und dem Core-Gerät verarbeitet.	Plug-In	Linux, Windows	Ja

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
MQTT 5-Broker (EMAX)	Führt einen MQTT 5-Broker aus, der Nachrichten zwischen Client-Geräten und dem Core-Gerät verarbeitet.	Generisch	Linux, Windows	Nein
PKCS #11-Anbieter	Ermöglicht Greengrass-Komponenten den Zugriff auf einen privaten Schlüssel und ein Zertifikat, die Sie sicher in einem Hardware-Sicherheitsmodul (HSM) speichern.	Plug-In	Linux	Ja

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
Geheimer Manager	Stellt Secrets aus AWS Secrets Manager Secrets bereit, damit Sie Anmeldeinformationen wie Passwörter sicher in benutzerdefinierten Komponenten auf dem Greengrass-Core-Gerät verwenden können.	Plug-In	Linux, Windows	Ja

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
Sicheres Tunneling	Ermöglicht AWS IoT sichere Tunneling-Verbindungen, mit denen Sie eine Bidirectional-Kommunikation mit Greengrass-Core-Geräten herstellen können, die sich hinter eingeschränkten Firewalls befinden.	Generisch	Linux	Nein

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
Schattenmanager	Ermöglicht die Interaktion mit Schatten auf dem Core-Gerät. Es verwaltet die Speicherung von Schattendokumenten sowie die Synchronisation lokaler Schattenzustände mit dem AWS IoT Gerätesch atten-Service.	Plug-In	Linux, Windows	Ja
Amazon SNS	Veröffentlicht Nachrichten zu Amazon SNS-Themen.	Lambda	Linux	Nein
Stream-Manager	Streamt Daten mit hohem Volumen aus lokalen Quellen in die AWS Cloud.	Generisch	Linux, Windows	Nein

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
Systemmanager-Agent	Verwalten Sie das Core-Gerät mit AWS Systems Manager, mit dem Sie Geräte patchen, Befehle ausführen und vieles mehr können.	Generisch	Linux	Nein
Token-Exchange-Service	Stellt AWS Anmeldeinformationen bereit, mit denen Sie mit AWS -Services interagieren können.	Generisch	Linux, Windows	Nein
IoT SiteWise -OPC-UA-Kollektor	Sammelt Daten von OPC-UA-Servern.	Generisch	Linux, Windows	Nein

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
IoT SiteWise OPC-UA-Datenquellensimulator	Führt einen lokalen OPC-UA-Server aus, der Beispieldaten generiert.	Generisch	Linux, Windows	Nein
SiteWise IoT-Herausgeber	Veröffentlicht Daten in der AWS Cloud.	Generisch	Linux, Windows	Nein
SiteWise IoT-Prozessor	Verarbeitet Daten auf den Greengrass-Core-Geräten.	Generisch	Linux, Windows	Nein

Grüngraskern

Die Greengrass Nucleus-Komponente (`aws.greengrass.Nucleus`) ist eine obligatorische Komponente und die Mindestanforderung, um die AWS IoT Greengrass Core-Software auf einem Gerät auszuführen. Sie können diese Komponente so konfigurieren, dass Ihre AWS IoT Greengrass Core-Software per Fernzugriff angepasst und aktualisiert wird. Stellen Sie diese Komponente bereit, um Einstellungen wie Proxy, Geräterolle und AWS IoT Dingkonfiguration auf Ihren Kerngeräten zu konfigurieren.

> **Important**
>
> Wenn sich die Version der Nucleus-Komponente ändert oder wenn Sie bestimmte Konfigurationsparameter ändern, wird die AWS IoT Greengrass Core-Software — zu der der Nucleus und alle anderen Komponenten auf Ihrem Gerät gehören — neu gestartet, um die Änderungen zu übernehmen.

Wenn Sie eine Komponente bereitstellen, werden die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente AWS IoT Greengrass installiert. Aus diesem Grund werden neue Patch-Versionen von AWS bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren Kerngeräten bereitgestellt, wenn Sie einer Dinggruppe neue Geräte hinzufügen oder wenn Sie die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Nucleus-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir, dass Sie Ihre bevorzugte Version dieser Komponente direkt angeben, wenn Sie eine Bereitstellung erstellen. Weitere Informationen zum Aktualisierungsverhalten der AWS IoT Greengrass Core-Software finden Sie unterAktualisieren der AWS IoT Greengrass Core-Software (OTA).

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.12.x
- 2.11.x
- 2.10.x
- 2.9.x
- 2.8.x
- 2.7.x

- 2.6.x
- 2.5.x
- 2.4.x
- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Betriebssystem

Diese Komponente kann auf Kerngeräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Weitere Informationen finden Sie unter Unterstützte Plattformen.

Voraussetzungen

Geräte müssen bestimmte Anforderungen erfüllen, um den Greengrass Nucleus und die AWS IoT Greengrass Core-Software installieren und ausführen zu können. Weitere Informationen finden Sie unter Anforderungen an Speichergeräte.

Die Greengrass Nucleus-Komponente wird für die Ausführung in einer VPC unterstützt. Um diese Komponente in einer VPC bereitzustellen, ist Folgendes erforderlich.

- Die Greengrass Nucleus-Komponente muss über Konnektivität zu AWS IoT data, AWS IoT Credentials und Amazon S3 verfügen.

Abhängigkeiten

Der Greengrass-Kern beinhaltet keine Komponentenabhängigkeiten. Einige der von AWS-bereitgestellten Komponenten beinhalten jedoch den Nukleus als Abhängigkeit. Weitere Informationen finden Sie unter AWSVon bereitgestellte Komponenten.

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Rezeptreferenz für Komponenten.

Herunterladen und Installation

Sie können ein Installationsprogramm herunterladen, das die Greengrass Nucleus-Komponente auf Ihrem Gerät einrichtet. Dieses Installationsprogramm richtet Ihr Gerät als Greengrass-Core-Gerät ein. Es gibt zwei Arten von Installationen, die Sie durchführen können: eine Schnellinstallation, bei der die erforderlichen AWS Ressourcen für Sie erstellt werden, oder eine manuelle Installation, bei der Sie die AWS Ressourcen selbst erstellen. Weitere Informationen finden Sie unter Installieren Sie die AWS IoT Greengrass Core-Software..

Sie können auch einem Tutorial folgen, um den Greengrass-Kern zu installieren und sich mit der Entwicklung von Greengrass-Komponenten vertraut zu machen. Weitere Informationen finden Sie unter Tutorial: Erste Schritte mit AWS IoT Greengrass V2.

Konfiguration

Diese Komponente bietet die folgenden Konfigurationsparameter, die Sie bei der Bereitstellung der Komponente anpassen können. Einige Parameter setzen voraus, dass die AWS IoT Greengrass Core-Software neu gestartet wird, um wirksam zu werden. Weitere Informationen darüber, warum und wie diese Komponente konfiguriert wird, finden Sie unterKonfigurieren Sie die AWS IoT Greengrass Core-Software.

`iotRoleAlias`

Der AWS IoT Rollenalias, der auf eine Token-Exchange-IAM-Rolle verweist. Der Anbieter AWS IoT für Anmeldeinformationen übernimmt diese Rolle, damit das Greengrass-Core-Gerät mit AWS Diensten interagieren kann. Weitere Informationen finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

Wenn Sie die AWS IoT Greengrass Core-Software mit dieser `--provision true` Option ausführen, stellt die Software einen Rollenalias bereit und legt seinen Wert in der Nucleus-Komponente fest.

`interpolateComponentConfiguration`

(Optional) Sie können den Greengrass-Kern aktivieren, um Rezeptvariablen für Komponenten in Komponentenkonfigurationen zu interpolieren und Konfigurationsupdates zusammenzuführen. Wir empfehlen, diese Option auf zu setzen, `true` damit das Kerngerät Greengrass-Komponenten ausführen kann, die Rezeptvariablen in ihren Konfigurationen verwenden.

Diese Funktion ist für Version 2.6.0 und höher dieser Komponente verfügbar.

Standard: `false`

`networkProxy`

(Optional) Der Netzwerk-Proxy, der für alle Verbindungen verwendet werden soll. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy.

> **Important**
>
> Wenn Sie eine Änderung an diesem Konfigurationsparameter vornehmen, wird die AWS IoT Greengrass Core-Software neu gestartet, damit die Änderung wirksam wird.

Dieses Objekt enthält die folgenden Informationen:

`noProxyAddresses`

(Optional) Eine durch Kommas getrennte Liste von IP-Adressen oder Hostnamen, die vom Proxy ausgenommen sind.

`proxy`

Der Proxy, zu dem eine Verbindung hergestellt werden soll. Dieses Objekt enthält die folgenden Informationen:

`url`

Die URL des Proxyservers im Format`scheme://userinfo@host:port`.

- `scheme`— Das Schema, das `http` oder sein muss`https`.

> **Important**
>
> Greengrass-Core-Geräte müssen Greengrass Nucleus v2.5.0 oder höher ausführen, um HTTPS-Proxys verwenden zu können.
> Wenn Sie einen HTTPS-Proxy konfigurieren, müssen Sie das Proxy-Server-CA-Zertifikat zum Amazon-Root-CA-Zertifikat des Kerngeräts hinzufügen. Weitere Informationen finden Sie unter Ermöglichen Sie dem Kerngerät, einem HTTPS-Proxy zu vertrauen.

- `userinfo`— (Optional) Der Benutzername und das Passwort. Wenn Sie diese Informationen in der angeben`url`, ignoriert das Greengrass-Core-Gerät die Felder `username` und`password`.
- `host`— Der Hostname oder die IP-Adresse des Proxyservers.
- `port`— (Optional) Die Portnummer. Wenn Sie den Port nicht angeben, verwendet das Greengrass-Core-Gerät die folgenden Standardwerte:
  - `http`— 80
  - `https`— 443

`username`

(Optional) Der Benutzername, der den Proxyserver authentifiziert.

`password`

(Optional) Das Passwort, das den Proxyserver authentifiziert.

`mqtt`

(Optional) Die MQTT-Konfiguration für das Greengrass-Core-Gerät. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy.

> **Important**
>
> Wenn Sie eine Änderung an diesem Konfigurationsparameter vornehmen, wird die AWS IoT Greengrass Core-Software neu gestartet, damit die Änderung wirksam wird.

Dieses Objekt enthält die folgenden Informationen:

`port`

(Optional) Der Port, der für MQTT-Verbindungen verwendet werden soll.

Standard: `8883`

`keepAliveTimeoutMs`

(Optional) Die Zeitspanne in Millisekunden zwischen den einzelnen `PING` Nachrichten, die der Client sendet, um die MQTT-Verbindung aufrechtzuerhalten. Dieser Wert muss größer als sein. `pingTimeoutMs`

Standard: `60000` (60 Sekunden)

`pingTimeoutMs`

(Optional) Die Zeit in Millisekunden, die der Client auf den Empfang einer `PINGACK` Nachricht vom Server wartet. Wenn die Wartezeit das Timeout überschreitet, schließt das Kerngerät die MQTT-Verbindung und öffnet sie erneut. Dieser Wert muss kleiner als sein. `keepAliveTimeoutMs`

Standard: `30000` (30 Sekunden)

`operationTimeoutMs`

(Optional) Die Zeit in Millisekunden, die der Client auf den Abschluss von MQTT-Operationen (wie `CONNECT` oder) wartet. `PUBLISH` Diese Option gilt nicht für `PING` MQTT- oder Keep-Alive-Nachrichten.

Standard: `30000` (30 Sekunden)

`maxInFlightPublishes`

(Optional) Die maximale Anzahl unbestätigter MQTT QoS 1-Nachrichten, die gleichzeitig gesendet werden können.

Diese Funktion ist für Version 2.1.0 und höher dieser Komponente verfügbar.

Standard: `5`

Gültiger Bereich: Maximalwert von 100

`maxMessageSizeInBytes`

(Optional) Die maximale Größe einer MQTT-Nachricht. Wenn eine Nachricht diese Größe überschreitet, lehnt der Greengrass-Kern die Nachricht mit einem Fehler ab.

Diese Funktion ist für Version 2.1.0 und höher dieser Komponente verfügbar.

Standard: `131072` (128 KB)

Gültiger Bereich: Maximalwert von `2621440` (2,5 MB)

`maxPublishRetry`

(Optional) Gibt an, wie oft eine Nachricht, deren Veröffentlichung fehlschlägt, maximal wiederholt werden soll. Sie können angeben, dass der `-1` Wiederholungsversuch unbegrenzt oft erfolgen soll.

Diese Funktion ist für Version 2.1.0 und höher dieser Komponente verfügbar.

Standard: `100`

`spooler`

(Optional) Die MQTT-Spooler-Konfiguration für das Greengrass-Core-Gerät. Dieses Objekt enthält die folgenden Informationen:

`storageType`

Der Speichertyp zum Speichern von Nachrichten. Wenn auf gesetzt `storageType` ist`Disk`, `pluginName` kann der konfiguriert werden. Sie können entweder `Memory` oder `Disk` angeben.

Diese Funktion ist für Version 2.11.0 und höher der Greengrass Nucleus-Komponente verfügbar.

> **Important**
>
> Wenn der MQTT-Spooler auf eingestellt `storageType` ist `Disk` und Sie Greengrass Nucleus von Version 2.11.x auf eine frühere Version downgraden möchten, müssen Sie die Konfiguration wieder auf ändern. `Memory` Die einzige Konfiguration dafür`storageType`, die in den Greengrass Nucleus-Versionen 2.10.x und früher unterstützt wird, ist. `Memory` Die Nichtbeachtung dieser Anleitung kann dazu führen, dass der Spooler kaputt geht. Dies würde dazu führen, dass Ihr Greengrass-Core-Gerät keine MQTT-Nachrichten an den senden kann. AWS Cloud

Standard: `Memory`

`pluginName`

(Optional) Der Name der Plugin-Komponente. Diese Komponente wird nur verwendet, wenn sie auf gesetzt `storageType` ist`Disk`. Diese Option ist standardmäßig auf den von Festplatten-Spooler Greengrass bereitgestellten Wert eingestellt `aws.greengrass.DiskSpooler` und verwendet diesen.

Diese Funktion ist für Version 2.11.0 und höher der Greengrass Nucleus-Komponente verfügbar.

Standard: `"aws.greengrass.DiskSpooler"`

`maxSizeInBytes`

(Optional) Die maximale Größe des Caches, in dem das Kerngerät unverarbeitete MQTT-Nachrichten im Speicher speichert. Wenn der Cache voll ist, werden neue Nachrichten zurückgewiesen.

Standard: `2621440` (2,5 MB)

`keepQos0WhenOffline`

(Optional) Sie können MQTT QoS 0-Nachrichten spoolen, die das Kerngerät empfängt, während es offline ist. Wenn Sie diese Option auf setzen`true`, spoolt das Kerngerät QoS 0-Nachrichten, die es nicht senden kann, während es offline ist. Wenn Sie diese Option auf setzen`false`, verwirft das Kerngerät diese Nachrichten. Das Core-Gerät spoolt immer QoS-1-Nachrichten, sofern der Spool nicht voll ist.

Standard: `false`

`version`

(Optional) Die Version von MQTT. Sie können entweder `mqtt3` oder `mqtt5` angeben.

Diese Funktion ist für Version 2.10.0 und höher der Greengrass Nucleus-Komponente verfügbar.

Standard: `mqtt5`

`receiveMaximum`

(Optional) Die maximale Anzahl unbestätigter QoS1-Pakete, die der Broker senden kann.

Diese Funktion ist für Version 2.10.0 und höher der Greengrass Nucleus-Komponente verfügbar.

Standard: `100`

`sessionExpirySeconds`

(Optional) Die Dauer in Sekunden, die Sie für die Dauer einer Sitzung bei IoT Core anfordern können. Die Standardeinstellung ist die maximale Zeit, die von unterstützt wird AWS IoT Core.

Diese Funktion ist für Version 2.10.0 und höher der Greengrass Nucleus-Komponente verfügbar.

Standard: `604800 (7 days)`

`minimumReconnectDelaySeconds`

(Optional) Eine Option für das Verhalten bei der Wiederverbindung. Die Mindestzeit in Sekunden, die MQTT benötigt, um die Verbindung wiederherzustellen.

Diese Funktion ist für Version 2.10.0 und höher der Greengrass Nucleus-Komponente verfügbar.

Standard: `1`

`maximumReconnectDelaySeconds`

(Optional) Eine Option für das Verhalten bei der Wiederverbindung. Die maximale Zeit in Sekunden, für die MQTT die Verbindung wiederherstellt.

Diese Funktion ist für Version 2.10.0 und höher der Greengrass Nucleus-Komponente verfügbar.

Standard: `120`

`minimumConnectedTimeBeforeRetryResetSeconds`

(Optional) Eine Option für das Verhalten bei der Wiederverbindung. Die Zeitspanne in Sekunden, die eine Verbindung aktiv sein muss, bevor die Wiederholungsverzögerung auf das Minimum zurückgesetzt wird.

Diese Funktion ist für Version 2.10.0 und höher der Greengrass Nucleus-Komponente verfügbar.

Standard: `30`

`jvmOptions`

(Optional) Die JVM-Optionen, die zur Ausführung der Core-Software verwendet werden sollen. AWS IoT Greengrass Informationen zu den empfohlenen JVM-Optionen für die Ausführung der AWS IoT Greengrass Core-Software finden Sie unter. Steuern Sie die Speicherzuweisung mit JVM-Optionen

> **Important**
>
> Wenn Sie eine Änderung an diesem Konfigurationsparameter bereitstellen, wird die AWS IoT Greengrass Core-Software neu gestartet, damit die Änderung wirksam wird.

`iotDataEndpoint`

Der AWS IoT Datenendpunkt für Ihren AWS-Konto.

Wenn Sie die AWS IoT Greengrass Core-Software mit der `--provision true` Option ausführen, ruft die Software Ihre Daten und Anmeldeinformationen von den Endpunkten ab AWS IoT und legt sie in der Nucleus-Komponente fest.

`iotCredEndpoint`

Der Endpunkt für Ihre AWS IoT AWS-Konto Anmeldeinformationen.

Wenn Sie die AWS IoT Greengrass Core-Software mit der `--provision true` Option ausführen, ruft die Software Ihre Daten und Anmeldeinformationen von den Endpunkten ab AWS IoT und legt sie in der Nucleus-Komponente fest.

`greengrassDataPlaneEndpoint`

Diese Funktion ist in Version 2.7.0 und höher dieser Komponente verfügbar.

Weitere Informationen finden Sie unter Verwenden Sie ein Gerätezertifikat, das von einer privaten Zertifizierungsstelle signiert wurde.

`greengrassDataPlanePort`

Diese Funktion ist in Version 2.0.4 und höher dieser Komponente verfügbar.

(Optional) Der Port, der für Datenebenenverbindungen verwendet werden soll. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy.

> **Important**
>
> Sie müssen einen Port angeben, über den das Gerät ausgehende Verbindungen herstellen kann. Wenn Sie einen Port angeben, der gesperrt ist, kann das Gerät keine Verbindung herstellen, um Bereitstellungen AWS IoT Greengrass zu empfangen.

Wählen Sie aus den folgenden Optionen aus:

- `443`
- `8443`

Standard: `8443`

`awsRegion`

Der AWS-Region zu verwendende.

`runWithDefault`

Der Systembenutzer, der zum Ausführen von Komponenten verwendet werden soll.

> **Important**
>
> Wenn Sie eine Änderung an diesem Konfigurationsparameter vornehmen, wird die AWS IoT Greengrass Core-Software neu gestartet, damit die Änderung wirksam wird.

Dieses Objekt enthält die folgenden Informationen:

`posixUser`

Der Name oder die ID des Systembenutzers und optional der Systemgruppe, die das Kerngerät verwendet, um generische Komponenten und Lambda-Komponenten auszuführen. Geben Sie den Benutzer und die Gruppe durch einen Doppelpunkt (`:`) getrennt im folgenden Format an: `user:group`. Die Gruppe ist optional. Wenn Sie keine Gruppe angeben, verwendet die AWS IoT Greengrass Core-Software die primäre Gruppe für den Benutzer. Sie können beispielsweise `ggc_user` oder `ggc_user:ggc_group` angeben. Weitere Informationen finden Sie unter Konfigurieren Sie den Benutzer, der die Komponenten ausführt.

Wenn Sie das AWS IoT Greengrass Core-Softwareinstallationsprogramm mit der `--component-default-user` *`ggc_user`*`:`*`ggc_group`* Option ausführen, legt die Software diesen Parameter in der Nucleus-Komponente fest.

`windowsUser`

Diese Funktion ist in Version 2.5.0 und höher dieser Komponente verfügbar.

Der Name des Windows-Benutzers, der für die Ausführung dieser Komponente auf Windows Core-Geräten verwendet werden soll. Der Benutzer muss auf jedem Windows Core-Gerät vorhanden sein, und sein Name und Passwort müssen in der Credentials Manager-Instanz des LocalSystem Kontos gespeichert sein. Weitere Informationen finden Sie unter Konfigurieren Sie den Benutzer, der die Komponenten ausführt.

Wenn Sie das AWS IoT Greengrass Core-Softwareinstallationsprogramm mit der `--component-default-user` *`ggc_user`* Option ausführen, legt die Software diesen Parameter in der Nucleus-Komponente fest.

`systemResourceLimits`

Diese Funktion ist in Version 2.4.0 und höher dieser Komponente verfügbar. AWS IoT Greengrass unterstützt diese Funktion derzeit nicht auf Windows Core-Geräten.

Die Systemressourcenlimits, die standardmäßig für generische und nicht containerisierte Lambda-Komponentenprozesse gelten sollen. Sie können die Systemressourcenlimits für einzelne Komponenten überschreiben, wenn Sie eine Bereitstellung erstellen. Weitere Informationen finden Sie unter Konfigurieren Sie die Systemressourcenlimits für Komponenten.

Dieses Objekt enthält die folgenden Informationen:

`cpus`

Die maximale Menge an CPU-Zeit, die die Prozesse der einzelnen Komponenten auf dem Kerngerät verwenden können. Die gesamte CPU-Zeit eines Core-Geräts entspricht der Anzahl der CPU-Kerne des Geräts. Auf einem Core-Gerät mit 4 CPU-Kernen können Sie diesen Wert beispielsweise auf festlegen, `2` um die Prozesse jeder Komponente auf 50 Prozent der Auslastung jedes CPU-Kerns zu beschränken. Auf einem Gerät mit einem CPU-Kern können Sie diesen Wert auf festlegen, `0.25` um die Prozesse jeder Komponente auf 25 Prozent der CPU-Auslastung zu beschränken. Wenn Sie diesen Wert auf eine Zahl festlegen, die größer als die Anzahl der CPU-Kerne ist, begrenzt die AWS IoT Greengrass Core-Software die CPU-Auslastung der Komponenten nicht.

`memory`

Die maximale Menge an RAM (in Kilobyte), die die Prozesse jeder Komponente auf dem Kerngerät verwenden können.

`s3EndpointType`

(Optional) Der S3-Endpunkttyp. Dieser Parameter gilt nur für die Region USA Ost (Nord-Virginia) (`us-east-1`). Die Einstellung dieses Parameters aus einer anderen Region wird ignoriert. Wählen Sie aus den folgenden Optionen aus:

- `REGIONAL`— Der S3-Client und die vorsignierte URL verwenden den regionalen Endpunkt.
- `GLOBAL`— Der S3-Client und die vorsignierte URL verwenden den Legacy-Endpunkt.

Standard: `GLOBAL`

`logging`

(Optional) Die Protokollierungskonfiguration für das Kerngerät. Weitere Informationen zur Konfiguration und Verwendung von Greengrass-Protokollen finden Sie unterÜberwachen von AWS IoT Greengrass Protokollen.

Dieses Objekt enthält die folgenden Informationen:

`level`

(Optional) Die Mindestanzahl der auszugebenden Protokollmeldungen.

Wählen Sie aus den folgenden Protokollebenen, die hier in der Reihenfolge der Stufen aufgeführt sind:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Standard: `INFO`

`format`

(Optional) Das Datenformat der Protokolle. Wählen Sie aus den folgenden Optionen aus:

- `TEXT`— Wählen Sie diese Option, wenn Sie Protokolle in Textform anzeigen möchten.
- `JSON`— Wählen Sie diese Option, wenn Sie Protokolle mit dem Greengrass-CLI-Befehl logs anzeigen oder programmgesteuert mit Protokollen interagieren möchten.

Standard: `TEXT`

`outputType`

(Optional) Der Ausgabetyp für Protokolle. Wählen Sie aus den folgenden Optionen aus:

- `FILE`— Die AWS IoT Greengrass Core-Software gibt Protokolle in Dateien in dem Verzeichnis aus, das Sie angeben`outputDirectory`.
- `CONSOLE`— Die AWS IoT Greengrass Core-Software druckt Protokolle in`stdout`. Wählen Sie diese Option, um Protokolle anzuzeigen, während das Core-Gerät sie druckt.

Standard: `FILE`

`fileSizeKB`

(Optional) Die maximale Größe jeder Protokolldatei (in Kilobyte). Wenn eine Protokolldatei diese maximale Dateigröße überschreitet, erstellt die AWS IoT Greengrass Core-Software eine neue Protokolldatei.

Dieser Parameter gilt nur, wenn Sie `FILE` für angeben`outputType`.

Standard: `1024`

`totalLogsSizeKB`

(Optional) Die maximale Gesamtgröße der Protokolldateien (in Kilobyte) für jede Komponente, einschließlich des Greengrass-Kerns. Die Protokolldateien von Greengrass Nucleus enthalten auch Protokolle von Plugin-Komponenten. Wenn die Gesamtgröße der Protokolldateien einer Komponente diese maximale Größe überschreitet, löscht die AWS IoT Greengrass Core-Software die ältesten Protokolldateien dieser Komponente.

Dieser Parameter entspricht dem Parameter für die Speicherplatzbeschränkung (`diskSpaceLimit`) der Log Manager-Komponente, den Sie für den Greengrass-Kern (System) und jede Komponente angeben können. Die AWS IoT Greengrass Core-Software verwendet das Minimum der beiden Werte als maximale Gesamtprotokollgröße für den Greengrass-Kern und jede Komponente.

Dieser Parameter gilt nur, wenn Sie `FILE` für `outputType` angeben.

Standard: `10240`

`outputDirectory`

(Optional) Das Ausgabeverzeichnis für Protokolldateien.

Dieser Parameter gilt nur, wenn Sie `FILE` für angeben`outputType`.

Standard:*`/greengrass/v2`*`/logs`, wo *`/greengrass/v2`* ist der AWS IoT Greengrass Stammordner.

`fleetstatus`

Dieser Parameter ist in Version 2.1.0 und höher dieser Komponente verfügbar.

(Optional) Die Flottenstatuskonfiguration für das Kerngerät.

Dieses Objekt enthält die folgenden Informationen:

`periodicStatusPublishIntervalSeconds`

(Optional) Der Zeitraum (in Sekunden), zwischen dem das Kerngerät den Gerätestatus an den veröffentlicht AWS Cloud.

Minimum: `86400` (24 Stunden)

Standard: `86400` (24 Stunden)

`telemetry`

(Optional) Die Konfiguration der Systemintegritäts-Telemetrie für das Kerngerät. Weitere Informationen zu Telemetriemetriken und dazu, wie Sie mit Telemetriedaten umgehen können, finden Sie unter. Erfassen von Telemetriedaten zum Systemstatus von -AWS IoT GreengrassCore-Geräten

Dieses Objekt enthält die folgenden Informationen:

`enabled`

(Optional) Sie können Telemetrie aktivieren oder deaktivieren.

Standard: `true`

`periodicAggregateMetricsIntervalSeconds`

(Optional) Das Intervall (in Sekunden), über das das Kerngerät Metriken aggregiert.

Wenn Sie diesen Wert unter den unterstützten Mindestwert setzen, verwendet der Nucleus stattdessen den Standardwert.

Minimum: `3600`

Standard: `3600`

`periodicPublishMetricsIntervalSeconds`

(Optional) Der Zeitraum (in Sekunden), zwischen dem das Kerngerät Telemetriedaten an den AWS Cloud veröffentlicht.

Wenn Sie diesen Wert unter den unterstützten Mindestwert einstellen, verwendet der Nucleus stattdessen den Standardwert.

Minimum: `86400`

Standard: `86400`

`deploymentPollingFrequencySeconds`

(Optional) Der Zeitraum in Sekunden, für den Bereitstellungsbenachrichtigungen abgefragt werden sollen.

Standard: `15`

`componentStoreMaxSizeBytes`

(Optional) Die maximale Größe des Komponentenspeichers auf der Festplatte, der Komponentenrezepte und Artefakte umfasst.

Standard: `10000000000` (10 GB)

`platformOverride`

(Optional) Ein Wörterbuch mit Attributen, die die Plattform des Kerngeräts identifizieren. Verwenden Sie dies, um benutzerdefinierte Plattformattribute zu definieren, anhand derer Komponentenrezepte den richtigen Lebenszyklus und die richtigen Artefakte für die Komponente identifizieren können. Sie können beispielsweise ein Hardwarefähigkeitsattribut definieren, um nur die minimale Menge an Artefakten bereitzustellen, die für die Ausführung einer Komponente erforderlich sind. Weitere Informationen finden Sie unter dem Plattformparameter Manifest im Komponentenrezept.

Sie können diesen Parameter auch verwenden, um die `os` und die `architecture` Plattformattribute des Kerngeräts zu überschreiben.

`httpClient`

Dieser Parameter ist in Version 2.5.0 und höher dieser Komponente verfügbar.

(Optional) Die HTTP-Client-Konfiguration für das Kerngerät. Diese Konfigurationsoptionen gelten für alle HTTP-Anfragen, die von dieser Komponente gestellt werden. Wenn ein Core-Gerät in einem langsameren Netzwerk läuft, können Sie diese Timeout-Dauern erhöhen, um zu verhindern, dass bei HTTP-Anfragen ein Timeout auftritt.

Dieses Objekt enthält die folgenden Informationen:

`connectionTimeoutMs`

(Optional) Die Wartezeit (in Millisekunden), bis eine Verbindung geöffnet wird, bevor bei der Verbindungsanforderung ein Timeout eintritt.

Standard: `2000` (2 Sekunden)

`socketTimeoutMs`

(Optional) Die Zeitspanne (in Millisekunden), die auf die Übertragung von Daten über eine offene Verbindung gewartet werden soll, bevor die Verbindung unterbrochen wird.

Standard: `30000` (30 Sekunden)

Example Beispiel: Aktualisierung der Konfigurationszusammenführung

```
{
  "iotRoleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "networkProxy": {
    "noProxyAddresses": "http://192.168.0.1,www.example.com",
    "proxy": {
      "url": "http://my-proxy-server:1100",
      "username": "Mary_Major",
      "password": "pass@word1357"
    }
  },
  "mqtt": {
    "port": 443
  },
  "greengrassDataPlanePort": 443,
  "jvmOptions": "-Xmx64m",
  "runWithDefault": {
    "posixUser": "ggc_user:ggc_group"
  }
}
```

Lokale Protokolldatei

Diese Komponente verwendet die folgende Protokolldatei.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

Um die Protokolle dieser Komponente einzusehen

- Führen Sie den folgenden Befehl auf dem Kerngerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

  Linux

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

  Windows (PowerShell)

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

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
2.12.6	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, das bei bestimmten ARMv8-Prozessoren, einschließlich dem Jetson Nano, beim Start zu einem Absturz führt.
2.12.5	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem das Rollback der Bereitstellung gelegentlich hängen bleibt, wenn eine zuvor defekte Komponente mit festen Abhängigkeiten zurückgesetzt wird. • Behebt ein Problem, bei dem der Nucleus nach der Flottenbereitstellung keine Status-Updates veröffentlicht. • Fügt Wiederholungsversuche für die `GetDeploymentConfiguration` API hinzu, nachdem 404-Fehler aufgetreten sind.

Version	Änderungen
2.12.4	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem der Nucleus beim Start auf einigen Linux-Geräten in einen Deadlock-Zustand gerät.
2.12.3	**Warning** Diese Version ist nicht mehr verfügbar. Die Verbesserungen in dieser Version sind in späteren Versionen dieser Komponente verfügbar. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem der Nucleus nach dem Neustart des Nucleus und während der Wiederherstellung der Komponenten nicht den korrekten Komponentenstatus meldet. • Allgemeine Fehlerbehebungen und Verbesserungen.
2.12.2	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem alte Logs nicht richtig bereinigt wurden. • Allgemeine Fehlerbehebungen und Verbesserungen.
2.12.1	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem der Nucleus MQTT-Abonnements für Bereitstellungsthemen duplizieren kann, was zu zusätzlicher Protokollierung und MQTT-Veröffentlichung führt.
2.12.0	Neue Features • Ermöglicht es Ihnen, die Bootstrap-Lebenszyklusschritte als Teil einer Rollback-Bereitstellung auszuführen.

Version	Änderungen
2.11.3	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem im Nucleus, bei dem eine Komponente möglicher weise falsch gestartet wird, wenn ihre Abhängigkeiten fehlschlagen. Neue Features • Fügt den konfigurierbaren S3-Endpunkttyp hinzu.
2.11.2	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem im Nucleus MQTT 5-Client, bei dem er offline angezeigt werden kann, wenn eine große Anzahl (> 50) von Abonnemen ts verwendet wird. • Fügt einen erneuten Versuch für den Docker-Dial-TCP-Fehler hinzu.
2.11.1	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem der Nucleus nicht startet, wenn eine Bootstrap-Aufgabe fehlschlägt und die Deployment-Metadatendatei beschädigt ist. • Behebt ein Problem, bei dem On-Demand-Lambda-Komponenten in Statusaktualisierungen für die Bereitstellung nicht gemeldet werden. • Fügt Unterstützung für doppelte Autorisierungsrichtlinien-IDs hinzu.
2.11.0	Neue Features • Ermöglicht es Ihnen, eine lokale Bereitstellung abzubrechen. • Ermöglicht die Konfiguration einer Fehlerbehandlungsrichtlinie für eine lokale Bereitstellung. • Fügt Unterstützung für ein Disk-Spooler-Plugin hinzu.
2.10.3	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem Greengrass keine Bereitstellungsben achrichtigungen abonniert, wenn der PKCS #11 -Anbieter verwendet wird.

Version	Änderungen
2.10.2	Fehlerkorrekturen und Verbesserungen • Ermöglicht die Analyse von Komponenten-Lebenszyklen ohne Berücksichtigung der Groß- und Kleinschreibung. • Behebt ein Problem, bei dem die Umgebungsvariable PATH nicht korrekt neu erstellt wurde. • Behebt die Proxy-URI-Kodierung für Komponenten, einschließlich des Stream-Managers für Benutzernamen mit Sonderzeichen.
2.10.1	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, das bei bestimmten ARMv8-Prozessoren, einschließlich des Jetson Nano, beim Start zu einem Absturz führen konnte. • Greengrass schließt den Standard einer Komponente nicht mehr, dadurch wird das Verhalten auf das Verhalten vor 2.10.0 zurückgesetzt

Version	Änderungen
2.10.0	Neue Features • Fügt `interpolateComponentConfiguration` Unterstützung für den leeren regulären Ausdruck hinzu. Greengrass interpoliert jetzt vom Root-Konfigurationsobjekt aus. • Fügt Unterstützung für MQTT5 hinzu. • Fügt einen Mechanismus zum schnellen Laden von Plugin-Ko mponenten ohne Scannen hinzu. • Ermöglicht Greengrass, Speicherplatz zu sparen, indem unbenutzte Docker-Images gelöscht werden. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem beim Rollback bestimmte Konfigura tionswerte aus einer Bereitstellung beibehalten werden. • Behebt ein Problem, bei dem der Greengrass-Nucleus nach einer AWS Domain-Sequenz in benutzerdefinierten AWS Nicht-Credentials und Datenendpunkten validiert. • Aktualisiert die Auflösung von Abhängigkeiten mehrerer Gruppen, sodass alle Gruppenabhängigkeiten per AWS Cloud Aushandlung erneut aufgelöst werden, anstatt sich an die aktive Version zu binden. Mit diesem Update wird auch der Bereitstellungsfehlercode entfernt. `INSTALLED_COMPONENT_NOT_FOUND` • Aktualisiert den Greengrass-Kern, sodass das Herunterladen von Docker-Images übersprungen wird, wenn sie bereits lokal vorhanden sind. • Aktualisiert den Greengrass-Nucleus so, dass er einen Komponent eninstallationsschritt neu startet, bevor das Timeout abläuft. • Zusätzliche kleinere Korrekturen und Verbesserungen.

Version	Änderungen
2.9.6	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem eine Greengrass-Bereitstellung mit dem Fehler LAUNCH_DIRECTORY_CORRUPTED fehlschlägt und Greengrass bei einem nachfolgenden Geräteneustart nicht gestartet werden kann. Dieser Fehler kann auftreten, wenn Sie das Greengrass-Gerät zwischen mehreren Dinggruppen mit Bereitstellungen verschieb en, für die Greengrass neu gestartet werden muss.
2.9.5	Neue Features • Fügt Unterstützung für die Überprüfung der Greengrass Nucleus-S oftware hinzu. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem eine Bereitstellung fehlschlägt, wenn die lokale Rezept-Metadatenregion nicht mit der Greengrass Nucleus-S tartregion übereinstimmt. Wenn das passiert, verhandelt der Greengras s-Kern nun neu mit der Cloud. • Behebt ein Problem, bei dem der MQTT-Nachrichtenspooler Nachricht en füllt und sie nie löscht. • Zusätzliche kleinere Korrekturen und Verbesserungen.
2.9.4	Fehlerkorrekturen und Verbesserungen • Prüft, ob eine Null-Nachricht vorliegt, bevor QOS 0-Nachrichten gelöscht werden. • Kürzt die Detailwerte für den Auftragsstatus, wenn sie das Limit von 1024 Zeichen überschreiten. • Aktualisiert das Bootstrap-Skript für Windows, sodass es den Greengras s-Stammpfad korrekt liest, falls dieser Pfad Leerzeichen enthält. • Aktualisiert das Abonnement, AWS IoT Core sodass Kundennachrichten gelöscht werden, wenn die Abonnementantwort nicht gesendet wurde. • Stellt sicher, dass der Nucleus seine Konfiguration aus Backup-Dateien lädt, wenn die Hauptkonfigurationsdatei beschädigt ist oder fehlt.

Version	Änderungen
2.9.3	Fehlerkorrekturen und Verbesserungen • Stellt sicher, dass MQTT-Client-IDs nicht dupliziert werden. • Fügt robusteres Lesen und Schreiben von Dateien hinzu, um Beschädigungen zu vermeiden und diese wiederherzustellen. • Wiederholt den Docker-Image-Pull bei bestimmten Netzwerkfehlern. • Fügt die `noProxyAddresses` Option für eine MQTT-Verbindung hinzu.
2.9.2	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem die Konfiguration `interpolateComponentConfiguration` nicht für eine laufende Bereitstellung gilt. • Verwendet OSHI, um alle untergeordneten Prozesse aufzulisten.
2.9.1	Fehlerkorrekturen und Verbesserungen • Fügt einen Fix hinzu, bei dem Greengrass neu startet, wenn eine Bereitstellung eine Plugin-Komponente entfernt.
2.9.0	Neue Features • Fügt die Möglichkeit hinzu, Unterbereitstellungen zu erstellen, die Bereitstellungen mit einer kleineren Teilmenge von Geräten wiederholen. Mit dieser Funktion können erfolglose Bereitstellungen effizienter getestet und behoben werden. Fehlerkorrekturen und Verbesserungen • Verbessert die Unterstützung für Systeme, die nicht über `useraddgroupadd`, und verfügen`usermod`. • Zusätzliche kleinere Korrekturen und Verbesserungen.

Version	Änderungen
2.8.1	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem Bereitstellungsfehlercodes aufgrund von Greengrass-API-Fehlern nicht korrekt generiert wurden. • Behebt ein Problem, bei dem Aktualisierungen des Flottenstatus ungenaue Informationen senden, wenn eine Komponente während eines Einsatzes einen `ERRORED` bestimmten Status erreicht. • Behebt ein Problem, bei dem Bereitstellungen nicht abgeschlossen werden konnten, wenn Greengrass über mehr als 50 bestehende Abonnements verfügte.

Version	Änderungen
2.8.0	Neue Features • Aktualisiert den Greengrass-Nucleus so, dass er eine Antwort auf den Bereitstellungsstatus meldet, die detaillierte Fehlercodes enthält, wenn bei der Bereitstellung von Komponenten auf einem Kerngerät ein Problem auftritt. Weitere Informationen finden Sie unter Detaillierte Bereitstellungsfehlercodes. • Aktualisiert den Greengrass-Kern so, dass er eine Antwort auf den Komponentenstatus meldet, die detaillierte Fehlercodes enthält, wenn eine Komponente in den `ERRORED` Status `BROKEN` oder wechselt. Weitere Informationen finden Sie unter Detaillierte Komponenten-Statuscodes. • Erweitert die Felder für Statusmeldungen, um die Informationen zur Cloud-Verfügbarkeit für Geräte zu verbessern. • Verbessert die Robustheit der Dienste für den Flottenstatus. Fehlerkorrekturen und Verbesserungen • Ermöglicht die Neuinstallation einer defekten Komponente, wenn sich ihre Konfiguration ändert. • Behebt ein Problem, bei dem ein Nucleus-Neustart während der Bootstrap-Bereitstellung dazu führt, dass eine Bereitstellung fehlschlägt. • Behebt ein Problem in Windows, bei dem die Installation fehlschlägt, wenn ein Stammpfad Leerzeichen enthält. • Behebt ein Problem, bei dem eine Komponente, die während einer Bereitstellung heruntergefahren wird, das Shutdown-Skript der neuen Version verwendet. • Verschiedene Verbesserungen beim Herunterfahren. • Zusätzliche kleinere Korrekturen und Verbesserungen.

Version	Änderungen
2.7.0	Neue Features • Aktualisiert den Greengrass-Nucleus, sodass Status-Updates an die AWS IoT Greengrass Cloud gesendet werden, wenn das Kerngerät eine lokale Bereitstellung anwendet. • Integriert die Unterstützung für Client-Zertifikate, die von einer benutzerdefinierten Zertifizierungsstelle (CA) signiert wurden, bei der die CA nicht registriert ist. AWS IoT Um diese Funktion zu verwenden, können Sie die neue `greengrassDataPlaneEndpoint` Konfigurationsoption auf einstellen`iotdata`. Weitere Informationen finden Sie unter Verwenden Sie ein Gerätezertifikat, das von einer privaten Zertifizierungsstelle signiert wurde. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem der Greengrass Nucleus eine Bereitstellung in bestimmten Szenarien rückgängig macht, wenn der Nucleus gestoppt oder neu gestartet wird. Der Nucleus nimmt den Einsatz jetzt wieder auf, nachdem der Nucleus neu gestartet wurde. • Aktualisiert das Greengrass-Installationsprogramm so, dass es das `--start` Argument berücksichtigt, wenn Sie angeben, dass die Software als Systemdienst eingerichtet werden soll. • Aktualisiert das Verhalten von SubscribeToComponentUpdates zum Einstellen der Bereitstellungs-ID bei Ereignissen, bei denen der Nucleus eine Komponente aktualisiert hat. • Zusätzliche kleinere Korrekturen und Verbesserungen.

Version	Änderungen
2.6.0	Neue Features • Fügt Unterstützung für MQTT-Platzhalter hinzu, wenn Sie lokale Themen zum Publizieren/Abonnieren abonnieren. Weitere Informationen finden Sie unter Lokale Nachrichten veröffentlichen/abonnieren und SubscribeToTopic. • Fügt Unterstützung für Rezeptvariablen in Komponentenkonfigurationen hinzu, mit Ausnahme der Rezeptvariablen. *component_dependency_name* :configuration: *json_pointer* Sie können diese Rezeptvariablen verwenden, wenn Sie Komponenten DefaultConfiguration in einem Rezept definieren oder wenn Sie eine Komponente in einer Bereitstellung konfigurieren. Um diese Funktion zu aktivieren, setzen Sie die ComponentConfiguration Konfigurationsoption interpolieren auf. true Weitere Informationen finden Sie unter Rezeptvariablen und Verwenden Sie Rezeptvariablen bei Merge-Updates. • Fügt vollständige Unterstützung für den * Platzhalter in Autorisierungsrichtlinien für die Interprozesskommunikation (IPC) hinzu. Sie können jetzt das * Zeichen in einer Ressourcenzeichenfolge so angeben, dass es einer beliebigen Zeichenkombination entspricht. Weitere Informationen finden Sie unter Platzhalter in Autorisierungsrichtlinien. • Integriert die Unterstützung für benutzerdefinierte Komponenten zum Aufrufen von IPC-Operationen, die die Greengrass-CLI verwendet. Sie können diese IPC-Operationen verwenden, um lokale Bereitstellungen zu verwalten, Komponentendetails anzuzeigen und ein Passwort zu generieren, mit dem Sie sich bei der lokalen Debug-Konsole anmelden können. Weitere Informationen finden Sie unter IPC: Lokale Bereitstellungen und Komponenten verwalten. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem abhängige Komponenten in bestimmten Szenarien nicht reagierten, wenn ihre festen Abhängigkeiten neu gestartet wurden oder ihren Status änderten.

Version	Änderungen
	• Verbessert die Fehlermeldungen, die das Kerngerät an den AWS IoT Greengrass Cloud-Dienst meldet, wenn eine Bereitstellung fehlschlägt. • Behebt ein Problem, bei dem der Greengrass-Kern in bestimmten Szenarien beim Neustart des Nucleus zweimal eine Ding-Bereitstellung durchgeführt hat. • Zusätzliche kleinere Korrekturen und Verbesserungen. Weitere Informationen finden Sie in den Veröffentlichungen unter GitHub.
2.5.6	Neue Features • Integriert die Unterstützung für Hardware-Sicherheitsmodule, die ECC-Schlüssel verwenden. Sie können ein Hardware-Sicherheitsmodul (HSM) verwenden, um den privaten Schlüssel und das Zertifikat des Geräts sicher zu speichern. Weitere Informationen finden Sie unter Integration von Hardware-Sicherheit. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem die Bereitstellung nie abgeschlossen wird, wenn Sie in bestimmten Szenarien eine Komponente mit einem defekten Installationsskript bereitstellen. • Verbessert die Leistung beim Start. • Zusätzliche kleinere Korrekturen und Verbesserungen.

Version	Änderungen
2.5.5	Neue Features • Fügt die `GG_ROOT_CA_PATH` Umgebungsvariable für Komponenten hinzu, sodass Sie in benutzerdefinierten Komponenten auf das Zertifikat der Stammzertifizierungsstelle (CA) zugreifen können. Fehlerkorrekturen und Verbesserungen • Integriert die Unterstützung für Windows-Geräte, die eine andere Anzeigesprache als Englisch verwenden. • Aktualisiert die Art und Weise, wie der Greengrass-Nucleus boolesche Installer-Argumente analysiert, sodass Sie ein boolesches Argument ohne einen booleschen Wert angeben können, um einen Wert anzugeben. `true` Beispielsweise können Sie jetzt angeben, dass die Installation `--provision` statt der automatischen Ressourcenbereitst ellung erfolgen soll. `--provision true` • Behebt ein Problem, bei dem das Kerngerät in bestimmten Szenarien seinen Status nach der Bereitstellung nicht an den AWS IoT Greengrass Cloud-Dienst gemeldet hat. • Zusätzliche kleinere Korrekturen und Verbesserungen.
2.5.4	Fehlerkorrekturen und Verbesserungen • Allgemeine Fehlerbehebungen und Verbesserungen.
2.5.3	Neue Features • Fügt Unterstützung für die Hardware-Sicherheitsintegration hinzu. Sie können ein Hardware-Sicherheitsmodul (HSM) verwenden, um den privaten Schlüssel und das Zertifikat des Geräts sicher zu speichern . Weitere Informationen finden Sie unter Integration von Hardware-Sicherheit. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem mit Laufzeitausnahmen, während der Nucleus MQTT-Verbindungen mit AWS IoT Core herstellt.

Version	Änderungen
2.5.2	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem der Windows-Dienst nach dem Greengrass Nucleus-Update nicht erneut gestartet werden kann, nachdem Sie ihn gestoppt oder das Gerät neu gestartet haben.
2.5.1	**Warning** Diese Version ist nicht mehr verfügbar. Die Verbesserungen in dieser Version sind in späteren Versionen dieser Komponente verfügbar. Fehlerkorrekturen und Verbesserungen • Integriert die Unterstützung für 32-Bit-Versionen des Java Runtime Environment (JRE) unter Windows. • Ändert das Verhalten beim Entfernen von Dinggruppen für Kerngeräte, deren AWS IoT Richtlinie diese `greengrass:ListThingGroupsForCoreDevice` Berechtigung nicht erteilt. Bei dieser Version wird die Bereitstellung fortgesetzt, es wird eine Warnung protokolliert und es werden keine Komponenten entfernt, wenn Sie das Kerngerät aus einer Dinggruppe entfernen. Weitere Informationen finden Sie unter Bereitstellen von AWS IoT Greengrass Komponenten auf Geräten. • Behebt ein Problem mit Systemumgebungsvariablen, die der Greengrass-Kern für Greengrass-Komponentenprozesse zur Verfügung stellt. Sie können eine Komponente jetzt neu starten, damit sie die neuesten Systemumgebungsvariablen verwendet.

Version	Änderungen
2.5.0	Neue Features • Fügt Unterstützung für Kerngeräte hinzu, auf denen Windows ausgeführt wird. • Ändert das Verhalten beim Entfernen von Dinggruppen. Mit dieser Version können Sie ein Kerngerät aus einer Dinggruppe entfernen, um die Komponenten dieser Dinggruppe in der nächsten Bereitstellung zu deinstallieren. Aufgrund dieser Änderung muss die AWS IoT Richtlinie eines Kerngeräts über die `greengrass:ListThingGroupsForCoreDevice` entsprechende Genehmigung verfügen. Wenn Sie das AWS IoT Greengrass Core-Softwareinstallationsprogramm zur Bereitstellung von Ressourcen verwendet haben, ist dies in der AWS IoT Standardrichtlinie zulässig`greengrass:*` , was auch diese Berechtigung beinhaltet. Weitere Informationen finden Sie unter Geräteauthentifizierung und -autorisierung für AWS IoT Greengrass. • Fügt Unterstützung für HTTPS-Proxykonfigurationen hinzu. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy. • Fügt den neuen `windowsUser` Konfigurationsparameter hinzu. Sie können diesen Parameter verwenden, um den Standardbenutzer anzugeben, der zum Ausführen von Komponenten auf einem Windows-Core-Gerät verwendet werden soll. Weitere Informationen finden Sie unter Konfigurieren Sie den Benutzer, der die Komponenten ausführt. • Fügt die neuen `httpClient` Konfigurationsoptionen hinzu, mit denen Sie Timeouts für HTTP-Anfragen anpassen können, um die Leistung in langsamen Netzwerken zu verbessern. Weitere Informationen finden Sie im HttpClient-Konfigurationsparameter. Fehlerkorrekturen und Verbesserungen • Behebt die Bootstrap-Lebenszyklusoption, um das Kerngerät von einer Komponente aus neu zu starten. • Fügt Unterstützung für Bindestriche in Rezeptvariablen hinzu.

Version	Änderungen
	• Behebt die IPC-Autorisierung für On-Demand-Lambda-Funktionsk omponenten. • Verbessert die Protokollnachrichten und ändert unkritische Protokolle von einer `DEBUG` Ebene `INFO` zur nächsten, sodass Protokolle nützliche r sind. • Entfernt die `iot:DescribeCertificate` Berechtigung aus der Standard-Token-Austauschrolle, die der Greengrass-Nucleus erstellt, wenn Sie die AWS IoT Greengrass Core-Software mit automatischer Bereitstellung installieren. Diese Erlaubnis wird vom Greengrass-Kern nicht verwendet. • Behebt ein Problem, sodass das automatische Bereitstellungsskript die `iam:GetPolicy` Genehmigung nicht benötigt, wenn `iam:CreatePolicy` es für dieselbe Richtlinie verfügbar ist. • Zusätzliche kleinere Korrekturen und Verbesserungen.

Version	Änderungen
2.4.0	Neue Features • Fügt Unterstützung für Systemressourcenlimits hinzu. Sie können die maximale CPU- und RAM-Auslastung konfigurieren, die die Prozesse der einzelnen Komponenten auf dem Kerngerät verwenden können. Weitere Informationen finden Sie unter Konfigurieren Sie die Systemressourcenlimits für Komponenten. • Fügt IPC-Operationen hinzu, um Komponenten anzuhalten und wieder aufzunehmen. Weitere Informationen finden Sie unter PauseComponent und ResumeComponent. • Fügt Unterstützung für Provisioning-Plugins hinzu. Sie können eine JAR-Datei angeben, die während der Installation ausgeführt wird, um die erforderlichen AWS Ressourcen für ein Greengrass-Core-Gerät bereitzustellen. Der Greengrass-Kern umfasst eine Schnittstelle, die Sie implementieren können, um benutzerdefinierte Provisioning-Plugins zu entwickeln. Weitere Informationen finden Sie unter Installieren Sie die AWS IoT Greengrass Core-Software mit benutzerdefinierter Ressourcenbereitstellung. • Fügt dem AWS IoT Greengrass Core-Softwareinstaller das optionale `thing-name-policy` Argument hinzu. Sie können diese Option verwenden, um eine vorhandene oder benutzerdefinierte AWS IoT Richtlinie anzugeben, wenn Sie die AWS IoT Greengrass Core-Software mit automatischer Ressourcenbereitstellung installieren. Fehlerkorrekturen und Verbesserungen • Aktualisiert die Protokollierungskonfiguration beim Start. Dies behebt ein Problem, bei dem die Protokollierungskonfiguration beim Start nicht angewendet wurde. • Aktualisiert den Nucleus-Loader-Symlink so, dass er während der Installation auf den Komponentenspeicher im Greengrass-Stammordner verweist. Mit diesem Update können Sie die JAR-Datei und andere Nucleus-Artefakte löschen, die Sie bei der Installation der AWS IoT Greengrass Core-Software herunterladen.

Version	Änderungen
	• Zusätzliche kleinere Korrekturen und Verbesserungen. Weitere Informationen finden Sie in den Veröffentlichungen unter GitHub.
2.3.0	Neue Features • Integriert die Unterstützung für Bereitstellungskonfigurationsdokumente bis zu 10 MB, statt 7 KB (für Bereitstellungen, die auf Dinge abzielen) oder 31 KB (für Bereitstellungen, die auf Dinggruppen abzielen). Um diese Funktion nutzen zu können, muss die AWS IoT Richtlinie eines Kerngeräts die `greengrass:GetDeploymentConfiguration` Genehmigung zulassen. Wenn Sie das AWS IoT Greengrass Core-Softwareinstallationsprogramm zur Bereitstellung von Ressourcen verwendet haben, ist dies gemäß den AWS IoT Richtlinien Ihres Kerngeräts zulässig`greengrass:*` , was auch diese Berechtigung einschließt. Weitere Informationen finden Sie unter Geräteauthentifizierung und -autorisierung für AWS IoT Greengrass. • Fügt die `iot:thingName` Rezeptvariable hinzu. Sie können diese Rezeptvariable verwenden, um den Namen der AWS IoT Sache des Kerngeräts in einem Rezept abzurufen. Weitere Informationen finden Sie unter Rezeptvariablen. Fehlerkorrekturen und Verbesserungen • Zusätzliche kleinere Korrekturen und Verbesserungen. Weitere Informationen finden Sie in den Veröffentlichungen unter GitHub.
2.2.0	Neue Features • Fügt IPC-Operationen für die lokale Schattenverwaltung hinzu. Fehlerkorrekturen und Verbesserungen • Reduziert die Größe der JAR-Datei. • Reduziert den Speicherverbrauch. • Behebt Probleme, bei denen die Protokollkonfiguration in bestimmten Fällen nicht aktualisiert wurde. • Zusätzliche kleinere Korrekturen und Verbesserungen. Weitere Informationen finden Sie in den Veröffentlichungen unter GitHub.

Version	Änderungen

2.1.0	Neue Features
	• Unterstützt das Herunterladen von Docker-Images aus privaten Repositorys in Amazon ECR.
	• Fügt die folgenden Parameter hinzu, um die MQTT-Konfiguration auf Kerngeräten anzupassen:
	  • `maxInFlightPublishes` — Die maximale Anzahl unbestätigter MQTT QoS 1-Nachrichten, die gleichzeitig gesendet werden können.
	  • `maxPublishRetry` — Die maximale Anzahl an Wiederholungen einer Nachricht, die nicht veröffentlicht werden kann.
	• Fügt den `fleetstatusservice` Konfigurationsparameter hinzu, um das Intervall zu konfigurieren, in dem das Kerngerät den Gerätestatus für veröffentlicht. AWS Cloud
	• Zusätzliche kleinere Korrekturen und Verbesserungen. Weitere Informationen finden Sie in den Veröffentlichungen unter GitHub.
	Fehlerkorrekturen und Verbesserungen
	• Behebt ein Problem, das dazu führte, dass Shadow-Bereitstellungen dupliziert wurden, wenn der Nucleus neu gestartet wurde.
	• Behebt ein Problem, das zum Absturz des Nucleus führte, wenn eine Service-Load-Ausnahme auftrat.
	• Verbessert die Auflösung von Komponentenabhängigkeiten, sodass eine Bereitstellung fehlschlägt, die eine zirkuläre Abhängigkeit enthält.
	• Behebt ein Problem, das verhinderte, dass eine Plugin-Komponente erneut bereitgestellt werden konnte, wenn diese Komponente zuvor vom Kerngerät entfernt worden war.
	• Behebt ein Problem, das dazu führte, dass die `HOME` Umgebungsvariable auf das */greengrass/v2* `/work` Verzeichnis für Lambda-Komponenten oder für Komponenten gesetzt wurde, die als Root ausgeführt werden. Die `HOME` Variable ist jetzt korrekt auf das Home-Verzeichnis des Benutzers gesetzt, der die Komponente ausführt.
	• Zusätzliche kleinere Korrekturen und Verbesserungen. Weitere Informationen finden Sie in den Veröffentlichungen unter GitHub.

Version	Änderungen
2.0.5	Fehlerkorrekturen und Verbesserungen • Leitet den Datenverkehr beim Herunterladen von AWS bereitgestellten Komponenten korrekt über einen konfigurierten Netzwerk-Proxy weiter. • Verwenden Sie den richtigen Endpunkt der Greengrass-Datenebene in AWS chinesischen Regionen.

Version	Änderungen
2.0.4	Neue Features • Aktiviert HTTPS-Verkehr über Port 443. Sie können den neuen `greengrassDataPlanePort` Konfigurationsparameter für Version 2.0.4 der Nucleus-Komponente verwenden, um die HTTPS-Kommunikation so zu konfigurieren, dass sie über Port 443 statt über den Standardport 8443 übertragen wird. Weitere Informationen finden Sie unter Konfigurieren Sie HTTPS über Port 443. • Fügt die Rezeptvariable für den Arbeitspfad hinzu. Sie können diese Rezeptvariable verwenden, um den Pfad zu den Arbeitsordnern der Komponenten abzurufen, die Sie verwenden können, um Dateien zwischen Komponenten und ihren Abhängigkeiten gemeinsam zu nutzen. Weitere Informationen finden Sie in der Rezeptvariablen für den Arbeitspfad. Fehlerkorrekturen und Verbesserungen • Verhindert die Erstellung der AWS Identity and Access Management Token-Exchange-Rollenrichtlinie (IAM), wenn bereits eine Rollenrichtlinie existiert. Aufgrund dieser Änderung benötigt das Installationsprogramm nun `iam:GetPolicy` und, `sts:GetCallerIdentity` wenn es mit `--provision true` ausgeführt wird. Weitere Informationen finden Sie unter Minimale IAM-Richtlinie für das Installationsprogramm zur Bereitstellung von Ressourcen. • Behandelt die Stornierung einer Bereitstellung, die noch nicht erfolgreich registriert wurde, korrekt. • Aktualisiert die Konfiguration, sodass beim Rollback einer Bereitstellung ältere Einträge mit neueren Zeitstempeln entfernt werden. • Zusätzliche kleinere Korrekturen und Verbesserungen. Weitere Informationen finden Sie in den Veröffentlichungen unter GitHub.
2.0.3	Erste Version

Authentifizierung auf Client-Geräten

Die Authentifizierungskomponente für Clientgeräte (`aws.greengrass.clientdevices.Auth`) authentifiziert Client-Geräte und autorisiert Aktionen auf Client-Geräten.

> **Note**
>
> Client-Geräte sind lokale IoT-Geräte, die eine Verbindung zu einem Greengrass-Core-Gerät herstellen, um MQTT-Nachrichten und Daten zur Verarbeitung zu senden. Weitere Informationen finden Sie unter Interagieren mit lokalen IoT-Geräten.

Themen

Versionen

> **Note**
>
> Version 2.3.0 für die Client-Geräteauthentifizierung wurde eingestellt. Wir empfehlen dringend, auf Version 2.3.1 oder höher für die Client-Geräteauthentifizierung zu aktualisieren.

Diese Komponente hat die folgenden Versionen:

- 2.4.x
- 2.3.x

- 2.2.x
- 2.1.x
- 2.0.x

Typ

Diese Komponente ist eine Plugin-Komponente (`aws.greengrass.plugin`). Der Greengrass-Kern führt diese Komponente in derselben Java Virtual Machine (JVM) wie der Nucleus aus. Der Nucleus wird neu gestartet, wenn Sie die Version dieser Komponente auf dem Kerngerät ändern.

Diese Komponente verwendet dieselbe Protokolldatei wie der Greengrass-Kern. Weitere Informationen finden Sie unter Überwachen von AWS IoT Greengrass Protokollen.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf Kerngeräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Die Greengrass-Servicerolle muss Ihrer zugeordnet sein AWS-Konto und die `iot:DescribeCertificate` Genehmigung zulassen.
- Die AWS IoT Richtlinie des Kerngeräts muss die folgenden Berechtigungen zulassen:
  - `greengrass:GetConnectivityInfo`, wobei die Ressourcen den ARN des Kerngeräts enthalten, auf dem diese Komponente ausgeführt wird
  - `greengrass:VerifyClientDeviceIoTCertificateAssociation`, wobei die Ressourcen den Amazon-Ressourcennamen (ARN) jedes Client-Geräts enthalten, das eine Verbindung zum Kerngerät herstellt
  - `greengrass:VerifyClientDeviceIdentity`

- `greengrass:PutCertificateAuthorities`
- `iot:Publish`, wobei die Ressourcen den ARN des folgenden MQTT-Themas beinhalten:
  - `$aws/things/coreDeviceThingName*-gci/shadow/get`
- `iot:Subscribe`, wobei die Ressourcen die ARNs der folgenden MQTT-Themenfilter enthalten:
  - `$aws/things/coreDeviceThingName*-gci/shadow/update/delta`
  - `$aws/things/coreDeviceThingName*-gci/shadow/get/accepted`
- `iot:Receive`, wobei die Ressourcen die ARNs der folgenden MQTT-Themen beinhalten:
  - `$aws/things/coreDeviceThingName*-gci/shadow/update/delta`
  - `$aws/things/coreDeviceThingName*-gci/shadow/get/accepted`

  Weitere Informationen finden Sie unter AWS IoT-Richtlinien für Operationen auf Datenebene und Minimale AWS IoT Richtlinie zur Unterstützung von Client-Geräten.

- (Optional) Um die Offline-Authentifizierung zu verwenden, muss die vom AWS IoT Greengrass Dienst verwendete AWS Identity and Access Management (IAM-) Rolle die folgende Berechtigung enthalten:
  - `greengrass:ListClientDevicesAssociatedWithCoreDevice`um es dem Kerngerät zu ermöglichen, Clients für die Offline-Authentifizierung aufzulisten.
- Die Authentifizierungskomponente für Clientgeräte wird für die Ausführung in einer VPC unterstützt. Um diese Komponente in einer VPC bereitzustellen, ist Folgendes erforderlich.
  - Die Authentifizierungskomponente für das Client-Gerät muss über Konnektivität zu AWS IoT data, AWS IoT Anmeldeinformationen und Amazon S3 verfügen.

Endpunkte und Ports

Diese Komponente muss in der Lage sein, ausgehende Anfragen an die folgenden Endpunkte und Ports auszuführen, zusätzlich zu den Endpunkten und Ports, die für den Basisbetrieb erforderlich sind. Weitere Informationen finden Sie unter Zulassen von Gerätedatenverkehr über einen Proxy oder eine Firewall.

Endpunkt	Port	Erforderlich	Beschreibung
`iot.region.amazonaws.com`	443	Ja	Wird verwendet

Endpunkt	Port	Erforderlich	Beschreibung
			, um Informationen über AWS IoT Ding-Zertifikate abzurufen.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt sie AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und all ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitstellen zu können. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente sowie die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste der Abhängigkeiten.

2.4.4

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.4.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.6.0 <2.13.0	Weich

2.4.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.4.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.6.0 <2.12.0	Weich

2.4.1 and 2.4.2

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.4.1 und 2.4.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.6.0 <2.11.0	Weich

2.3.0 – 2.4.0

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.3.0 bis 2.4.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.6.0 <2.10.0	Weich

2.3.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.3.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.6.0 <2.10.0	Weich

2.2.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.2.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2,6,0 <=2,9,0	Weich

2.2.2

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.2.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2,6,0 <=2,8,0	Weich

2.2.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.2.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2,6,0 <2,8,0	Weich

2.2.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.2.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.6.0 <2.7.0	Weich

2.1.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.2.0 <2.7.0	Weich

2.0.4

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.2.0 <2.6.0	Weich

2.0.2 and 2.0.3

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.0.2 und 2.0.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.2.0 <2.5.0	Weich

2.0.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.2.0 <2.4.0	Weich

2.0.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.2.0 <2.3.0	Weich

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zu den Komponentenrezepten.

Konfiguration

Diese Komponente stellt die folgenden Konfigurationsparameter bereit, die Sie bei der Bereitstellung der Komponente anpassen können.

> **Note**
>
> Die Abonnementberechtigung wird während einer Client-Abonnementanfrage an den lokalen MQTT-Broker ausgewertet. Wenn die bestehende Abonnementberechtigung des Kunden

widerrufen wird, kann der Client ein Thema nicht mehr abonnieren. Er wird jedoch weiterhin Nachrichten zu Themen empfangen, die zuvor abonniert wurden. Um dieses Verhalten zu verhindern, sollte der lokale MQTT-Broker nach dem Widerruf der Abonnementberechtigung neu gestartet werden, um die erneute Autorisierung von Clients zu erzwingen.
Aktualisieren Sie für die Komponente MQTT 5-Broker (EMQX) die `restartIdentifier` Konfiguration, um den MQTT 5-Broker neu zu starten. Weitere Informationen finden Sie in der Konfiguration der MQTT 5-Broker-Komponente.
Die MQTT 3.1.1-Broker-Komponente (Moquette) wird standardmäßig wöchentlich neu gestartet, wenn sich das Serverzertifikat ändert, was die Clients zur erneuten Autorisierung zwingt. Sie können einen Neustart erzwingen, indem Sie entweder die Konnektivitätsinformationen (IP-Adressen) des Kerngeräts ändern oder indem Sie eine Einrichtung vornehmen, um die Broker-Komponente zu entfernen und sie später erneut bereitzustellen.

v2.5.0

`deviceGroups`

Gerätegruppen sind Gruppen von Clientgeräten, die berechtigt sind, eine Verbindung zu einem Kerngerät herzustellen und mit diesem zu kommunizieren. Verwenden Sie Auswahlregeln, um Gruppen von Client-Geräten zu identifizieren, und definieren Sie Autorisierungsrichtlinien für Client-Geräte, die die Berechtigungen für jede Gerätegruppe spezifizieren.

Dieses Objekt enthält die folgenden Informationen:

`formatVersion`

Die Formatversion für dieses Konfigurationsobjekt.

Wählen Sie aus den folgenden Optionen aus:

- `2021-03-05`

`definitions`

Die Gerätegruppen für dieses Kerngerät. Jede Definition gibt eine Auswahlregel an, mit der bewertet wird, ob ein Client-Gerät Mitglied der Gruppe ist. Jede Definition gibt auch die Berechtigungsrichtlinie an, die auf Client-Geräte angewendet werden soll, die der Auswahlregel entsprechen. Wenn ein Client-Gerät Mitglied mehrerer Gerätegruppen ist, setzen sich die Berechtigungen des Geräts aus den Berechtigungsrichtlinien der einzelnen Gruppen zusammen.

Dieses Objekt enthält die folgenden Informationen:

groupNameKey

Der Name dieser Gerätegruppe. *groupNameKey*Ersetzen Sie ihn durch einen Namen, der Ihnen hilft, diese Gerätegruppe zu identifizieren.

Dieses Objekt enthält die folgenden Informationen:

`selectionRule`

Die Abfrage, die angibt, welche Client-Geräte Mitglieder dieser Gerätegruppe sind. Wenn ein Client-Gerät eine Verbindung herstellt, wertet das Core-Gerät diese Auswahlregel aus, um festzustellen, ob das Client-Gerät Mitglied dieser Gerätegruppe ist. Wenn das Client-Gerät Mitglied ist, verwendet das Kerngerät die Richtlinie dieser Gerätegruppe, um die Aktionen des Client-Geräts zu autorisieren.

Jede Auswahlregel umfasst mindestens eine Auswahlregelklausel, bei der es sich um eine Abfrage mit einem einzelnen Ausdruck handelt, die auf Client-Geräte zutreffen kann. Auswahlregeln verwenden dieselbe Abfragesyntax wie die AWS IoT Flottenindizierung. Weitere Informationen zur Syntax von Auswahlregeln finden Sie unter Abfragesyntax für die AWS IoT Flottenindizierung im AWS IoT Core Entwicklerhandbuch.

Verwenden Sie den * Platzhalter, um mehreren Client-Geräten eine Auswahlregelklausel zuzuordnen. Sie können diesen Platzhalter am Anfang und Ende des Dingnamens verwenden, um nach Client-Geräten zu suchen, deren Namen mit der von Ihnen angegebenen Zeichenfolge beginnen oder enden. Sie können diesen Platzhalter auch verwenden, um alle Client-Geräte abzugleichen.

> **Note**
>
> Um einen Wert auszuwählen, der einen Doppelpunkt (`:`) enthält, maskieren Sie den Doppelpunkt mit einem umgekehrten Schrägstrich (). \ In Formaten wie JSON müssen Sie umgekehrte Schrägstriche maskieren, sodass Sie vor dem Doppelpunkt zwei umgekehrte Schrägstriche eingeben. Geben Sie beispielsweise an, dass Sie ein Ding auswählen `thingName: MyTeam\\:ClientDevice1` möchten, dessen Name lautet. `MyTeam:ClientDevice1`

Sie können den folgenden Selektor angeben:

- `thingName`— Der Name des Dings eines Client-Geräts. AWS IoT

Example Beispiel für eine Auswahlregel

Die folgende Auswahlregel entspricht Client-Geräten mit dem Namen `MyClientDevice1` oder`MyClientDevice2`.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example Beispiel für eine Auswahlregel (verwenden Sie Platzhalter)

Die folgende Auswahlregel entspricht Client-Geräten, deren Namen mit `MyClientDevice` beginnen.

```
thingName: MyClientDevice*
```

Example Beispiel für eine Auswahlregel (verwenden Sie Platzhalter)

Die folgende Auswahlregel entspricht Client-Geräten, deren Namen auf enden`MyClientDevice`.

```
thingName: *MyClientDevice
```

Example Beispiel für eine Auswahlregel (entspricht allen Geräten)

Die folgende Auswahlregel gilt für alle Client-Geräte.

```
thingName: *
```

`policyName`

Die Berechtigungsrichtlinie, die für Client-Geräte in dieser Gerätegruppe gilt. Geben Sie den Namen einer Richtlinie an, die Sie im `policies` Objekt definieren.

`policies`

Die Autorisierungsrichtlinien für Client-Geräte für Client-Geräte, die eine Verbindung zum Kerngerät herstellen. Jede Autorisierungsrichtlinie spezifiziert eine Reihe von Aktionen und die Ressourcen, über die ein Client-Gerät diese Aktionen ausführen kann.

Dieses Objekt enthält die folgenden Informationen:

policyNameKey

Der Name dieser Autorisierungsrichtlinie. *policyNameKey*Ersetzen Sie ihn durch einen Namen, anhand dessen Sie diese Autorisierungsrichtlinie leichter identifizieren können. Sie verwenden diesen Richtliniennamen, um zu definieren, welche Richtlinie für eine Gerätegruppe gilt.

Dieses Objekt enthält die folgenden Informationen:

statementNameKey

Der Name dieser Richtlinienerklärung. *statementNameKey*Ersetzen Sie ihn durch einen Namen, anhand dessen Sie diese Grundsatzerklärung leichter identifizieren können.

Dieses Objekt enthält die folgenden Informationen:

`operations`

Die Liste der Vorgänge, bei denen die Ressourcen in dieser Richtlinie berücksichtigt werden sollen.

Sie können jede der folgenden Operationen einbeziehen:

- `mqtt:connect`— Erteilt die Erlaubnis, eine Verbindung zum Kerngerät herzustellen. Client-Geräte müssen über diese Berechtigung verfügen, um eine Verbindung zu einem Kerngerät herzustellen.

  Dieser Vorgang unterstützt die folgenden Ressourcen:

  - `mqtt:clientId:`*deviceClientId*— Beschränken Sie den Zugriff auf der Grundlage der Client-ID, die ein Client-Gerät verwendet, um eine Verbindung zum MQTT-Broker des Kerngeräts herzustellen. Durch die *deviceClientId*zu verwendende Client-ID ersetzen.
- `mqtt:publish`— Erteilt die Erlaubnis, MQTT-Nachrichten zu Themen zu veröffentlichen.

  Dieser Vorgang unterstützt die folgenden Ressourcen:

  - `mqtt:topic:`*mqttTopic*— Beschränken Sie den Zugriff auf der Grundlage des MQTT-Themas, in dem ein Client-Gerät eine Nachricht veröffentlicht. Ersetzen Sie *MQTTTopic durch das* zu verwendende Thema.

Diese Ressource unterstützt keine Platzhalter für MQTT-Themen.

- `mqtt:subscribe`— Erteilt die Erlaubnis, MQTT-Themenfilter zum Empfangen von Nachrichten zu abonnieren.

  Dieser Vorgang unterstützt die folgenden Ressourcen:

  - `mqtt:topicfilter:`*`mqttTopicFilter`*— Beschränken Sie den Zugriff auf der Grundlage der MQTT-Themen, bei denen ein Client-Gerät Nachrichten abonnieren kann. *`mqttTopicFilter`*Ersetzen Sie es durch den zu verwendenden Themenfilter.

    Diese Ressource unterstützt die Platzhalter + und # MQTT-Themen-Platzhalter. Weitere Informationen finden Sie unter MQTT-Themen im Developer Guide.AWS IoT Core

    Das Client-Gerät kann genau die Themenfilter abonnieren, die Sie zulassen. Wenn Sie dem Client-Gerät beispielsweise erlauben, die `mqtt:topicfilter:client/+/status` Ressource zu abonnieren, kann das Client-Gerät `client/+/status` zwar abonnieren, aber nicht`client/client1/status`.

Sie können den * Platzhalter angeben, um den Zugriff auf alle Aktionen zu ermöglichen.

`resources`

Die Liste der Ressourcen, die für die Operationen in dieser Richtlinie zugelassen werden sollen. Geben Sie Ressourcen an, die den Vorgängen in dieser Richtlinie entsprechen. Sie könnten beispielsweise eine Liste von MQTT-Themenressourcen (`mqtt:topic:`*`mqttTopic`*) in einer Richtlinie angeben, die den `mqtt:publish` Vorgang spezifiziert.

Sie können den * Platzhalter an einer beliebigen Stelle in der Ressourcenvariablen angeben, um den Zugriff auf alle Ressourcen zu ermöglichen. Sie können beispielsweise angeben**`mqtt:topic:my*`**, dass der Zugriff auf Ressourcen zulässig ist, die dieser Eingabe entsprechen.

Die folgende Ressourcenvariable wird unterstützt:

- `mqtt:topic:${iot:Connection.Thing.ThingName}`

Dies ergibt den Namen der Sache in der AWS IoT Core Registrierung, für die die Richtlinie ausgewertet wird. AWS IoT Core verwendet das Zertifikat, das das Gerät bei der Authentifizierung vorlegt, um zu ermitteln, welches Objekt zur Überprüfung der Verbindung verwendet werden soll. Diese Richtlinienvariable ist nur verfügbar, wenn ein Gerät eine Verbindung über MQTT oder MQTT über das Protokoll herstellt. WebSocket

`statementDescription`

(Optional) Eine Beschreibung für diese Richtlinienerklärung.

`certificates`

(Optional) Die Zertifikatkonfigurationsoptionen für dieses Kerngerät. Dieses Objekt enthält die folgenden Informationen:

`serverCertificateValiditySeconds`

(Optional) Die Zeitspanne (in Sekunden), nach der das lokale MQTT-Serverzertifikat abläuft. Sie können diese Option konfigurieren, um festzulegen, wie oft Client-Geräte die Verbindung zum Kerngerät trennen und wieder herstellen.

Diese Komponente rotiert das lokale MQTT-Serverzertifikat 24 Stunden vor seinem Ablauf. Der MQTT-Broker, wie die Moquette MQTT-Broker-Komponente, generiert ein neues Zertifikat und startet neu. In diesem Fall werden alle mit diesem Kerngerät verbundenen Client-Geräte getrennt. Client-Geräte können nach kurzer Zeit wieder eine Verbindung zum Kerngerät herstellen.

Standard: `604800` (7 Tage)

Mindestwert: `172800` (2 Tage)

Höchstwert: `864000` (10 Tage)

`performance`

(Optional) Die Leistungskonfigurationsoptionen für dieses Kerngerät. Dieses Objekt enthält die folgenden Informationen:

`maxActiveAuthTokens`

(Optional) Die maximale Anzahl aktiver Autorisierungstoken für Client-Geräte. Sie können diese Anzahl erhöhen, damit eine größere Anzahl von Client-Geräten eine Verbindung zu einem Single-Core-Gerät herstellen kann, ohne sie erneut authentifizieren zu müssen.

Standard: `2500`

`cloudRequestQueueSize`

(Optional) Die maximale Anzahl von AWS Cloud Anfragen, die in die Warteschlange gestellt werden müssen, bevor diese Komponente Anfragen ablehnt.

Standard: `100`

`maxConcurrentCloudRequests`

(Optional) Die maximale Anzahl gleichzeitiger Anfragen, die an die gesendet werden sollen. AWS Cloud Sie können diese Zahl erhöhen, um die Authentifizierungsleistung auf Kerngeräten zu verbessern, auf denen Sie eine große Anzahl von Client-Geräten verbinden.

Standard: 1

`certificateAuthority`

(Optional) Konfigurationsoptionen für Zertifizierungsstellen, um die zwischengeschaltete Zertifizierungsstelle des Kerngeräts durch Ihre eigene Zwischenzertifizierungsstelle zu ersetzen.

> **Note**
>
> Wenn Sie Ihr Greengrass-Core-Gerät mit einer benutzerdefinierten Zertifizierungsstelle (CA) konfigurieren und dieselbe Zertifizierungsstelle verwenden, um Client-Gerätezertifikate auszustellen, umgeht Greengrass Autorisierungsrichtlinienprüfungen für MQTT-Operationen auf Client-Geräten. Die Authentifizierungskomponente für das Client-Gerät vertraut voll und ganz auf Clients, die Zertifikate verwenden, die von der Zertifizierungsstelle signiert wurden, für deren Verwendung sie konfiguriert ist.
> Um dieses Verhalten bei der Verwendung einer benutzerdefinierten Zertifizierungsstelle einzuschränken, erstellen und signieren Sie Client-Geräte, die eine andere Zertifizierungsstelle oder Zwischenzertifizierungsstelle verwenden, und passen Sie dann die `certificateChainUri` Felder `certificateUri` und so an, dass sie auf die richtige Zwischenzertifizierungsstelle verweisen.

Dieses Objekt enthält die folgenden Informationen.

Uri des Zertifikats

Der Speicherort des Zertifikats. Dabei kann es sich um einen Dateisystem-URI oder einen URI handeln, der auf ein Zertifikat verweist, das in einem Hardware-Sicherheitsmodul gespeichert ist.

`certificateChainUri`

Der Speicherort der Zertifikatskette für die CA des Kerngeräts. Dies sollte die komplette Zertifikatskette bis zu Ihrer Stammzertifizierungsstelle sein. Dabei kann es sich um einen Dateisystem-URI oder einen URI handeln, der auf eine Zertifikatskette verweist, die in einem Hardware-Sicherheitsmodul gespeichert ist.

`privateKeyUri`

Der Speicherort des privaten Schlüssels des Kerngeräts. Dies kann ein Dateisystem-URI oder ein URI sein, der auf einen privaten Schlüssel eines Zertifikats verweist, der in einem Hardware-Sicherheitsmodul gespeichert ist.

`security`

(Optional) Sicherheitskonfigurationsoptionen für dieses Kerngerät. Dieses Objekt enthält die folgenden Informationen.

`clientDeviceTrustDurationMinutes`

Die Dauer in Minuten, während der die Authentifizierungsinformationen eines Client-Geräts als vertrauenswürdig eingestuft werden können, bevor eine erneute Authentifizierung beim Kerngerät erforderlich ist. Der Standardwert lautet 1.

`metrics`

(Optional) Die Metrikoptionen für dieses Kerngerät. Fehlermetriken werden nur angezeigt, wenn bei der Authentifizierung des Client-Geräts ein Fehler auftritt. Dieses Objekt enthält die folgenden Informationen:

`disableMetrics`

Wenn das `disableMetrics` Feld auf gesetzt ist`true`, erfasst die Authentifizierung auf dem Client-Gerät keine Metriken.

Standard: `false`

`aggregatePeriodSeconds`

Der Aggregationszeitraum in Sekunden, der bestimmt, wie oft die Authentifizierung auf dem Client-Gerät Messwerte aggregiert und an den Telemetrieagenten sendet. Dies ändert nichts daran, wie oft Metriken veröffentlicht werden, da der Telemetrieagent sie immer noch einmal täglich veröffentlicht.

Standard: `3600`

startupTimeoutSeconds

(Optional) Die maximale Zeit in Sekunden für den Start der Komponente. Der Status der Komponente ändert sich auf, `BROKEN` wenn dieser Timeout überschritten wird.

Standard: `120`

Example Beispiel: Aktualisierung der Zusammenführung von Konfigurationen (unter Verwendung einer restriktiven Richtlinie)

In der folgenden Beispielkonfiguration wird festgelegt, dass Client-Geräte, deren Namen mit beginnen, eine Verbindung herstellen und Informationen `MyClientDevice` zu allen Themen veröffentlichen/abonnieren dürfen.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
```

```
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example Beispiel: Update zur Zusammenführung von Konfigurationen (unter Verwendung einer permissiven Richtlinie)

In der folgenden Beispielkonfiguration wird festgelegt, dass alle Client-Geräte eine Verbindung herstellen und Informationen zu allen Themen veröffentlichen/abonnieren können.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
```

```
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

Example Beispiel: Aktualisierung zur Zusammenführung von Konfigurationen (unter Verwendung einer Richtlinie für einen Dingnamen)

Die folgende Beispielkonfiguration ermöglicht es Client-Geräten, Beiträge zu Themen zu veröffentlichen, die mit dem Ding-Namen des Client-Geräts beginnen und mit der Zeichenfolge end`topic`.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "myThing": {
        "selectionRule": "thingName: *",
        "policyName": "MyThingNamePolicy"
      }
    },
    "policies": {
      "MyThingNamePolicy": {
        "policyStatement": {
          "statementDescription": "mqtt publish",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:${iot:Connection.Thing.ThingName}/*/topic"
          ]
        }
      }
    }
  }
```

```
}
```

v2.4.5

`deviceGroups`

Gerätegruppen sind Gruppen von Clientgeräten, die berechtigt sind, eine Verbindung zu einem Kerngerät herzustellen und mit diesem zu kommunizieren. Verwenden Sie Auswahlregeln, um Gruppen von Client-Geräten zu identifizieren, und definieren Sie Autorisierungsrichtlinien für Client-Geräte, die die Berechtigungen für jede Gerätegruppe spezifizieren.

Dieses Objekt enthält die folgenden Informationen:

`formatVersion`

Die Formatversion für dieses Konfigurationsobjekt.

Wählen Sie aus den folgenden Optionen aus:

- `2021-03-05`

`definitions`

Die Gerätegruppen für dieses Kerngerät. Jede Definition gibt eine Auswahlregel an, mit der bewertet wird, ob ein Client-Gerät Mitglied der Gruppe ist. Jede Definition gibt auch die Berechtigungsrichtlinie an, die auf Client-Geräte angewendet werden soll, die der Auswahlregel entsprechen. Wenn ein Client-Gerät Mitglied mehrerer Gerätegruppen ist, setzen sich die Berechtigungen des Geräts aus den Berechtigungsrichtlinien der einzelnen Gruppen zusammen.

Dieses Objekt enthält die folgenden Informationen:

`groupNameKey`

Der Name dieser Gerätegruppe. *`groupNameKey`*Ersetzen Sie ihn durch einen Namen, der Ihnen hilft, diese Gerätegruppe zu identifizieren.

Dieses Objekt enthält die folgenden Informationen:

`selectionRule`

Die Abfrage, die angibt, welche Client-Geräte Mitglieder dieser Gerätegruppe sind. Wenn ein Client-Gerät eine Verbindung herstellt, wertet das Core-Gerät diese Auswahlregel aus, um festzustellen, ob das Client-Gerät Mitglied dieser

Gerätegruppe ist. Wenn das Client-Gerät Mitglied ist, verwendet das Kerngerät die Richtlinie dieser Gerätegruppe, um die Aktionen des Client-Geräts zu autorisieren.

Jede Auswahlregel umfasst mindestens eine Auswahlregelklausel, bei der es sich um eine Abfrage mit einem einzelnen Ausdruck handelt, die auf Client-Geräte zutreffen kann. Auswahlregeln verwenden dieselbe Abfragesyntax wie die AWS IoT Flottenindizierung. Weitere Informationen zur Syntax von Auswahlregeln finden Sie unter Abfragesyntax für die AWS IoT Flottenindizierung im AWS IoT Core Entwicklerhandbuch.

Verwenden Sie den * Platzhalter, um mehreren Client-Geräten eine Auswahlregelklausel zuzuordnen. Sie können diesen Platzhalter am Anfang und Ende des Dingnamens verwenden, um nach Client-Geräten zu suchen, deren Namen mit der von Ihnen angegebenen Zeichenfolge beginnen oder enden. Sie können diesen Platzhalter auch verwenden, um alle Client-Geräte abzugleichen.

> **Note**
>
> Um einen Wert auszuwählen, der einen Doppelpunkt (`:`) enthält, maskieren Sie den Doppelpunkt mit einem umgekehrten Schrägstrich (). \ In Formaten wie JSON müssen Sie umgekehrte Schrägstriche maskieren, sodass Sie vor dem Doppelpunkt zwei umgekehrte Schrägstriche eingeben. Geben Sie beispielsweise an, dass Sie ein Ding auswählen `thingName: MyTeam\\:ClientDevice1` möchten, dessen Name lautet. `MyTeam:ClientDevice1`

Sie können den folgenden Selektor angeben:

- `thingName`— Der Name des Dings eines Client-Geräts. AWS IoT

Example Beispiel für eine Auswahlregel

Die folgende Auswahlregel entspricht Client-Geräten mit dem Namen `MyClientDevice1` oder`MyClientDevice2`.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example Beispiel für eine Auswahlregel (verwenden Sie Platzhalter)

Die folgende Auswahlregel entspricht Client-Geräten, deren Namen mit `MyClientDevice` beginnen.

```
thingName: MyClientDevice*
```

Example Beispiel für eine Auswahlregel (verwenden Sie Platzhalter)

Die folgende Auswahlregel entspricht Client-Geräten, deren Namen auf endenMyClientDevice.

```
thingName: *MyClientDevice
```

Example Beispiel für eine Auswahlregel (entspricht allen Geräten)

Die folgende Auswahlregel gilt für alle Client-Geräte.

```
thingName: *
```

`policyName`

Die Berechtigungsrichtlinie, die für Client-Geräte in dieser Gerätegruppe gilt. Geben Sie den Namen einer Richtlinie an, die Sie im `policies` Objekt definieren.

`policies`

Die Autorisierungsrichtlinien für Client-Geräte für Client-Geräte, die eine Verbindung zum Kerngerät herstellen. Jede Autorisierungsrichtlinie spezifiziert eine Reihe von Aktionen und die Ressourcen, über die ein Client-Gerät diese Aktionen ausführen kann.

Dieses Objekt enthält die folgenden Informationen:

`policyNameKey`

Der Name dieser Autorisierungsrichtlinie. *`policyNameKey`*Ersetzen Sie ihn durch einen Namen, anhand dessen Sie diese Autorisierungsrichtlinie leichter identifizieren können. Sie verwenden diesen Richtliniennamen, um zu definieren, welche Richtlinie für eine Gerätegruppe gilt.

Dieses Objekt enthält die folgenden Informationen:

statementNameKey

Der Name dieser Richtlinienerklärung. *statementNameKey*Ersetzen Sie ihn durch einen Namen, anhand dessen Sie diese Grundsatzerklärung leichter identifizieren können.

Dieses Objekt enthält die folgenden Informationen:

`operations`

Die Liste der Vorgänge, bei denen die Ressourcen in dieser Richtlinie berücksichtigt werden sollen.

Sie können jede der folgenden Operationen einbeziehen:

- `mqtt:connect`— Erteilt die Erlaubnis, eine Verbindung zum Kerngerät herzustellen. Client-Geräte müssen über diese Berechtigung verfügen, um eine Verbindung zu einem Kerngerät herzustellen.

  Dieser Vorgang unterstützt die folgenden Ressourcen:

  - `mqtt:clientId:`*deviceClientId*— Beschränken Sie den Zugriff auf der Grundlage der Client-ID, die ein Client-Gerät verwendet, um eine Verbindung zum MQTT-Broker des Kerngeräts herzustellen. Durch die *deviceClientId*zu verwendende Client-ID ersetzen.
- `mqtt:publish`— Erteilt die Erlaubnis, MQTT-Nachrichten zu Themen zu veröffentlichen.

  Dieser Vorgang unterstützt die folgenden Ressourcen:

  - `mqtt:topic:`*mqttTopic*— Beschränken Sie den Zugriff auf der Grundlage des MQTT-Themas, in dem ein Client-Gerät eine Nachricht veröffentlicht. Ersetzen Sie *MQTTTopic durch das* zu verwendende Thema.

    Diese Ressource unterstützt keine Platzhalter für MQTT-Themen.
- `mqtt:subscribe`— Erteilt die Erlaubnis, MQTT-Themenfilter zum Empfangen von Nachrichten zu abonnieren.

  Dieser Vorgang unterstützt die folgenden Ressourcen:

  - `mqtt:topicfilter:`*mqttTopicFilter*— Beschränken Sie den Zugriff auf der Grundlage der MQTT-Themen, bei denen ein Client-Gerät

Nachrichten abonnieren kann. *`mqttTopicFilter`*Ersetzen Sie es durch den zu verwendenden Themenfilter.

Diese Ressource unterstützt die Platzhalter + und # MQTT-Themen-Platzhalter. Weitere Informationen finden Sie unter MQTT-Themen im Developer Guide.AWS IoT Core

Das Client-Gerät kann genau die Themenfilter abonnieren, die Sie zulassen. Wenn Sie dem Client-Gerät beispielsweise erlauben, die `mqtt:topicfilter:client/+/status` Ressource zu abonnieren, kann das Client-Gerät `client/+/status` zwar abonnieren, aber nicht`client/client1/status`.

Sie können den * Platzhalter angeben, um den Zugriff auf alle Aktionen zu ermöglichen.

`resources`

Die Liste der Ressourcen, die für die Operationen in dieser Richtlinie zugelassen werden sollen. Geben Sie Ressourcen an, die den Vorgängen in dieser Richtlinie entsprechen. Sie könnten beispielsweise eine Liste von MQTT-Themenressourcen (`mqtt:topic:`*`mqttTopic`*) in einer Richtlinie angeben, die den `mqtt:publish` Vorgang spezifiziert.

Sie können den * Platzhalter angeben, um den Zugriff auf alle Ressourcen zu ermöglichen. Sie können den * Platzhalter nicht verwenden, um unvollständige Ressourcen-IDs abzugleichen. Sie können beispielsweise angeben**`"resources": "*"`**, aber Sie können nicht angeben. **`"resources": "mqtt:clientId:*"`**

`statementDescription`

(Optional) Eine Beschreibung für diese Richtlinienerklärung.

`certificates`

(Optional) Die Zertifikatkonfigurationsoptionen für dieses Kerngerät. Dieses Objekt enthält die folgenden Informationen:

`serverCertificateValiditySeconds`

(Optional) Die Zeitspanne (in Sekunden), nach der das lokale MQTT-Serverzertifikat abläuft. Sie können diese Option konfigurieren, um festzulegen, wie oft Client-Geräte die Verbindung zum Kerngerät trennen und wieder herstellen.

Diese Komponente rotiert das lokale MQTT-Serverzertifikat 24 Stunden vor seinem Ablauf. Der MQTT-Broker, wie die Moquette MQTT-Broker-Komponente, generiert ein neues Zertifikat und startet neu. In diesem Fall werden alle mit diesem Kerngerät verbundenen Client-Geräte getrennt. Client-Geräte können nach kurzer Zeit wieder eine Verbindung zum Kerngerät herstellen.

Standard: `604800` (7 Tage)

Mindestwert: `172800` (2 Tage)

Höchstwert: `864000` (10 Tage)

`performance`

(Optional) Die Leistungskonfigurationsoptionen für dieses Kerngerät. Dieses Objekt enthält die folgenden Informationen:

`maxActiveAuthTokens`

(Optional) Die maximale Anzahl aktiver Autorisierungstoken für Client-Geräte. Sie können diese Anzahl erhöhen, damit eine größere Anzahl von Client-Geräten eine Verbindung zu einem Single-Core-Gerät herstellen kann, ohne sie erneut authentifizieren zu müssen.

Standard: `2500`

`cloudRequestQueueSize`

(Optional) Die maximale Anzahl von AWS Cloud Anfragen, die in die Warteschlange gestellt werden müssen, bevor diese Komponente Anfragen ablehnt.

Standard: `100`

`maxConcurrentCloudRequests`

(Optional) Die maximale Anzahl gleichzeitiger Anfragen, die an die gesendet werden sollen. AWS Cloud Sie können diese Zahl erhöhen, um die Authentifizierungsleistung

auf Kerngeräten zu verbessern, auf denen Sie eine große Anzahl von Client-Geräten verbinden.

Standard: 1

`certificateAuthority`

(Optional) Konfigurationsoptionen für Zertifizierungsstellen, um die zwischengeschaltete Zertifizierungsstelle des Kerngeräts durch Ihre eigene Zwischenzertifizierungsstelle zu ersetzen.

> **Note**
>
> Wenn Sie Ihr Greengrass-Core-Gerät mit einer benutzerdefinierten Zertifizierungsstelle (CA) konfigurieren und dieselbe Zertifizierungsstelle verwenden, um Client-Gerätezertifikate auszustellen, umgeht Greengrass Autorisierungsrichtlinienprüfungen für MQTT-Operationen auf Client-Geräten. Die Authentifizierungskomponente für das Client-Gerät vertraut voll und ganz auf Clients, die Zertifikate verwenden, die von der Zertifizierungsstelle signiert wurden, für deren Verwendung sie konfiguriert ist.
> Um dieses Verhalten bei der Verwendung einer benutzerdefinierten Zertifizierungsstelle einzuschränken, erstellen und signieren Sie Client-Geräte, die eine andere Zertifizierungsstelle oder Zwischenzertifizierungsstelle verwenden, und passen Sie dann die `certificateChainUri` Felder `certificateUri` und so an, dass sie auf die richtige Zwischenzertifizierungsstelle verweisen.

Dieses Objekt enthält die folgenden Informationen.

Uri des Zertifikats

Der Speicherort des Zertifikats. Dabei kann es sich um einen Dateisystem-URI oder einen URI handeln, der auf ein Zertifikat verweist, das in einem Hardware-Sicherheitsmodul gespeichert ist.

`certificateChainUri`

Der Speicherort der Zertifikatskette für die CA des Kerngeräts. Dies sollte die komplette Zertifikatskette bis zu Ihrer Stammzertifizierungsstelle sein. Dabei kann es sich um einen Dateisystem-URI oder einen URI handeln, der auf eine Zertifikatskette verweist, die in einem Hardware-Sicherheitsmodul gespeichert ist.

`privateKeyUri`

Der Speicherort des privaten Schlüssels des Kerngeräts. Dies kann ein Dateisystem-URI oder ein URI sein, der auf einen privaten Schlüssel eines Zertifikats verweist, der in einem Hardware-Sicherheitsmodul gespeichert ist.

`security`

(Optional) Sicherheitskonfigurationsoptionen für dieses Kerngerät. Dieses Objekt enthält die folgenden Informationen.

`clientDeviceTrustDurationMinutes`

Die Dauer in Minuten, während der die Authentifizierungsinformationen eines Client-Geräts als vertrauenswürdig eingestuft werden können, bevor eine erneute Authentifizierung beim Kerngerät erforderlich ist. Der Standardwert lautet 1.

`metrics`

(Optional) Die Metrikoptionen für dieses Kerngerät. Fehlermetriken werden nur angezeigt, wenn bei der Authentifizierung des Client-Geräts ein Fehler auftritt. Dieses Objekt enthält die folgenden Informationen:

`disableMetrics`

Wenn das `disableMetrics` Feld auf gesetzt ist`true`, erfasst die Authentifizierung auf dem Client-Gerät keine Metriken.

Standard: `false`

`aggregatePeriodSeconds`

Der Aggregationszeitraum in Sekunden, der bestimmt, wie oft die Authentifizierung auf dem Client-Gerät Messwerte aggregiert und an den Telemetrieagenten sendet. Dies ändert nichts daran, wie oft Metriken veröffentlicht werden, da der Telemetrieagent sie immer noch einmal täglich veröffentlicht.

Standard: `3600`

startupTimeoutSeconds

(Optional) Die maximale Zeit in Sekunden für den Start der Komponente. Der Status der Komponente ändert sich auf, `BROKEN` wenn dieser Timeout überschritten wird.

Standard: `120`

Example Beispiel: Aktualisierung der Zusammenführung von Konfigurationen (unter Verwendung einer restriktiven Richtlinie)

In der folgenden Beispielkonfiguration wird festgelegt, dass Client-Geräte, deren Namen mit beginnen, eine Verbindung herstellen und Informationen `MyClientDevice` zu allen Themen veröffentlichen/abonnieren dürfen.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
```

```
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example Beispiel: Update zur Zusammenführung von Konfigurationen (unter Verwendung einer permissiven Richtlinie)

In der folgenden Beispielkonfiguration wird festgelegt, dass alle Client-Geräte eine Verbindung herstellen und Informationen zu allen Themen veröffentlichen/abonnieren können.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

v2.4.2 - v2.4.4

`deviceGroups`

Gerätegruppen sind Gruppen von Client-Geräten, die berechtigt sind, eine Verbindung zu einem Kerngerät herzustellen und mit diesem zu kommunizieren. Verwenden Sie Auswahlregeln, um Gruppen von Client-Geräten zu identifizieren, und definieren Sie Autorisierungsrichtlinien für Client-Geräte, die die Berechtigungen für jede Gerätegruppe spezifizieren.

Dieses Objekt enthält die folgenden Informationen:

`formatVersion`

Die Formatversion für dieses Konfigurationsobjekt.

Wählen Sie aus den folgenden Optionen aus:

- `2021-03-05`

`definitions`

Die Gerätegruppen für dieses Kerngerät. Jede Definition gibt eine Auswahlregel an, mit der bewertet wird, ob ein Client-Gerät Mitglied der Gruppe ist. Jede Definition gibt auch die Berechtigungsrichtlinie an, die auf Client-Geräte angewendet werden soll, die der Auswahlregel entsprechen. Wenn ein Client-Gerät Mitglied mehrerer Gerätegruppen ist, setzen sich die Berechtigungen des Geräts aus den Berechtigungsrichtlinien der einzelnen Gruppen zusammen.

Dieses Objekt enthält die folgenden Informationen:

`groupNameKey`

Der Name dieser Gerätegruppe. *`groupNameKey`*Ersetzen Sie ihn durch einen Namen, der Ihnen hilft, diese Gerätegruppe zu identifizieren.

Dieses Objekt enthält die folgenden Informationen:

`selectionRule`

Die Abfrage, die angibt, welche Client-Geräte Mitglieder dieser Gerätegruppe sind. Wenn ein Client-Gerät eine Verbindung herstellt, wertet das Core-Gerät diese Auswahlregel aus, um festzustellen, ob das Client-Gerät Mitglied dieser Gerätegruppe ist. Wenn das Client-Gerät Mitglied ist, verwendet das Kerngerät die Richtlinie dieser Gerätegruppe, um die Aktionen des Client-Geräts zu autorisieren.

Jede Auswahlregel umfasst mindestens eine Auswahlregelklausel, bei der es sich um eine Abfrage mit einem einzelnen Ausdruck handelt, die auf Client-Geräte zutreffen kann. Auswahlregeln verwenden dieselbe Abfragesyntax wie die AWS IoT Flottenindizierung. Weitere Informationen zur Syntax von Auswahlregeln finden Sie unter Abfragesyntax für die AWS IoT Flottenindizierung im AWS IoT Core Entwicklerhandbuch.

Verwenden Sie den * Platzhalter, um mehreren Client-Geräten eine Auswahlregelklausel zuzuordnen. Sie können diesen Platzhalter am Ende des Dingnamens verwenden, um nach Client-Geräten zu suchen, deren Namen mit einer von Ihnen angegebenen Zeichenfolge beginnen. Sie können diesen Platzhalter auch verwenden, um alle Client-Geräte abzugleichen.

> **Note**
>
> Um einen Wert auszuwählen, der einen Doppelpunkt (`:`) enthält, maskieren Sie den Doppelpunkt mit einem umgekehrten Schrägstrich (). \\ In Formaten wie JSON müssen Sie umgekehrte Schrägstriche maskieren, sodass Sie vor dem Doppelpunkt zwei umgekehrte Schrägstriche eingeben. Geben Sie beispielsweise an, dass Sie ein Ding auswählen `thingName: MyTeam\\\\:ClientDevice1` möchten, dessen Name lautet. `MyTeam:ClientDevice1`

Sie können den folgenden Selektor angeben:

- `thingName`— Der Name des Dings eines Client-Geräts. AWS IoT

Example Beispiel für eine Auswahlregel

Die folgende Auswahlregel entspricht Client-Geräten mit dem Namen `MyClientDevice1` oder`MyClientDevice2`.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example Beispiel für eine Auswahlregel (verwenden Sie Platzhalter)

Die folgende Auswahlregel entspricht Client-Geräten, deren Namen mit `MyClientDevice` beginnen.

```
thingName: MyClientDevice*
```

Example Beispiel für eine Auswahlregel (trifft auf alle Geräte zu)

Die folgende Auswahlregel gilt für alle Client-Geräte.

```
thingName: *
```

`policyName`

Die Berechtigungsrichtlinie, die für Client-Geräte in dieser Gerätegruppe gilt. Geben Sie den Namen einer Richtlinie an, die Sie im `policies` Objekt definieren.

`policies`

Die Autorisierungsrichtlinien für Client-Geräte für Client-Geräte, die eine Verbindung zum Kerngerät herstellen. Jede Autorisierungsrichtlinie spezifiziert eine Reihe von Aktionen und die Ressourcen, über die ein Client-Gerät diese Aktionen ausführen kann.

Dieses Objekt enthält die folgenden Informationen:

`policyNameKey`

Der Name dieser Autorisierungsrichtlinie. *`policyNameKey`*Ersetzen Sie ihn durch einen Namen, anhand dessen Sie diese Autorisierungsrichtlinie leichter identifizieren können. Sie verwenden diesen Richtliniennamen, um zu definieren, welche Richtlinie für eine Gerätegruppe gilt.

Dieses Objekt enthält die folgenden Informationen:

`statementNameKey`

Der Name dieser Richtlinienerklärung. *`statementNameKey`*Ersetzen Sie ihn durch einen Namen, anhand dessen Sie diese Grundsatzerklärung leichter identifizieren können.

Dieses Objekt enthält die folgenden Informationen:

`operations`

Die Liste der Vorgänge, bei denen die Ressourcen in dieser Richtlinie berücksichtigt werden sollen.

Sie können jede der folgenden Operationen einbeziehen:

- `mqtt:connect`— Erteilt die Erlaubnis, eine Verbindung zum Kerngerät herzustellen. Client-Geräte müssen über diese Berechtigung verfügen, um eine Verbindung zu einem Kerngerät herzustellen.

  Dieser Vorgang unterstützt die folgenden Ressourcen:

  - `mqtt:clientId:`*`deviceClientId`*— Beschränken Sie den Zugriff auf der Grundlage der Client-ID, die ein Client-Gerät verwendet, um eine Verbindung zum MQTT-Broker des Kerngeräts herzustellen. Durch die *`deviceClientId`*zu verwendende Client-ID ersetzen.

- `mqtt:publish`— Erteilt die Erlaubnis, MQTT-Nachrichten zu Themen zu veröffentlichen.

  Dieser Vorgang unterstützt die folgenden Ressourcen:

  - `mqtt:topic:`*`mqttTopic`*— Beschränken Sie den Zugriff auf der Grundlage des MQTT-Themas, in dem ein Client-Gerät eine Nachricht veröffentlicht. Ersetzen Sie *`MQTTTopic durch das`* zu verwendende Thema.

    Diese Ressource unterstützt keine Platzhalter für MQTT-Themen.

- `mqtt:subscribe`— Erteilt die Erlaubnis, MQTT-Themenfilter zum Empfangen von Nachrichten zu abonnieren.

  Dieser Vorgang unterstützt die folgenden Ressourcen:

  - `mqtt:topicfilter:`*`mqttTopicFilter`*— Beschränken Sie den Zugriff auf der Grundlage der MQTT-Themen, bei denen ein Client-Gerät Nachrichten abonnieren kann. *`mqttTopicFilter`*Ersetzen Sie es durch den zu verwendenden Themenfilter.

    Diese Ressource unterstützt die Platzhalter + und # MQTT-Themen-Platzhalter. Weitere Informationen finden Sie unter MQTT-Themen im Developer Guide.AWS IoT Core

    Das Client-Gerät kann genau die Themenfilter abonnieren, die Sie zulassen. Wenn Sie dem Client-Gerät beispielsweise erlauben, die `mqtt:topicfilter:client/+/status` Ressource zu abonnieren, kann das Client-Gerät `client/+/status` zwar abonnieren, aber nicht`client/client1/status`.

Sie können den * Platzhalter angeben, um den Zugriff auf alle Aktionen zu ermöglichen.

`resources`

Die Liste der Ressourcen, die für die Operationen in dieser Richtlinie zugelassen werden sollen. Geben Sie Ressourcen an, die den Vorgängen in dieser Richtlinie entsprechen. Sie könnten beispielsweise eine Liste von MQTT-Themenressourcen (`mqtt:topic:`*`mqttTopic`*) in einer Richtlinie angeben, die den `mqtt:publish` Vorgang spezifiziert.

Sie können den * Platzhalter angeben, um den Zugriff auf alle Ressourcen zu ermöglichen. Sie können den * Platzhalter nicht verwenden, um unvollständige Ressourcen-IDs abzugleichen. Sie können beispielsweise angeben**`"resources": "*"`**, aber Sie können nicht angeben. **`"resources": "mqtt:clientId:*"`**

`statementDescription`

(Optional) Eine Beschreibung für diese Richtlinienerklärung.

`certificates`

(Optional) Die Zertifikatkonfigurationsoptionen für dieses Kerngerät. Dieses Objekt enthält die folgenden Informationen:

`serverCertificateValiditySeconds`

(Optional) Die Zeitspanne (in Sekunden), nach der das lokale MQTT-Serverzertifikat abläuft. Sie können diese Option konfigurieren, um festzulegen, wie oft Client-Geräte die Verbindung zum Kerngerät trennen und wieder herstellen.

Diese Komponente rotiert das lokale MQTT-Serverzertifikat 24 Stunden vor seinem Ablauf. Der MQTT-Broker, wie die Moquette MQTT-Broker-Komponente, generiert ein neues Zertifikat und startet neu. In diesem Fall werden alle mit diesem Kerngerät verbundenen Client-Geräte getrennt. Client-Geräte können nach kurzer Zeit wieder eine Verbindung zum Kerngerät herstellen.

Standard: `604800` (7 Tage)

Mindestwert: `172800` (2 Tage)

Höchstwert: `864000` (10 Tage)

`performance`

(Optional) Die Leistungskonfigurationsoptionen für dieses Kerngerät. Dieses Objekt enthält die folgenden Informationen:

`maxActiveAuthTokens`

(Optional) Die maximale Anzahl aktiver Autorisierungstoken für Client-Geräte. Sie können diese Anzahl erhöhen, damit eine größere Anzahl von Client-Geräten eine Verbindung zu einem Single-Core-Gerät herstellen kann, ohne sie erneut authentifizieren zu müssen.

Standard: `2500`

`cloudRequestQueueSize`

(Optional) Die maximale Anzahl von AWS Cloud Anfragen, die in die Warteschlange gestellt werden müssen, bevor diese Komponente Anfragen ablehnt.

Standard: `100`

`maxConcurrentCloudRequests`

(Optional) Die maximale Anzahl gleichzeitiger Anfragen, die an die gesendet werden sollen. AWS Cloud Sie können diese Zahl erhöhen, um die Authentifizierungsleistung auf Kerngeräten zu verbessern, auf denen Sie eine große Anzahl von Client-Geräten verbinden.

Standard: 1

`certificateAuthority`

(Optional) Konfigurationsoptionen für Zertifizierungsstellen, um die zwischengeschaltete Zertifizierungsstelle des Kerngeräts durch Ihre eigene Zwischenzertifizierungsstelle zu ersetzen.

> **Note**
>
> Wenn Sie Ihr Greengrass-Core-Gerät mit einer benutzerdefinierten Zertifizierungsstelle (CA) konfigurieren und dieselbe Zertifizierungsstelle verwenden, um Client-Gerätezertifikate auszustellen, umgeht Greengrass Autorisierungsrichtlinienprüfungen für MQTT-Operationen auf Client-Geräten. Die Authentifizierungskomponente für das Client-Gerät vertraut voll und ganz auf Clients, die Zertifikate verwenden, die von der Zertifizierungsstelle signiert wurden, für deren Verwendung sie konfiguriert ist.

Um dieses Verhalten bei der Verwendung einer benutzerdefinierten Zertifizierungsstelle einzuschränken, erstellen und signieren Sie Client-Geräte, die eine andere Zertifizierungsstelle oder Zwischenzertifizierungsstelle verwenden, und passen Sie dann die `certificateChainUri` Felder `certificateUri` und so an, dass sie auf die richtige Zwischenzertifizierungsstelle verweisen.

Dieses Objekt enthält die folgenden Informationen.

Uri des Zertifikats

Der Speicherort des Zertifikats. Dabei kann es sich um einen Dateisystem-URI oder einen URI handeln, der auf ein Zertifikat verweist, das in einem Hardware-Sicherheitsmodul gespeichert ist.

`certificateChainUri`

Der Speicherort der Zertifikatskette für die CA des Kerngeräts. Dies sollte die komplette Zertifikatskette bis zu Ihrer Stammzertifizierungsstelle sein. Dabei kann es sich um einen Dateisystem-URI oder einen URI handeln, der auf eine Zertifikatskette verweist, die in einem Hardware-Sicherheitsmodul gespeichert ist.

`privateKeyUri`

Der Speicherort des privaten Schlüssels des Kerngeräts. Dies kann ein Dateisystem-URI oder ein URI sein, der auf einen privaten Schlüssel eines Zertifikats verweist, der in einem Hardware-Sicherheitsmodul gespeichert ist.

`security`

(Optional) Sicherheitskonfigurationsoptionen für dieses Kerngerät. Dieses Objekt enthält die folgenden Informationen.

`clientDeviceTrustDurationMinutes`

Die Dauer in Minuten, während der die Authentifizierungsinformationen eines Client-Geräts als vertrauenswürdig eingestuft werden können, bevor eine erneute Authentifizierung beim Kerngerät erforderlich ist. Der Standardwert lautet 1.

`metrics`

(Optional) Die Metrikoptionen für dieses Kerngerät. Fehlermetriken werden nur angezeigt, wenn bei der Authentifizierung des Client-Geräts ein Fehler auftritt. Dieses Objekt enthält die folgenden Informationen:

`disableMetrics`

Wenn das `disableMetrics` Feld auf gesetzt ist`true`, erfasst die Authentifizierung auf dem Client-Gerät keine Metriken.

Standard: `false`

`aggregatePeriodSeconds`

Der Aggregationszeitraum in Sekunden, der bestimmt, wie oft die Authentifizierung auf dem Client-Gerät Messwerte aggregiert und an den Telemetrieagenten sendet. Dies ändert nichts daran, wie oft Metriken veröffentlicht werden, da der Telemetrieagent sie immer noch einmal täglich veröffentlicht.

Standard: `3600`

startupTimeoutSeconds

(Optional) Die maximale Zeit in Sekunden für den Start der Komponente. Der Status der Komponente ändert sich auf, `BROKEN` wenn dieser Timeout überschritten wird.

Standard: `120`

Example Beispiel: Aktualisierung der Zusammenführung von Konfigurationen (unter Verwendung einer restriktiven Richtlinie)

In der folgenden Beispielkonfiguration wird festgelegt, dass Client-Geräte, deren Namen mit beginnen, eine Verbindung herstellen und Informationen `MyClientDevice` zu allen Themen veröffentlichen/abonnieren dürfen.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
```

```
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example Beispiel: Update zur Zusammenführung von Konfigurationen (unter Verwendung einer permissiven Richtlinie)

In der folgenden Beispielkonfiguration wird festgelegt, dass alle Client-Geräte eine Verbindung herstellen und Informationen zu allen Themen veröffentlichen/abonnieren können.

```
{
```

```
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

v2.4.0 - v2.4.1

`deviceGroups`

Gerätegruppen sind Gruppen von Client-Geräten, die berechtigt sind, eine Verbindung zu einem Kerngerät herzustellen und mit diesem zu kommunizieren. Verwenden Sie Auswahlregeln, um Gruppen von Client-Geräten zu identifizieren, und definieren Sie Autorisierungsrichtlinien für Client-Geräte, die die Berechtigungen für jede Gerätegruppe spezifizieren.

Dieses Objekt enthält die folgenden Informationen:

`formatVersion`

Die Formatversion für dieses Konfigurationsobjekt.

Wählen Sie aus den folgenden Optionen aus:

- `2021-03-05`

`definitions`

Die Gerätegruppen für dieses Kerngerät. Jede Definition gibt eine Auswahlregel an, mit der bewertet wird, ob ein Client-Gerät Mitglied der Gruppe ist. Jede Definition gibt auch die Berechtigungsrichtlinie an, die auf Client-Geräte angewendet werden soll, die der Auswahlregel entsprechen. Wenn ein Client-Gerät Mitglied mehrerer Gerätegruppen ist, setzen sich die Berechtigungen des Geräts aus den Berechtigungsrichtlinien der einzelnen Gruppen zusammen.

Dieses Objekt enthält die folgenden Informationen:

`groupNameKey`

Der Name dieser Gerätegruppe. *`groupNameKey`*Ersetzen Sie ihn durch einen Namen, der Ihnen hilft, diese Gerätegruppe zu identifizieren.

Dieses Objekt enthält die folgenden Informationen:

`selectionRule`

Die Abfrage, die angibt, welche Client-Geräte Mitglieder dieser Gerätegruppe sind. Wenn ein Client-Gerät eine Verbindung herstellt, wertet das Core-Gerät diese Auswahlregel aus, um festzustellen, ob das Client-Gerät Mitglied dieser Gerätegruppe ist. Wenn das Client-Gerät Mitglied ist, verwendet das Kerngerät die Richtlinie dieser Gerätegruppe, um die Aktionen des Client-Geräts zu autorisieren.

Jede Auswahlregel umfasst mindestens eine Auswahlregelklausel, bei der es sich um eine Abfrage mit einem einzelnen Ausdruck handelt, die auf Client-Geräte zutreffen kann. Auswahlregeln verwenden dieselbe Abfragesyntax wie die AWS IoT Flottenindizierung. Weitere Informationen zur Syntax von Auswahlregeln finden Sie unter Abfragesyntax für die AWS IoT Flottenindizierung im AWS IoT Core Entwicklerhandbuch.

Verwenden Sie den * Platzhalter, um mehreren Client-Geräten eine Auswahlregelklausel zuzuordnen. Sie können diesen Platzhalter am Ende des Dingnamens verwenden, um nach Client-Geräten zu suchen, deren Namen mit einer von Ihnen angegebenen Zeichenfolge beginnen. Sie können diesen Platzhalter auch verwenden, um alle Client-Geräte abzugleichen.

> **Note**
>
> Um einen Wert auszuwählen, der einen Doppelpunkt (`:`) enthält, maskieren Sie den Doppelpunkt mit einem umgekehrten Schrägstrich (). \\ In Formaten wie JSON müssen Sie umgekehrte Schrägstriche maskieren, sodass Sie vor dem Doppelpunkt zwei umgekehrte Schrägstriche eingeben. Geben Sie beispielsweise an, dass Sie ein Ding auswählen `thingName: MyTeam\\\\:ClientDevice1` möchten, dessen Name lautet. `MyTeam:ClientDevice1`

Sie können den folgenden Selektor angeben:

- `thingName`— Der Name des Dings eines Client-Geräts. AWS IoT

Example Beispiel für eine Auswahlregel

Die folgende Auswahlregel entspricht Client-Geräten mit dem Namen `MyClientDevice1` oder`MyClientDevice2`.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example Beispiel für eine Auswahlregel (verwenden Sie Platzhalter)

Die folgende Auswahlregel entspricht Client-Geräten, deren Namen mit `MyClientDevice` beginnen.

```
thingName: MyClientDevice*
```

Example Beispiel für eine Auswahlregel (trifft auf alle Geräte zu)

Die folgende Auswahlregel gilt für alle Client-Geräte.

```
thingName: *
```

`policyName`

Die Berechtigungsrichtlinie, die für Client-Geräte in dieser Gerätegruppe gilt. Geben Sie den Namen einer Richtlinie an, die Sie im `policies` Objekt definieren.

`policies`

Die Autorisierungsrichtlinien für Client-Geräte für Client-Geräte, die eine Verbindung zum Kerngerät herstellen. Jede Autorisierungsrichtlinie spezifiziert eine Reihe von Aktionen und die Ressourcen, über die ein Client-Gerät diese Aktionen ausführen kann.

Dieses Objekt enthält die folgenden Informationen:

`policyNameKey`

Der Name dieser Autorisierungsrichtlinie. *`policyNameKey`*Ersetzen Sie ihn durch einen Namen, anhand dessen Sie diese Autorisierungsrichtlinie leichter identifizieren können. Sie verwenden diesen Richtliniennamen, um zu definieren, welche Richtlinie für eine Gerätegruppe gilt.

Dieses Objekt enthält die folgenden Informationen:

`statementNameKey`

Der Name dieser Richtlinienerklärung. *`statementNameKey`*Ersetzen Sie ihn durch einen Namen, anhand dessen Sie diese Grundsatzerklärung leichter identifizieren können.

Dieses Objekt enthält die folgenden Informationen:

`operations`

Die Liste der Vorgänge, bei denen die Ressourcen in dieser Richtlinie berücksichtigt werden sollen.

Sie können jede der folgenden Operationen einbeziehen:

- `mqtt:connect`— Erteilt die Erlaubnis, eine Verbindung zum Kerngerät herzustellen. Client-Geräte müssen über diese Berechtigung verfügen, um eine Verbindung zu einem Kerngerät herzustellen.

  Dieser Vorgang unterstützt die folgenden Ressourcen:

  - `mqtt:clientId:`*`deviceClientId`*— Beschränken Sie den Zugriff auf der Grundlage der Client-ID, die ein Client-Gerät verwendet, um eine Verbindung zum MQTT-Broker des Kerngeräts herzustellen. Durch die *`deviceClientId`*zu verwendende Client-ID ersetzen.
- `mqtt:publish`— Erteilt die Erlaubnis, MQTT-Nachrichten zu Themen zu veröffentlichen.

Dieser Vorgang unterstützt die folgenden Ressourcen:

- `mqtt:topic:`*`mqttTopic`*— Beschränken Sie den Zugriff auf der Grundlage des MQTT-Themas, in dem ein Client-Gerät eine Nachricht veröffentlicht. Ersetzen Sie *`MQTTTopic durch das`* zu verwendende Thema.

  Diese Ressource unterstützt keine Platzhalter für MQTT-Themen.
- `mqtt:subscribe`— Erteilt die Erlaubnis, MQTT-Themenfilter zum Empfangen von Nachrichten zu abonnieren.

  Dieser Vorgang unterstützt die folgenden Ressourcen:

  - `mqtt:topicfilter:`*`mqttTopicFilter`*— Beschränken Sie den Zugriff auf der Grundlage der MQTT-Themen, bei denen ein Client-Gerät Nachrichten abonnieren kann. *`mqttTopicFilter`*Ersetzen Sie es durch den zu verwendenden Themenfilter.

    Diese Ressource unterstützt die Platzhalter + und # MQTT-Themen-Platzhalter. Weitere Informationen finden Sie unter MQTT-Themen im Developer Guide.AWS IoT Core

    Das Client-Gerät kann genau die Themenfilter abonnieren, die Sie zulassen. Wenn Sie dem Client-Gerät beispielsweise erlauben, die `mqtt:topicfilter:client/+/status` Ressource zu abonnieren, kann das Client-Gerät `client/+/status` zwar abonnieren, aber nicht`client/client1/status`.

Sie können den * Platzhalter angeben, um den Zugriff auf alle Aktionen zu ermöglichen.

`resources`

Die Liste der Ressourcen, die für die Operationen in dieser Richtlinie zugelassen werden sollen. Geben Sie Ressourcen an, die den Vorgängen in dieser Richtlinie entsprechen. Sie könnten beispielsweise eine Liste von MQTT-Themenressourcen (`mqtt:topic:`*`mqttTopic`*) in einer Richtlinie angeben, die den `mqtt:publish` Vorgang spezifiziert.

Sie können den * Platzhalter angeben, um den Zugriff auf alle Ressourcen zu ermöglichen. Sie können den * Platzhalter nicht verwenden, um

unvollständige Ressourcen-IDs abzugleichen. Sie können beispielsweise angeben**`"resources": "*"`**, aber Sie können nicht angeben. **`"resources": "mqtt:clientId:*"`**

`statementDescription`

(Optional) Eine Beschreibung für diese Richtlinienerklärung.

`certificates`

(Optional) Die Zertifikatkonfigurationsoptionen für dieses Kerngerät. Dieses Objekt enthält die folgenden Informationen:

`serverCertificateValiditySeconds`

(Optional) Die Zeitspanne (in Sekunden), nach der das lokale MQTT-Serverzertifikat abläuft. Sie können diese Option konfigurieren, um festzulegen, wie oft Client-Geräte die Verbindung zum Kerngerät trennen und wieder herstellen.

Diese Komponente rotiert das lokale MQTT-Serverzertifikat 24 Stunden vor seinem Ablauf. Der MQTT-Broker, wie die Moquette MQTT-Broker-Komponente, generiert ein neues Zertifikat und startet neu. In diesem Fall werden alle mit diesem Kerngerät verbundenen Client-Geräte getrennt. Client-Geräte können nach kurzer Zeit wieder eine Verbindung zum Kerngerät herstellen.

Standard: `604800` (7 Tage)

Mindestwert: `172800` (2 Tage)

Höchstwert: `864000` (10 Tage)

`performance`

(Optional) Die Leistungskonfigurationsoptionen für dieses Kerngerät. Dieses Objekt enthält die folgenden Informationen:

`maxActiveAuthTokens`

(Optional) Die maximale Anzahl aktiver Autorisierungstoken für Client-Geräte. Sie können diese Anzahl erhöhen, damit eine größere Anzahl von Client-Geräten eine Verbindung zu einem Single-Core-Gerät herstellen kann, ohne sie erneut authentifizieren zu müssen.

Standard: `2500`

`cloudRequestQueueSize`

(Optional) Die maximale Anzahl von AWS Cloud Anfragen, die in die Warteschlange gestellt werden müssen, bevor diese Komponente Anfragen ablehnt.

Standard: `100`

`maxConcurrentCloudRequests`

(Optional) Die maximale Anzahl gleichzeitiger Anfragen, die an die gesendet werden sollen. AWS Cloud Sie können diese Zahl erhöhen, um die Authentifizierungsleistung auf Kerngeräten zu verbessern, auf denen Sie eine große Anzahl von Client-Geräten verbinden.

Standard: 1

`certificateAuthority`

(Optional) Konfigurationsoptionen für Zertifizierungsstellen, um die zwischengeschaltete Zertifizierungsstelle des Kerngeräts durch Ihre eigene Zwischenzertifizierungsstelle zu ersetzen. Dieses Objekt enthält die folgenden Informationen.

Dieses Objekt enthält die folgenden Informationen:

Uri des Zertifikats

Der Speicherort des Zertifikats. Dabei kann es sich um einen Dateisystem-URI oder einen URI handeln, der auf ein Zertifikat verweist, das in einem Hardware-Sicherheitsmodul gespeichert ist.

`certificateChainUri`

Der Speicherort der Zertifikatskette für die CA des Kerngeräts. Dies sollte die komplette Zertifikatskette bis zu Ihrer Stammzertifizierungsstelle sein. Dabei kann es sich um einen Dateisystem-URI oder einen URI handeln, der auf eine Zertifikatskette verweist, die in einem Hardware-Sicherheitsmodul gespeichert ist.

`privateKeyUri`

Der Speicherort des privaten Schlüssels des Kerngeräts. Dies kann ein Dateisystem-URI oder ein URI sein, der auf einen privaten Schlüssel eines Zertifikats verweist, der in einem Hardware-Sicherheitsmodul gespeichert ist.

`security`

(Optional) Sicherheitskonfigurationsoptionen für dieses Kerngerät. Dieses Objekt enthält die folgenden Informationen.

`clientDeviceTrustDurationMinutes`

Die Dauer in Minuten, während der die Authentifizierungsinformationen eines Client-Geräts als vertrauenswürdig eingestuft werden können, bevor eine erneute Authentifizierung beim Kerngerät erforderlich ist. Der Standardwert lautet 1.

`metrics`

(Optional) Die Metrikoptionen für dieses Kerngerät. Fehlermetriken werden nur angezeigt, wenn bei der Authentifizierung des Client-Geräts ein Fehler auftritt. Dieses Objekt enthält die folgenden Informationen:

`disableMetrics`

Wenn das `disableMetrics` Feld auf gesetzt ist`true`, erfasst die Authentifizierung auf dem Client-Gerät keine Metriken.

Standard: `false`

`aggregatePeriodSeconds`

Der Aggregationszeitraum in Sekunden, der bestimmt, wie oft die Authentifizierung auf dem Client-Gerät Messwerte aggregiert und an den Telemetrieagenten sendet. Dies ändert nichts daran, wie oft Metriken veröffentlicht werden, da der Telemetrieagent sie immer noch einmal täglich veröffentlicht.

Standard: `3600`

Example Beispiel: Update zur Zusammenführung von Konfigurationen (unter Verwendung einer restriktiven Richtlinie)

In der folgenden Beispielkonfiguration wird festgelegt, dass Client-Geräte, deren Namen mit beginnen, eine Verbindung herstellen und Informationen `MyClientDevice` zu allen Themen veröffentlichen/abonnieren dürfen.

```
{
  "deviceGroups": {
```

```
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example Beispiel: Update zur Zusammenführung von Konfigurationen (unter Verwendung einer permissiven Richtlinie)

In der folgenden Beispielkonfiguration wird festgelegt, dass alle Client-Geräte eine Verbindung herstellen und Informationen zu allen Themen veröffentlichen/abonnieren können.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

v2.3.x

`deviceGroups`

Gerätegruppen sind Gruppen von Client-Geräten, die berechtigt sind, eine Verbindung zu einem Kerngerät herzustellen und mit diesem zu kommunizieren. Verwenden Sie Auswahlregeln, um Gruppen von Client-Geräten zu identifizieren, und definieren Sie Autorisierungsrichtlinien für Client-Geräte, die die Berechtigungen für jede Gerätegruppe spezifizieren.

Dieses Objekt enthält die folgenden Informationen:

`formatVersion`

Die Formatversion für dieses Konfigurationsobjekt.

Wählen Sie aus den folgenden Optionen aus:

- `2021-03-05`

`definitions`

Die Gerätegruppen für dieses Kerngerät. Jede Definition gibt eine Auswahlregel an, mit der bewertet wird, ob ein Client-Gerät Mitglied der Gruppe ist. Jede Definition gibt auch die Berechtigungsrichtlinie an, die auf Client-Geräte angewendet werden soll, die der Auswahlregel entsprechen. Wenn ein Client-Gerät Mitglied mehrerer Gerätegruppen ist, setzen sich die Berechtigungen des Geräts aus den Berechtigungsrichtlinien der einzelnen Gruppen zusammen.

Dieses Objekt enthält die folgenden Informationen:

`groupNameKey`

Der Name dieser Gerätegruppe. *`groupNameKey`*Ersetzen Sie ihn durch einen Namen, der Ihnen hilft, diese Gerätegruppe zu identifizieren.

Dieses Objekt enthält die folgenden Informationen:

`selectionRule`

Die Abfrage, die angibt, welche Client-Geräte Mitglieder dieser Gerätegruppe sind. Wenn ein Client-Gerät eine Verbindung herstellt, wertet das Core-Gerät diese Auswahlregel aus, um festzustellen, ob das Client-Gerät Mitglied dieser Gerätegruppe ist. Wenn das Client-Gerät Mitglied ist, verwendet das Kerngerät die Richtlinie dieser Gerätegruppe, um die Aktionen des Client-Geräts zu autorisieren.

Jede Auswahlregel umfasst mindestens eine Auswahlregelklausel, bei der es sich um eine Abfrage mit einem einzelnen Ausdruck handelt, die auf Client-Geräte zutreffen kann. Auswahlregeln verwenden dieselbe Abfragesyntax wie die AWS IoT Flottenindizierung. Weitere Informationen zur Syntax von Auswahlregeln finden Sie unter Abfragesyntax für die AWS IoT Flottenindizierung im AWS IoT Core Entwicklerhandbuch.

Verwenden Sie den * Platzhalter, um mehreren Client-Geräten eine Auswahlregelklausel zuzuordnen. Sie können diesen Platzhalter am Ende des

Dingnamens verwenden, um nach Client-Geräten zu suchen, deren Namen mit einer von Ihnen angegebenen Zeichenfolge beginnen. Sie können diesen Platzhalter auch verwenden, um alle Client-Geräte abzugleichen.

> **Note**
>
> Um einen Wert auszuwählen, der einen Doppelpunkt (`:`) enthält, maskieren Sie den Doppelpunkt mit einem umgekehrten Schrägstrich (). \\ In Formaten wie JSON müssen Sie umgekehrte Schrägstriche maskieren, sodass Sie vor dem Doppelpunkt zwei umgekehrte Schrägstriche eingeben. Geben Sie beispielsweise an, dass Sie ein Ding auswählen `thingName: MyTeam\\\\:ClientDevice1` möchten, dessen Name lautet. `MyTeam:ClientDevice1`

Sie können den folgenden Selektor angeben:

- `thingName`— Der Name des Dings eines Client-Geräts. AWS IoT

Example Beispiel für eine Auswahlregel

Die folgende Auswahlregel entspricht Client-Geräten mit dem Namen `MyClientDevice1` oder`MyClientDevice2`.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example Beispiel für eine Auswahlregel (verwenden Sie Platzhalter)

Die folgende Auswahlregel entspricht Client-Geräten, deren Namen mit `MyClientDevice` beginnen.

```
thingName: MyClientDevice*
```

Example Beispiel für eine Auswahlregel (trifft auf alle Geräte zu)

Die folgende Auswahlregel gilt für alle Client-Geräte.

```
thingName: *
```

`policyName`

Die Berechtigungsrichtlinie, die für Client-Geräte in dieser Gerätegruppe gilt. Geben Sie den Namen einer Richtlinie an, die Sie im `policies` Objekt definieren.

`policies`

Die Autorisierungsrichtlinien für Client-Geräte für Client-Geräte, die eine Verbindung zum Kerngerät herstellen. Jede Autorisierungsrichtlinie spezifiziert eine Reihe von Aktionen und die Ressourcen, über die ein Client-Gerät diese Aktionen ausführen kann.

Dieses Objekt enthält die folgenden Informationen:

`policyNameKey`

Der Name dieser Autorisierungsrichtlinie. *`policyNameKey`*Ersetzen Sie ihn durch einen Namen, anhand dessen Sie diese Autorisierungsrichtlinie leichter identifizieren können. Sie verwenden diesen Richtliniennamen, um zu definieren, welche Richtlinie für eine Gerätegruppe gilt.

Dieses Objekt enthält die folgenden Informationen:

`statementNameKey`

Der Name dieser Richtlinienerklärung. *`statementNameKey`*Ersetzen Sie ihn durch einen Namen, anhand dessen Sie diese Grundsatzerklärung leichter identifizieren können.

Dieses Objekt enthält die folgenden Informationen:

`operations`

Die Liste der Vorgänge, bei denen die Ressourcen in dieser Richtlinie berücksichtigt werden sollen.

Sie können jede der folgenden Operationen einbeziehen:

- `mqtt:connect`— Erteilt die Erlaubnis, eine Verbindung zum Kerngerät herzustellen. Client-Geräte müssen über diese Berechtigung verfügen, um eine Verbindung zu einem Kerngerät herzustellen.

  Dieser Vorgang unterstützt die folgenden Ressourcen:

  - `mqtt:clientId:`*`deviceClientId`*— Beschränken Sie den Zugriff auf der Grundlage der Client-ID, die ein Client-Gerät verwendet, um eine

Verbindung zum MQTT-Broker des Kerngeräts herzustellen. Durch die *`deviceClientId`*zu verwendende Client-ID ersetzen.

- `mqtt:publish`— Erteilt die Erlaubnis, MQTT-Nachrichten zu Themen zu veröffentlichen.

  Dieser Vorgang unterstützt die folgenden Ressourcen:

  - `mqtt:topic:`*`mqttTopic`*— Beschränken Sie den Zugriff auf der Grundlage des MQTT-Themas, in dem ein Client-Gerät eine Nachricht veröffentlicht. Ersetzen Sie *`MQTTTopic durch das`* zu verwendende Thema.

    Diese Ressource unterstützt keine Platzhalter für MQTT-Themen.

- `mqtt:subscribe`— Erteilt die Erlaubnis, MQTT-Themenfilter zum Empfangen von Nachrichten zu abonnieren.

  Dieser Vorgang unterstützt die folgenden Ressourcen:

  - `mqtt:topicfilter:`*`mqttTopicFilter`*— Beschränken Sie den Zugriff auf der Grundlage der MQTT-Themen, bei denen ein Client-Gerät Nachrichten abonnieren kann. *`mqttTopicFilter`*Ersetzen Sie es durch den zu verwendenden Themenfilter.

    Diese Ressource unterstützt die Platzhalter + und # MQTT-Themen-Platzhalter. Weitere Informationen finden Sie unter MQTT-Themen im Developer Guide.AWS IoT Core

    Das Client-Gerät kann genau die Themenfilter abonnieren, die Sie zulassen. Wenn Sie dem Client-Gerät beispielsweise erlauben, die `mqtt:topicfilter:client/+/status` Ressource zu abonnieren, kann das Client-Gerät `client/+/status` zwar abonnieren, aber nicht`client/client1/status`.

Sie können den * Platzhalter angeben, um den Zugriff auf alle Aktionen zu ermöglichen.

`resources`

Die Liste der Ressourcen, die für die Operationen in dieser Richtlinie zugelassen werden sollen. Geben Sie Ressourcen an, die den Vorgängen in dieser Richtlinie entsprechen. Sie könnten beispielsweise eine Liste von MQTT-

Themenressourcen (`mqtt:topic:`*`mqttTopic`*) in einer Richtlinie angeben, die den `mqtt:publish` Vorgang spezifiziert.

Sie können den * Platzhalter angeben, um den Zugriff auf alle Ressourcen zu ermöglichen. Sie können den * Platzhalter nicht verwenden, um unvollständige Ressourcen-IDs abzugleichen. Sie können beispielsweise angeben**`"resources": "*"`**, aber Sie können nicht angeben. **`"resources": "mqtt:clientId:*"`**

`statementDescription`

(Optional) Eine Beschreibung für diese Richtlinienerklärung.

`certificates`

(Optional) Die Zertifikatkonfigurationsoptionen für dieses Kerngerät. Dieses Objekt enthält die folgenden Informationen:

`serverCertificateValiditySeconds`

(Optional) Die Zeitspanne (in Sekunden), nach der das lokale MQTT-Serverzertifikat abläuft. Sie können diese Option konfigurieren, um festzulegen, wie oft Client-Geräte die Verbindung zum Kerngerät trennen und wieder herstellen.

Diese Komponente rotiert das lokale MQTT-Serverzertifikat 24 Stunden vor seinem Ablauf. Der MQTT-Broker, wie die Moquette MQTT-Broker-Komponente, generiert ein neues Zertifikat und startet neu. In diesem Fall werden alle Client-Geräte, die mit diesem Core-Gerät verbunden sind, getrennt. Client-Geräte können nach kurzer Zeit wieder eine Verbindung zum Kerngerät herstellen.

Standard: `604800` (7 Tage)

Mindestwert: `172800` (2 Tage)

Höchstwert: `864000` (10 Tage)

`performance`

(Optional) Die Leistungskonfigurationsoptionen für dieses Kerngerät. Dieses Objekt enthält die folgenden Informationen:

`maxActiveAuthTokens`

(Optional) Die maximale Anzahl aktiver Autorisierungstoken für Client-Geräte. Sie können diese Anzahl erhöhen, damit eine größere Anzahl von Client-Geräten eine Verbindung zu einem Single-Core-Gerät herstellen kann, ohne sie erneut authentifizieren zu müssen.

Standard: `2500`

`cloudRequestQueueSize`

(Optional) Die maximale Anzahl von AWS Cloud Anfragen, die in die Warteschlange gestellt werden müssen, bevor diese Komponente Anfragen ablehnt.

Standard: `100`

`maxConcurrentCloudRequests`

(Optional) Die maximale Anzahl gleichzeitiger Anfragen, die an die gesendet werden sollen. AWS Cloud Sie können diese Zahl erhöhen, um die Authentifizierungsleistung auf Kerngeräten zu verbessern, auf denen Sie eine große Anzahl von Client-Geräten verbinden.

Standard: 1

`certificateAuthority`

(Optional) Konfigurationsoptionen für Zertifizierungsstellen, um die zwischengeschaltete Zertifizierungsstelle des Kerngeräts durch Ihre eigene Zwischenzertifizierungsstelle zu ersetzen. Dieses Objekt enthält die folgenden Informationen.

Uri des Zertifikats

Der Speicherort des Zertifikats. Dabei kann es sich um einen Dateisystem-URI oder einen URI handeln, der auf ein Zertifikat verweist, das in einem Hardware-Sicherheitsmodul gespeichert ist.

`certificateChainUri`

Der Speicherort der Zertifikatskette für die CA des Kerngeräts. Dies sollte die komplette Zertifikatskette bis zu Ihrer Stammzertifizierungsstelle sein. Dabei kann es sich um einen Dateisystem-URI oder einen URI handeln, der auf eine Zertifikatskette verweist, die in einem Hardware-Sicherheitsmodul gespeichert ist.

`privateKeyUri`

Der Speicherort des privaten Schlüssels des Kerngeräts. Dies kann ein Dateisystem-URI oder ein URI sein, der auf einen privaten Schlüssel eines Zertifikats verweist, der in einem Hardware-Sicherheitsmodul gespeichert ist.

`security`

(Optional) Sicherheitskonfigurationsoptionen für dieses Kerngerät. Dieses Objekt enthält die folgenden Informationen.

`clientDeviceTrustDurationMinutes`

Die Dauer in Minuten, während der die Authentifizierungsinformationen eines Client-Geräts als vertrauenswürdig eingestuft werden können, bevor eine erneute Authentifizierung beim Kerngerät erforderlich ist. Der Standardwert lautet 1.

Example Beispiel: Update zur Zusammenführung von Konfigurationen (unter Verwendung einer restriktiven Richtlinie)

In der folgenden Beispielkonfiguration wird festgelegt, dass Client-Geräte, deren Namen mit beginnen, eine Verbindung herstellen und Informationen `MyClientDevice` zu allen Themen veröffentlichen/abonnieren dürfen.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
```

```
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example Beispiel: Update zur Zusammenführung von Konfigurationen (unter Verwendung einer permissiven Richtlinie)

In der folgenden Beispielkonfiguration wird festgelegt, dass alle Client-Geräte eine Verbindung herstellen und Informationen zu allen Themen veröffentlichen/abonnieren können.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
```

```
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

v2.2.x

`deviceGroups`

Gerätegruppen sind Gruppen von Client-Geräten, die berechtigt sind, eine Verbindung zu einem Kerngerät herzustellen und mit diesem zu kommunizieren. Verwenden Sie Auswahlregeln, um Gruppen von Client-Geräten zu identifizieren, und definieren Sie Autorisierungsrichtlinien für Client-Geräte, die die Berechtigungen für jede Gerätegruppe spezifizieren.

Dieses Objekt enthält die folgenden Informationen:

`formatVersion`

Die Formatversion für dieses Konfigurationsobjekt.

Wählen Sie aus den folgenden Optionen aus:

- `2021-03-05`

`definitions`

Die Gerätegruppen für dieses Kerngerät. Jede Definition gibt eine Auswahlregel an, mit der bewertet wird, ob ein Client-Gerät Mitglied der Gruppe ist. Jede Definition gibt auch die Berechtigungsrichtlinie an, die auf Client-Geräte angewendet werden soll, die der Auswahlregel entsprechen. Wenn ein Client-Gerät Mitglied mehrerer Gerätegruppen ist, setzen sich die Berechtigungen des Geräts aus den Berechtigungsrichtlinien der einzelnen Gruppen zusammen.

Dieses Objekt enthält die folgenden Informationen:

groupNameKey

Der Name dieser Gerätegruppe. *groupNameKey*Ersetzen Sie ihn durch einen Namen, der Ihnen hilft, diese Gerätegruppe zu identifizieren.

Dieses Objekt enthält die folgenden Informationen:

`selectionRule`

Die Abfrage, die angibt, welche Client-Geräte Mitglieder dieser Gerätegruppe sind. Wenn ein Client-Gerät eine Verbindung herstellt, wertet das Core-Gerät diese Auswahlregel aus, um festzustellen, ob das Client-Gerät Mitglied dieser Gerätegruppe ist. Wenn das Client-Gerät Mitglied ist, verwendet das Kerngerät die Richtlinie dieser Gerätegruppe, um die Aktionen des Client-Geräts zu autorisieren.

Jede Auswahlregel umfasst mindestens eine Auswahlregelklausel, bei der es sich um eine Abfrage mit einem einzelnen Ausdruck handelt, die auf Client-Geräte zutreffen kann. Auswahlregeln verwenden dieselbe Abfragesyntax wie die AWS IoT Flottenindizierung. Weitere Informationen zur Syntax von Auswahlregeln finden Sie unter Abfragesyntax für die AWS IoT Flottenindizierung im AWS IoT Core Entwicklerhandbuch.

Verwenden Sie den * Platzhalter, um mehreren Client-Geräten eine Auswahlregelklausel zuzuordnen. Sie können diesen Platzhalter am Ende des Dingnamens verwenden, um nach Client-Geräten zu suchen, deren Namen mit einer von Ihnen angegebenen Zeichenfolge beginnen. Sie können diesen Platzhalter auch verwenden, um alle Client-Geräte abzugleichen.

> **Note**
>
> Um einen Wert auszuwählen, der einen Doppelpunkt (`:`) enthält, maskieren Sie den Doppelpunkt mit einem umgekehrten Schrägstrich (). `\\` In Formaten wie JSON müssen Sie umgekehrte Schrägstriche maskieren, sodass Sie vor dem Doppelpunkt zwei umgekehrte Schrägstriche eingeben. Geben Sie beispielsweise an, dass Sie ein Ding auswählen `thingName: MyTeam\\\\:ClientDevice1` möchten, dessen Name lautet. `MyTeam:ClientDevice1`

Sie können den folgenden Selektor angeben:

- `thingName`— Der Name des Dings eines Client-Geräts. AWS IoT

Example Beispiel für eine Auswahlregel

Die folgende Auswahlregel entspricht Client-Geräten mit dem Namen `MyClientDevice1` oder`MyClientDevice2`.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example Beispiel für eine Auswahlregel (verwenden Sie Platzhalter)

Die folgende Auswahlregel entspricht Client-Geräten, deren Namen mit `MyClientDevice` beginnen.

```
thingName: MyClientDevice*
```

Example Beispiel für eine Auswahlregel (trifft auf alle Geräte zu)

Die folgende Auswahlregel gilt für alle Client-Geräte.

```
thingName: *
```

`policyName`

Die Berechtigungsrichtlinie, die für Client-Geräte in dieser Gerätegruppe gilt. Geben Sie den Namen einer Richtlinie an, die Sie im `policies` Objekt definieren.

`policies`

Die Autorisierungsrichtlinien für Client-Geräte für Client-Geräte, die eine Verbindung zum Kerngerät herstellen. Jede Autorisierungsrichtlinie spezifiziert eine Reihe von Aktionen und die Ressourcen, über die ein Client-Gerät diese Aktionen ausführen kann.

Dieses Objekt enthält die folgenden Informationen:

`policyNameKey`

Der Name dieser Autorisierungsrichtlinie. *`policyNameKey`*Ersetzen Sie ihn durch einen Namen, anhand dessen Sie diese Autorisierungsrichtlinie leichter identifizieren

können. Sie verwenden diesen Richtliniennamen, um zu definieren, welche Richtlinie für eine Gerätegruppe gilt.

Dieses Objekt enthält die folgenden Informationen:

statementNameKey

Der Name dieser Richtlinienerklärung. *statementNameKey*Ersetzen Sie ihn durch einen Namen, anhand dessen Sie diese Grundsatzerklärung leichter identifizieren können.

Dieses Objekt enthält die folgenden Informationen:

operations

Die Liste der Vorgänge, bei denen die Ressourcen in dieser Richtlinie berücksichtigt werden sollen.

Sie können jede der folgenden Operationen einbeziehen:

- `mqtt:connect`— Erteilt die Erlaubnis, eine Verbindung zum Kerngerät herzustellen. Client-Geräte müssen über diese Berechtigung verfügen, um eine Verbindung zu einem Kerngerät herzustellen.

  Dieser Vorgang unterstützt die folgenden Ressourcen:

  - `mqtt:clientId:`*deviceClientId*— Beschränken Sie den Zugriff auf der Grundlage der Client-ID, die ein Client-Gerät verwendet, um eine Verbindung zum MQTT-Broker des Kerngeräts herzustellen. Durch die *deviceClientId*zu verwendende Client-ID ersetzen.
- `mqtt:publish`— Erteilt die Erlaubnis, MQTT-Nachrichten zu Themen zu veröffentlichen.

  Dieser Vorgang unterstützt die folgenden Ressourcen:

  - `mqtt:topic:`*mqttTopic*— Beschränken Sie den Zugriff auf der Grundlage des MQTT-Themas, in dem ein Client-Gerät eine Nachricht veröffentlicht. Ersetzen Sie *MQTTTopic durch das* zu verwendende Thema.

    Diese Ressource unterstützt keine Platzhalter für MQTT-Themen.
- `mqtt:subscribe`— Erteilt die Erlaubnis, MQTT-Themenfilter zum Empfangen von Nachrichten zu abonnieren.

Dieser Vorgang unterstützt die folgenden Ressourcen:

- `mqtt:topicfilter:`*`mqttTopicFilter`*— Beschränken Sie den Zugriff auf der Grundlage der MQTT-Themen, bei denen ein Client-Gerät Nachrichten abonnieren kann. *`mqttTopicFilter`*Ersetzen Sie es durch den zu verwendenden Themenfilter.

  Diese Ressource unterstützt die Platzhalter + und # MQTT-Themen-Platzhalter. Weitere Informationen finden Sie unter MQTT-Themen im Developer Guide.AWS IoT Core

  Das Client-Gerät kann genau die Themenfilter abonnieren, die Sie zulassen. Wenn Sie dem Client-Gerät beispielsweise erlauben, die `mqtt:topicfilter:client/+/status` Ressource zu abonnieren, kann das Client-Gerät `client/+/status` zwar abonnieren, aber nicht`client/client1/status`.

Sie können den * Platzhalter angeben, um den Zugriff auf alle Aktionen zu ermöglichen.

`resources`

Die Liste der Ressourcen, die für die Operationen in dieser Richtlinie zugelassen werden sollen. Geben Sie Ressourcen an, die den Vorgängen in dieser Richtlinie entsprechen. Sie könnten beispielsweise eine Liste von MQTT-Themenressourcen (`mqtt:topic:`*`mqttTopic`*) in einer Richtlinie angeben, die den `mqtt:publish` Vorgang spezifiziert.

Sie können den * Platzhalter angeben, um den Zugriff auf alle Ressourcen zu ermöglichen. Sie können den * Platzhalter nicht verwenden, um unvollständige Ressourcen-IDs abzugleichen. Sie können beispielsweise angeben**`"resources": "*"`**, aber Sie können nicht angeben. **`"resources": "mqtt:clientId:*"`**

`statementDescription`

(Optional) Eine Beschreibung für diese Richtlinienerklärung.

`certificates`

(Optional) Die Zertifikatkonfigurationsoptionen für dieses Kerngerät. Dieses Objekt enthält die folgenden Informationen:

`serverCertificateValiditySeconds`

(Optional) Die Zeitspanne (in Sekunden), nach der das lokale MQTT-Serverzertifikat abläuft. Sie können diese Option konfigurieren, um festzulegen, wie oft Client-Geräte die Verbindung zum Kerngerät trennen und wieder herstellen.

Diese Komponente rotiert das lokale MQTT-Serverzertifikat 24 Stunden vor seinem Ablauf. Der MQTT-Broker, wie die Moquette MQTT-Broker-Komponente, generiert ein neues Zertifikat und startet neu. In diesem Fall werden alle Client-Geräte, die mit diesem Core-Gerät verbunden sind, getrennt. Client-Geräte können nach kurzer Zeit wieder eine Verbindung zum Kerngerät herstellen.

Standard: `604800` (7 Tage)

Mindestwert: `172800` (2 Tage)

Höchstwert: `864000` (10 Tage)

`performance`

(Optional) Die Leistungskonfigurationsoptionen für dieses Kerngerät. Dieses Objekt enthält die folgenden Informationen:

`maxActiveAuthTokens`

(Optional) Die maximale Anzahl aktiver Autorisierungstoken für Client-Geräte. Sie können diese Anzahl erhöhen, damit eine größere Anzahl von Client-Geräten eine Verbindung zu einem Single-Core-Gerät herstellen kann, ohne sie erneut authentifizieren zu müssen.

Standard: `2500`

`cloudRequestQueueSize`

(Optional) Die maximale Anzahl von AWS Cloud Anfragen, die in die Warteschlange gestellt werden müssen, bevor diese Komponente Anfragen ablehnt.

Standard: `100`

`maxConcurrentCloudRequests`

(Optional) Die maximale Anzahl gleichzeitiger Anfragen, die an die gesendet werden sollen. AWS Cloud Sie können diese Zahl erhöhen, um die Authentifizierungsleistung

auf Kerngeräten zu verbessern, auf denen Sie eine große Anzahl von Client-Geräten verbinden.

Standard: 1

Example Beispiel: Update zur Zusammenführung von Konfigurationen (unter Verwendung einer restriktiven Richtlinie)

In der folgenden Beispielkonfiguration wird festgelegt, dass Client-Geräte, deren Namen mit beginnen, eine Verbindung herstellen und Informationen `MyClientDevice` zu allen Themen veröffentlichen/abonnieren dürfen.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
```

```
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example Beispiel: Update zur Zusammenführung von Konfigurationen (unter Verwendung einer permissiven Richtlinie)

In der folgenden Beispielkonfiguration wird festgelegt, dass alle Client-Geräte eine Verbindung herstellen und Informationen zu allen Themen veröffentlichen/abonnieren können.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
```

```
}
```

v2.1.x

`deviceGroups`

Gerätegruppen sind Gruppen von Client-Geräten, die berechtigt sind, eine Verbindung zu einem Kerngerät herzustellen und mit diesem zu kommunizieren. Verwenden Sie Auswahlregeln, um Gruppen von Client-Geräten zu identifizieren, und definieren Sie Autorisierungsrichtlinien für Client-Geräte, die die Berechtigungen für jede Gerätegruppe spezifizieren.

Dieses Objekt enthält die folgenden Informationen:

`formatVersion`

Die Formatversion für dieses Konfigurationsobjekt.

Wählen Sie aus den folgenden Optionen aus:

- `2021-03-05`

`definitions`

Die Gerätegruppen für dieses Kerngerät. Jede Definition gibt eine Auswahlregel an, mit der bewertet wird, ob ein Client-Gerät Mitglied der Gruppe ist. Jede Definition gibt auch die Berechtigungsrichtlinie an, die auf Client-Geräte angewendet werden soll, die der Auswahlregel entsprechen. Wenn ein Client-Gerät Mitglied mehrerer Gerätegruppen ist, setzen sich die Berechtigungen des Geräts aus den Berechtigungsrichtlinien der einzelnen Gruppen zusammen.

Dieses Objekt enthält die folgenden Informationen:

`groupNameKey`

Der Name dieser Gerätegruppe. *`groupNameKey`*Ersetzen Sie ihn durch einen Namen, der Ihnen hilft, diese Gerätegruppe zu identifizieren.

Dieses Objekt enthält die folgenden Informationen:

`selectionRule`

Die Abfrage, die angibt, welche Client-Geräte Mitglieder dieser Gerätegruppe sind. Wenn ein Client-Gerät eine Verbindung herstellt, wertet das Core-Gerät

diese Auswahlregel aus, um festzustellen, ob das Client-Gerät Mitglied dieser Gerätegruppe ist. Wenn das Client-Gerät Mitglied ist, verwendet das Kerngerät die Richtlinie dieser Gerätegruppe, um die Aktionen des Client-Geräts zu autorisieren.

Jede Auswahlregel umfasst mindestens eine Auswahlregelklausel, bei der es sich um eine Abfrage mit einem einzelnen Ausdruck handelt, die auf Client-Geräte zutreffen kann. Auswahlregeln verwenden dieselbe Abfragesyntax wie die AWS IoT Flottenindizierung. Weitere Informationen zur Syntax von Auswahlregeln finden Sie unter Abfragesyntax für die AWS IoT Flottenindizierung im AWS IoT Core Entwicklerhandbuch.

Verwenden Sie den * Platzhalter, um mehreren Client-Geräten eine Auswahlregelklausel zuzuordnen. Sie können diesen Platzhalter am Ende des Dingnamens verwenden, um nach Client-Geräten zu suchen, deren Namen mit einer von Ihnen angegebenen Zeichenfolge beginnen. Sie können diesen Platzhalter auch verwenden, um alle Client-Geräte abzugleichen.

> **Note**
>
> Um einen Wert auszuwählen, der einen Doppelpunkt (`:`) enthält, maskieren Sie den Doppelpunkt mit einem umgekehrten Schrägstrich (). \\ In Formaten wie JSON müssen Sie umgekehrte Schrägstriche maskieren, sodass Sie vor dem Doppelpunkt zwei umgekehrte Schrägstriche eingeben. Geben Sie beispielsweise an, dass Sie ein Ding auswählen `thingName: MyTeam\\\\:ClientDevice1` möchten, dessen Name lautet. `MyTeam:ClientDevice1`

Sie können den folgenden Selektor angeben:

- `thingName`— Der Name des Dings eines Client-Geräts. AWS IoT

Example Beispiel für eine Auswahlregel

Die folgende Auswahlregel entspricht Client-Geräten mit dem Namen `MyClientDevice1` oder`MyClientDevice2`.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example Beispiel für eine Auswahlregel (verwenden Sie Platzhalter)

Die folgende Auswahlregel entspricht Client-Geräten, deren Namen mit `MyClientDevice` beginnen.

```
thingName: MyClientDevice*
```

Example Beispiel für eine Auswahlregel (trifft auf alle Geräte zu)

Die folgende Auswahlregel gilt für alle Client-Geräte.

```
thingName: *
```

`policyName`

Die Berechtigungsrichtlinie, die für Client-Geräte in dieser Gerätegruppe gilt. Geben Sie den Namen einer Richtlinie an, die Sie im `policies` Objekt definieren.

`policies`

Die Autorisierungsrichtlinien für Client-Geräte für Client-Geräte, die eine Verbindung zum Kerngerät herstellen. Jede Autorisierungsrichtlinie spezifiziert eine Reihe von Aktionen und die Ressourcen, über die ein Client-Gerät diese Aktionen ausführen kann.

Dieses Objekt enthält die folgenden Informationen:

`policyNameKey`

Der Name dieser Autorisierungsrichtlinie. *`policyNameKey`*Ersetzen Sie ihn durch einen Namen, anhand dessen Sie diese Autorisierungsrichtlinie leichter identifizieren können. Sie verwenden diesen Richtliniennamen, um zu definieren, welche Richtlinie für eine Gerätegruppe gilt.

Dieses Objekt enthält die folgenden Informationen:

`statementNameKey`

Der Name dieser Richtlinienerklärung. *`statementNameKey`*Ersetzen Sie ihn durch einen Namen, anhand dessen Sie diese Grundsatzerklärung leichter identifizieren können.

Dieses Objekt enthält die folgenden Informationen:

`operations`

Die Liste der Vorgänge, bei denen die Ressourcen in dieser Richtlinie berücksichtigt werden sollen.

Sie können jede der folgenden Operationen einbeziehen:

- `mqtt:connect`— Erteilt die Erlaubnis, eine Verbindung zum Kerngerät herzustellen. Client-Geräte müssen über diese Berechtigung verfügen, um eine Verbindung zu einem Kerngerät herzustellen.

  Dieser Vorgang unterstützt die folgenden Ressourcen:

  - `mqtt:clientId:`*`deviceClientId`*— Beschränken Sie den Zugriff auf der Grundlage der Client-ID, die ein Client-Gerät verwendet, um eine Verbindung zum MQTT-Broker des Kerngeräts herzustellen. Durch die *`deviceClientId`*zu verwendende Client-ID ersetzen.

- `mqtt:publish`— Erteilt die Erlaubnis, MQTT-Nachrichten zu Themen zu veröffentlichen.

  Dieser Vorgang unterstützt die folgenden Ressourcen:

  - `mqtt:topic:`*`mqttTopic`*— Beschränken Sie den Zugriff auf der Grundlage des MQTT-Themas, in dem ein Client-Gerät eine Nachricht veröffentlicht. Ersetzen Sie *`MQTTTopic durch das`* zu verwendende Thema.

    Diese Ressource unterstützt keine Platzhalter für MQTT-Themen.

- `mqtt:subscribe`— Erteilt die Erlaubnis, MQTT-Themenfilter zum Empfangen von Nachrichten zu abonnieren.

  Dieser Vorgang unterstützt die folgenden Ressourcen:

  - `mqtt:topicfilter:`*`mqttTopicFilter`*— Beschränken Sie den Zugriff auf der Grundlage der MQTT-Themen, bei denen ein Client-Gerät Nachrichten abonnieren kann. *`mqttTopicFilter`*Ersetzen Sie es durch den zu verwendenden Themenfilter.

    Diese Ressource unterstützt die Platzhalter + und # MQTT-Themen-Platzhalter. Weitere Informationen finden Sie unter MQTT-Themen im Developer Guide.AWS IoT Core

Das Client-Gerät kann genau die Themenfilter abonnieren, die Sie zulassen. Wenn Sie dem Client-Gerät beispielsweise erlauben, die `mqtt:topicfilter:client/+/status` Ressource zu abonnieren, kann das Client-Gerät `client/+/status` zwar abonnieren, aber nicht`client/client1/status`.

Sie können den * Platzhalter angeben, um den Zugriff auf alle Aktionen zu ermöglichen.

`resources`

Die Liste der Ressourcen, die für die Operationen in dieser Richtlinie zugelassen werden sollen. Geben Sie Ressourcen an, die den Vorgängen in dieser Richtlinie entsprechen. Sie könnten beispielsweise eine Liste von MQTT-Themenressourcen (`mqtt:topic:`*`mqttTopic`*) in einer Richtlinie angeben, die den `mqtt:publish` Vorgang spezifiziert.

Sie können den * Platzhalter angeben, um den Zugriff auf alle Ressourcen zu ermöglichen. Sie können den * Platzhalter nicht verwenden, um unvollständige Ressourcen-IDs abzugleichen. Sie können beispielsweise angeben**`"resources": "*"`**, aber Sie können nicht angeben. **`"resources": "mqtt:clientId:*"`**

`statementDescription`

(Optional) Eine Beschreibung für diese Richtlinienerklärung.

`certificates`

(Optional) Die Zertifikatkonfigurationsoptionen für dieses Kerngerät. Dieses Objekt enthält die folgenden Informationen:

`serverCertificateValiditySeconds`

(Optional) Die Zeitspanne (in Sekunden), nach der das lokale MQTT-Serverzertifikat abläuft. Sie können diese Option konfigurieren, um festzulegen, wie oft Client-Geräte die Verbindung zum Kerngerät trennen und wieder herstellen.

Diese Komponente rotiert das lokale MQTT-Serverzertifikat 24 Stunden vor seinem Ablauf. Der MQTT-Broker, wie die Moquette MQTT-Broker-Komponente, generiert ein neues Zertifikat und startet neu. In diesem Fall werden alle Client-Geräte, die mit diesem

Core-Gerät verbunden sind, getrennt. Client-Geräte können nach kurzer Zeit wieder eine Verbindung zum Kerngerät herstellen.

Standard: `604800` (7 Tage)

Mindestwert: `172800` (2 Tage)

Höchstwert: `864000` (10 Tage)

Example Beispiel: Update zur Zusammenführung von Konfigurationen (unter Verwendung einer restriktiven Richtlinie)

In der folgenden Beispielkonfiguration wird festgelegt, dass Client-Geräte, deren Namen mit beginnen, eine Verbindung herstellen und Informationen `MyClientDevice` zu allen Themen veröffentlichen/abonnieren dürfen.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
```

```
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example Beispiel: Update zur Zusammenführung von Konfigurationen (unter Verwendung einer permissiven Richtlinie)

In der folgenden Beispielkonfiguration wird festgelegt, dass alle Client-Geräte eine Verbindung herstellen und Informationen zu allen Themen veröffentlichen/abonnieren können.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
```

```
        }
      }
    }
  }
}
```

v2.0.x

`deviceGroups`

Gerätegruppen sind Gruppen von Client-Geräten, die berechtigt sind, eine Verbindung zu einem Kerngerät herzustellen und mit diesem zu kommunizieren. Verwenden Sie Auswahlregeln, um Gruppen von Client-Geräten zu identifizieren, und definieren Sie Autorisierungsrichtlinien für Client-Geräte, die die Berechtigungen für jede Gerätegruppe spezifizieren.

Dieses Objekt enthält die folgenden Informationen:

`formatVersion`

Die Formatversion für dieses Konfigurationsobjekt.

Wählen Sie aus den folgenden Optionen aus:

- `2021-03-05`

`definitions`

Die Gerätegruppen für dieses Kerngerät. Jede Definition gibt eine Auswahlregel an, mit der bewertet wird, ob ein Client-Gerät Mitglied der Gruppe ist. Jede Definition gibt auch die Berechtigungsrichtlinie an, die auf Client-Geräte angewendet werden soll, die der Auswahlregel entsprechen. Wenn ein Client-Gerät Mitglied mehrerer Gerätegruppen ist, setzen sich die Berechtigungen des Geräts aus den Berechtigungsrichtlinien der einzelnen Gruppen zusammen.

Dieses Objekt enthält die folgenden Informationen:

`groupNameKey`

Der Name dieser Gerätegruppe. *`groupNameKey`*Ersetzen Sie ihn durch einen Namen, der Ihnen hilft, diese Gerätegruppe zu identifizieren.

Dieses Objekt enthält die folgenden Informationen:

`selectionRule`

Die Abfrage, die angibt, welche Client-Geräte Mitglieder dieser Gerätegruppe sind. Wenn ein Client-Gerät eine Verbindung herstellt, wertet das Core-Gerät diese Auswahlregel aus, um festzustellen, ob das Client-Gerät Mitglied dieser Gerätegruppe ist. Wenn das Client-Gerät Mitglied ist, verwendet das Kerngerät die Richtlinie dieser Gerätegruppe, um die Aktionen des Client-Geräts zu autorisieren.

Jede Auswahlregel umfasst mindestens eine Auswahlregelklausel, bei der es sich um eine Abfrage mit einem einzelnen Ausdruck handelt, die auf Client-Geräte zutreffen kann. Auswahlregeln verwenden dieselbe Abfragesyntax wie die AWS IoT Flottenindizierung. Weitere Informationen zur Syntax von Auswahlregeln finden Sie unter Abfragesyntax für die AWS IoT Flottenindizierung im AWS IoT Core Entwicklerhandbuch.

Verwenden Sie den * Platzhalter, um mehreren Client-Geräten eine Auswahlregelklausel zuzuordnen. Sie können diesen Platzhalter am Ende des Dingnamens verwenden, um nach Client-Geräten zu suchen, deren Namen mit einer von Ihnen angegebenen Zeichenfolge beginnen. Sie können diesen Platzhalter auch verwenden, um alle Client-Geräte abzugleichen.

> **Note**
>
> Um einen Wert auszuwählen, der einen Doppelpunkt (`:`) enthält, maskieren Sie den Doppelpunkt mit einem umgekehrten Schrägstrich (). `\\` In Formaten wie JSON müssen Sie umgekehrte Schrägstriche maskieren, sodass Sie vor dem Doppelpunkt zwei umgekehrte Schrägstriche eingeben. Geben Sie beispielsweise an, dass Sie ein Ding auswählen `thingName: MyTeam\\\\:ClientDevice1` möchten, dessen Name lautet. `MyTeam:ClientDevice1`

Sie können den folgenden Selektor angeben:

- `thingName`— Der Name des Dings eines Client-Geräts. AWS IoT

Example Beispiel für eine Auswahlregel

Die folgende Auswahlregel entspricht Client-Geräten mit dem Namen `MyClientDevice1` oder`MyClientDevice2`.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example Beispiel für eine Auswahlregel (verwenden Sie Platzhalter)

Die folgende Auswahlregel entspricht Client-Geräten, deren Namen mit `MyClientDevice` beginnen.

```
thingName: MyClientDevice*
```

Example Beispiel für eine Auswahlregel (trifft auf alle Geräte zu)

Die folgende Auswahlregel gilt für alle Client-Geräte.

```
thingName: *
```

`policyName`

Die Berechtigungsrichtlinie, die für Client-Geräte in dieser Gerätegruppe gilt. Geben Sie den Namen einer Richtlinie an, die Sie im `policies` Objekt definieren.

`policies`

Die Autorisierungsrichtlinien für Client-Geräte für Client-Geräte, die eine Verbindung zum Kerngerät herstellen. Jede Autorisierungsrichtlinie spezifiziert eine Reihe von Aktionen und die Ressourcen, über die ein Client-Gerät diese Aktionen ausführen kann.

Dieses Objekt enthält die folgenden Informationen:

`policyNameKey`

Der Name dieser Autorisierungsrichtlinie. *`policyNameKey`*Ersetzen Sie ihn durch einen Namen, anhand dessen Sie diese Autorisierungsrichtlinie leichter identifizieren können. Sie verwenden diesen Richtliniennamen, um zu definieren, welche Richtlinie für eine Gerätegruppe gilt.

Dieses Objekt enthält die folgenden Informationen:

`statementNameKey`

Der Name dieser Richtlinienerklärung. *`statementNameKey`*Ersetzen Sie ihn durch einen Namen, anhand dessen Sie diese Grundsatzerklärung leichter identifizieren können.

Dieses Objekt enthält die folgenden Informationen:

`operations`

Die Liste der Vorgänge, bei denen die Ressourcen in dieser Richtlinie berücksichtigt werden sollen.

Sie können jede der folgenden Operationen einbeziehen:

- `mqtt:connect`— Erteilt die Erlaubnis, eine Verbindung zum Kerngerät herzustellen. Client-Geräte müssen über diese Berechtigung verfügen, um eine Verbindung zu einem Kerngerät herzustellen.

  Dieser Vorgang unterstützt die folgenden Ressourcen:

  - `mqtt:clientId:`*`deviceClientId`*— Beschränken Sie den Zugriff auf der Grundlage der Client-ID, die ein Client-Gerät verwendet, um eine Verbindung zum MQTT-Broker des Kerngeräts herzustellen. Durch die *`deviceClientId`*zu verwendende Client-ID ersetzen.

- `mqtt:publish`— Erteilt die Erlaubnis, MQTT-Nachrichten zu Themen zu veröffentlichen.

  Dieser Vorgang unterstützt die folgenden Ressourcen:

  - `mqtt:topic:`*`mqttTopic`*— Beschränken Sie den Zugriff auf der Grundlage des MQTT-Themas, in dem ein Client-Gerät eine Nachricht veröffentlicht. Ersetzen Sie *`MQTTTopic durch das`* zu verwendende Thema.

    Diese Ressource unterstützt keine Platzhalter für MQTT-Themen.

- `mqtt:subscribe`— Erteilt die Erlaubnis, MQTT-Themenfilter zum Empfangen von Nachrichten zu abonnieren.

  Dieser Vorgang unterstützt die folgenden Ressourcen:

  - `mqtt:topicfilter:`*`mqttTopicFilter`*— Beschränken Sie den Zugriff auf der Grundlage der MQTT-Themen, bei denen ein Client-Gerät Nachrichten abonnieren kann. *`mqttTopicFilter`*Ersetzen Sie es durch den zu verwendenden Themenfilter.

    Diese Ressource unterstützt die Platzhalter + und # MQTT-Themen-Platzhalter. Weitere Informationen finden Sie unter MQTT-Themen im Developer Guide.AWS IoT Core

Das Client-Gerät kann genau die Themenfilter abonnieren, die Sie zulassen. Wenn Sie dem Client-Gerät beispielsweise erlauben, die `mqtt:topicfilter:client/+/status` Ressource zu abonnieren, kann das Client-Gerät `client/+/status` zwar abonnieren, aber nicht`client/client1/status`.

Sie können den * Platzhalter angeben, um den Zugriff auf alle Aktionen zu ermöglichen.

`resources`

Die Liste der Ressourcen, die für die Operationen in dieser Richtlinie zugelassen werden sollen. Geben Sie Ressourcen an, die den Vorgängen in dieser Richtlinie entsprechen. Sie könnten beispielsweise eine Liste von MQTT-Themenressourcen (`mqtt:topic:`*`mqttTopic`*) in einer Richtlinie angeben, die den `mqtt:publish` Vorgang spezifiziert.

Sie können den * Platzhalter angeben, um den Zugriff auf alle Ressourcen zu ermöglichen. Sie können den * Platzhalter nicht verwenden, um unvollständige Ressourcen-IDs abzugleichen. Sie können beispielsweise angeben**`"resources": "*"`**, aber Sie können nicht angeben. **`"resources": "mqtt:clientId:*"`**

`statementDescription`

(Optional) Eine Beschreibung für diese Richtlinienerklärung.

Example Beispiel: Update zur Zusammenführung von Konfigurationen (unter Verwendung einer restriktiven Richtlinie)

In der folgenden Beispielkonfiguration wird festgelegt, dass Client-Geräte, deren Namen mit beginnen, eine Verbindung herstellen und Informationen `MyClientDevice` zu allen Themen veröffentlichen/abonnieren dürfen.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
```

```
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example Beispiel: Update zur Zusammenführung von Konfigurationen (unter Verwendung einer permissiven Richtlinie)

In der folgenden Beispielkonfiguration wird festgelegt, dass alle Client-Geräte eine Verbindung herstellen und Informationen zu allen Themen veröffentlichen/abonnieren können.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

Lokale Protokolldatei

Diese Komponente verwendet dieselbe Protokolldatei wie die Greengrass Nucleus-Komponente.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

Um die Protokolle dieser Komponente einzusehen

- Führen Sie den folgenden Befehl auf dem Kerngerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

  Linux

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

  Windows (PowerShell)

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

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
2.5.0	Neue Features • Ermöglicht die Ersetzung von Richtlinienressourcen durch `${iot:Connection.Thing.ThingName}` Variablen. • Ermöglicht Richtlinienressourcen mit Platzhaltern wie. `mqtt:topic:my*`
2.4.5	Neue Features Integriert die Unterstützung für Platzhalterpräfixe zur Auswahl von Dingnamen mit dem Parameter. `selectionRule` Fehlerkorrekturen und Verbesserungen Behebt ein Problem, bei dem Zertifikate in bestimmten Fällen nicht mit neuen Verbindungsinformationen aktualisiert werden.
2.4.4	Die Version wurde für die Version 2.12.0 von Greengrass Nucleus aktualisiert.

Version	Änderungen
2.4.3	Die Version wurde für die Version 2.11.0 von Greengrass Nucleus aktualisiert.
2.4.2	Neue Features Fügt eine neue `startupTimeoutSeconds` Konfigurationsoption hinzu.
2.4.1	Die Version wurde für die Version 2.10.0 von Greengrass Nucleus aktualisiert.
2.4.0	Neue Features • Integriert die Unterstützung für die Authentifizierung von Client-Geräten zur Ausgabe von Betriebsmetriken, die vom Telemetrieagent veröffent licht werden. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem die Authentifizierung des Client-Geräts mehr als 10 Sekunden benötigt, um die Identität eines Client-Geräts zu überprüfen. • Zusätzliche kleinere Korrekturen und Verbesserungen.
2.3.2	Fehlerkorrekturen und Verbesserungen • Integriert die Unterstützung für das Zwischenspeichern von Hostnamen informationen, sodass die Komponente die Zertifikatsanträge korrekt generiert, wenn sie offline neu gestartet wird.
2.3.1	Fehlerkorrekturen und Verbesserungen • Behebt ein Speicherleck.

Version	Änderungen
2.3.0	**Warning** Diese Version ist nicht mehr verfügbar. Die Verbesserungen in dieser Version sind in späteren Versionen dieser Komponente verfügbar. Neue Features • Integriert die Unterstützung für die Offline-Authentifizierung von Client-Geräten, sodass diese weiterhin eine Verbindung zum Kerngerät herstellen können, wenn das Core-Gerät nicht mit dem Internet verbunden ist. • Integriert die Unterstützung für vom Kunden bereitgestellte Zertifizierungsstellen, die das Kerngerät als Stammzertifikat zur Generierung von MQTT-Brokerzertifikaten verwendet.
2.2.3	Die Version wurde für die Version 2.8.0 von Greengrass Nucleus aktualisiert.
2.2.2	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem das lokale MQTT-Serverzertifikat in bestimmten Szenarien häufiger rotiert als vorgesehen.
2.2.1	Die Version wurde für die Version 2.7.0 von Greengrass Nucleus aktualisiert.
2.2.0	Neue Features • Integriert die Unterstützung für benutzerdefinierte Komponenten zum Aufrufen von Interprozesskommunikation (IPC) zur Authentifizierung und Autorisierung von Client-Geräten. Sie können diese Operationen beispielsweise in einer benutzerdefinierten MQTT-Broker-Komponente verwenden. Weitere Informationen finden Sie unter IPC: Client-Geräte authentifizieren und autorisieren. • Fügt die `threadPoolSize` Optionen `maxActiveAuthTokens` `cloudQueueSize` , und hinzu, die Sie konfigurieren können, um die Leistung dieser Komponente zu optimieren.

Version	Änderungen
2.1.0	Neue Features • Fügt die `serverCertificateValiditySeconds` Option hinzu, die Sie so konfigurieren können, dass Sie anpassen können, wann das MQTT-Broker-Serverzertifikat abläuft. Sie können das Serverzertifikat so konfigurieren, dass es nach 2 bis 10 Tagen abläuft. Fehlerkorrekturen und Verbesserungen • Behebt Probleme mit der Art und Weise, wie diese Komponente Updates zum Zurücksetzen der Konfiguration verarbeitet. • Behebt ein Problem, bei dem das lokale MQTT-Serverzertifikat in bestimmten Szenarien häufiger rotiert als vorgesehen. Um diesen Fix anzuwenden, müssen Sie auch Version 2.1.0 oder höher der Moquette MQTT-Broker-Komponente verwenden. • Verbessert die Meldungen, die diese Komponente protokolliert, wenn sie Zertifikate rotiert. • Die Version wurde für die Version 2.6.0 von Greengrass Nucleus aktualisiert.
2.0.4	Die Version wurde für die Version 2.5.0 von Greengrass Nucleus aktualisiert.
2.0.3	Fehlerkorrekturen und Verbesserungen • Die Anmeldeinformationen werden jetzt aktualisiert, wenn Sie den privaten Schlüssel des Kerngeräts wechseln. • Aktualisierungen, um Protokollnachrichten klarer zu machen.
2.0.2	Die Version wurde für die Version 2.4.0 von Greengrass Nucleus aktualisiert.
2.0.1	Die Version wurde für die Version 2.3.0 von Greengrass Nucleus aktualisiert.
2.0.0	Erste Version

CloudWatch -Metriken

Die Amazon- CloudWatch Metrikkomponente (`aws.greengrass.Cloudwatch`) veröffentlicht benutzerdefinierte Metriken von Greengrass-Core-Geräten in Amazon CloudWatch. Die Komponente ermöglicht es Komponenten, CloudWatch Metriken zu veröffentlichen, mit denen Sie die Umgebung des Greengrass-Kerngeräts überwachen und analysieren können. Weitere Informationen finden Sie unter Verwenden von Amazon CloudWatch-Metriken im Amazon- CloudWatch Benutzerhandbuch.

Um eine CloudWatch Metrik mit dieser Komponente zu veröffentlichen, veröffentlichen Sie eine Nachricht in einem Thema, in dem diese Komponente abonniert wird. Standardmäßig abonniert diese Komponente das `cloudwatch/metric/put` lokale Veröffentlichungs-/Abonnementthema. Sie können bei der Bereitstellung dieser Komponente andere Themen angeben, einschließlich AWS IoT Core MQTT-Themen.

Diese Komponente stapelt Metriken, die sich im selben Namespace befinden, und veröffentlicht sie CloudWatch in regelmäßigen Abständen in .

> **Note**
>
> Diese Komponente bietet ähnliche Funktionen wie der CloudWatch Metrik-Konnektor in AWS IoT Greengrass V1. Weitere Informationen finden Sie unter CloudWatch Metrics Connector im AWS IoT Greengrass V1-Entwicklerhandbuch.

Themen

- Änderungsprotokoll
- Weitere Informationen finden Sie auch unter

Versionen

Diese Komponente hat die folgenden Versionen:

- 3.1.x
- 3.0.x
- 2.1.x
- 2.0.x

Informationen zu Änderungen in jeder Version der Komponente finden Sie im Änderungsprotokoll .

Typ

v3.x

Diese Komponente ist eine generische Komponente (`aws.greengrass.generic`). Der Greengrass-Kern führt die Lebenszyklusskripte der Komponente aus.

v2.x

Diese Komponente ist eine Lambda-Komponente (`aws.greengrass.lambda`). Der Greengrass-Kern führt die Lambda-Funktion dieser Komponente mit der Lambda-Launcher-Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

v3.x

Diese Komponente kann auf -Core-Geräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

v2.x

Diese Komponente kann nur auf Linux-Core-Geräten installiert werden.

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

3.x

- Python Version 3.7 ist auf dem Core-Gerät installiert und der PATH-Umgebungsvariablen hinzugefügt.
- Die Greengrass-Geräterolle muss die `cloudwatch:PutMetricData` Aktion zulassen, wie in der folgenden Beispiel-IAM-Richtlinie gezeigt.

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

Weitere Informationen finden Sie in der Amazon CloudWatch-Berechtigungsreferenz im Amazon- CloudWatch Benutzerhandbuch.

2.x

- Ihr Core-Gerät muss die Anforderungen erfüllen, um Lambda-Funktionen auszuführen. Wenn Sie möchten, dass das Core-Gerät containerisierte Lambda-Funktionen ausführt, muss das Gerät die Voraussetzungen dafür erfüllen. Weitere Informationen finden Sie unter Anforderungen an die Lambda-Funktion.
- Python Version 3.7 ist auf dem Core-Gerät installiert und der PATH-Umgebungsvariablen hinzugefügt.

- Die Greengrass-Geräterolle muss die `cloudwatch:PutMetricData` Aktion zulassen, wie in der folgenden Beispiel-IAM-Richtlinie gezeigt.

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

  Weitere Informationen finden Sie in der Amazon CloudWatch-Berechtigungsreferenz im Amazon- CloudWatch Benutzerhandbuch.

- Um Ausgabedaten von dieser Komponente zu erhalten, müssen Sie das folgende Konfigurationsupdate für die Legacy-Abonnement-Routerkomponente (`aws.greengrass.LegacySubscriptionRouter`) zusammenführen, wenn Sie diese Komponente bereitstellen. Diese Konfiguration gibt das Thema an, in dem diese Komponente Antworten veröffentlicht.

  Legacy subscription router v2.1.x

```
{
  "subscriptions": {
    "aws-greengrass-cloudwatch": {
      "id": "aws-greengrass-cloudwatch",
      "source": "component:aws.greengrass.Cloudwatch",
      "subject": "cloudwatch/metric/put/status",
      "target": "cloud"
    }
  }
}
```

  Legacy subscription router v2.0.x

```
{
  "subscriptions": {
```

```
    "aws-greengrass-cloudwatch": {
      "id": "aws-greengrass-cloudwatch",
      "source": "arn:aws:lambda:region:aws:function:aws-greengrass-
cloudwatch:version",
      "subject": "cloudwatch/metric/put/status",
      "target": "cloud"
    }
  }
}
```

- Ersetzen Sie *region* durch die AWS-Region , die Sie verwenden.
- Ersetzen Sie *Version* durch die Version der Lambda-Funktion, die diese Komponente ausführt. Um die Version der Lambda-Funktion zu finden, müssen Sie das Rezept für die Version dieser Komponente anzeigen, die Sie bereitstellen möchten. Öffnen Sie die Detailseite dieser Komponente in der AWS IoT Greengrass Konsole und suchen Sie nach dem Schlüssel-Wert-Paar der Lambda-Funktion. Dieses Schlüssel-Wert-Paar enthält den Namen und die Version der Lambda-Funktion.

> **Important**
>
> Sie müssen die Lambda-Funktionsversion auf dem Legacy-Abonnement-Router jedes Mal aktualisieren, wenn Sie diese Komponente bereitstellen. Dadurch wird sichergestellt, dass Sie die richtige Lambda-Funktionsversion für die von Ihnen bereitgestellte Komponentenversion verwenden.

Weitere Informationen finden Sie unter Erstellen von Bereitstellungen.

Endpunkte und Ports

Diese Komponente muss in der Lage sein, ausgehende Anfragen an die folgenden Endpunkte und Ports auszuführen, zusätzlich zu den für den grundlegenden Betrieb erforderlichen Endpunkten und Ports. Weitere Informationen finden Sie unter Zulassen von Gerätedatenverkehr über einen Proxy oder eine Firewall.

Endpunkt	Port	Erforderlich	Beschreibung
`monitoring. region.amazonaws.com`	443	Ja	Laden Sie CloudWatch Metriken hoch.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und alle ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitzustellen. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente und die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste Abhängigkeiten.

3.0.0 - 3.1.0

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 3.0.0 bis 3.1.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <3.0.0	Weich
Token-Exchange-Service	>=0.0.0	Hart

2.1.2 and 2.1.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.2 und 2.1.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.8.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.1.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.7.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.0.8 - 2.1.0

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.0.8 bis 2.1.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.6.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.0.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.5.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.0.6

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.4.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.0.5

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.3.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Token-Exchange-Service	^2.0.0	Hart

2.0.4

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.2.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.0.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.3 <2.1.0	Hart
Lambda-Launcher	>=1.0.0	Hart
Lambda-Laufzeiten	>=1.0.0	Weich
Token-Exchange-Service	>=1.0.0	Hart

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zum Komponentenrezept .

Konfiguration

Diese Komponente bietet die folgenden Konfigurationsparameter, die Sie bei der Bereitstellung der Komponente anpassen können.

v3.x

`PublishInterval`

(Optional) Die maximale Anzahl von Sekunden, die gewartet werden muss, bevor die Komponente Batch-Metriken für einen bestimmten Namespace veröffentlicht. Um die Komponente so zu konfigurieren, dass Metriken veröffentlicht werden, sobald sie sie empfangen, d. h. ohne Stapelverarbeitung, geben Sie an`0`.

Die Komponente veröffentlicht in , CloudWatch nachdem sie 20 Metriken im selben Namespace oder nach dem von Ihnen angegebenen Intervall erhalten hat.

> **Note**
>
> Die Komponente gibt nicht die Reihenfolge an, in der Ereignisse veröffentlicht werden.

Dieser Wert kann maximal 900 Sekunden betragen.

Standard: 10 Sekunden

`MaxMetricsToRetain`

(Optional) Die maximale Anzahl von Metriken für alle Namespaces, die im Speicher gespeichert werden sollen, bevor die Komponente sie durch neuere Metriken ersetzt.

Dieses Limit gilt, wenn das Core-Gerät keine Verbindung zum Internet hat, sodass die Komponente die Metriken puffert, um sie später zu veröffentlichen. Wenn der Puffer voll ist, ersetzt die Komponente die ältesten Metriken durch neuere. Metriken in einem bestimmten Namespace ersetzen nur Metriken im selben Namespace.

> **Note**
>
> Wenn der Host-Prozess für die Komponente unterbrochen wird, speichert die Komponente keine Metriken. Dies kann beispielsweise während einer Bereitstellung oder beim Neustart des Core-Geräts geschehen.

Dieser Wert muss mindestens 2 000 Metriken betragen.

Standard: 5 000 Metriken

`InputTopic`

(Optional) Das Thema, das die Komponente abonniert, um Nachrichten zu empfangen. Wenn Sie `true` für angeben`PubSubToIoTCore`, können Sie in diesem Thema MQTT-Platzhalter (+ und #) verwenden.

Standard: `cloudwatch/metric/put`

`OutputTopic`

(Optional) Das Thema, zu dem die Komponente Statusantworten veröffentlicht.

Standard: `cloudwatch/metric/put/status`

`PubSubToIoTCore`

(Optional) Zeichenfolgenwert, der definiert, ob AWS IoT Core MQTT-Themen veröffentlicht und abonniert werden sollen. Unterstützte Werte sind `true` und `false`.

Standard: `false`

`UseInstaller`

(Optional) Boolescher Wert, der definiert, ob das Installationsskript in dieser Komponente verwendet werden soll, um die SDK-Abhängigkeiten dieser Komponente zu installieren.

Setzen Sie diesen Wert auf , `false` wenn Sie ein benutzerdefiniertes Skript zum Installieren von Abhängigkeiten verwenden möchten, oder wenn Sie Laufzeitabhängigkeiten in ein vorgefertigtes Linux-Image aufnehmen möchten. Um diese Komponente zu verwenden, müssen Sie die folgenden Bibliotheken, einschließlich aller Abhängigkeiten, installieren und sie dem Standard-Greengrass-Systembenutzer zur Verfügung stellen.

- AWS IoT Device SDK v2 für Python
- AWS SDK for Python (Boto3)

Standard: `true`

`PublishRegion`

(Optional) Die AWS-Region , in der CloudWatch Metriken veröffentlicht werden sollen. Dieser Wert überschreibt die Standardregion für das Core-Gerät. Dieser Parameter ist nur für regionsübergreifende Metriken erforderlich.

`accessControl`

(Optional) Das Objekt, das die Autorisierungsrichtlinie enthält, die es der Komponente ermöglicht, die angegebenen Themen zu veröffentlichen und zu abonnieren. Wenn Sie benutzerdefinierte Werte für `InputTopic` und angeben`OutputTopic`, müssen Sie die Ressourcenwerte in diesem Objekt aktualisieren.

Standard:

```
{
  "aws.greengrass.ipc.pubsub": {
    "aws.greengrass.Cloudwatch:pubsub:1": {
      "policyDescription": "Allows access to subscribe to input topics.",
      "operations": [
        "aws.greengrass#SubscribeToTopic"
      ],
      "resources": [
        "cloudwatch/metric/put"
      ]
    },
    "aws.greengrass.Cloudwatch:pubsub:2": {
      "policyDescription": "Allows access to publish to output topics.",
      "operations": [
        "aws.greengrass#PublishToTopic"
      ],
      "resources": [
        "cloudwatch/metric/put/status"
      ]
    }
  },
  "aws.greengrass.ipc.mqttproxy": {
    "aws.greengrass.Cloudwatch:mqttproxy:1": {
      "policyDescription": "Allows access to subscribe to input topics.",
      "operations": [
        "aws.greengrass#SubscribeToIoTCore"
      ],
      "resources": [
        "cloudwatch/metric/put"
      ]
    },
    "aws.greengrass.Cloudwatch:mqttproxy:2": {
      "policyDescription": "Allows access to publish to output topics.",
      "operations": [
```

```
        "aws.greengrass#PublishToIoTCore"
      ],
      "resources": [
        "cloudwatch/metric/put/status"
      ]
    }
  }
}
```

Example Beispiel: Aktualisierung der Konfigurationszusammenführung

```
{
  "PublishInterval": 0,
  "PubSubToIoTCore": true
}
```

v2.x

> **Note**
>
> Die Standardkonfiguration dieser Komponente enthält Lambda-Funktionsparameter. Wir empfehlen Ihnen, nur die folgenden Parameter zu bearbeiten, um diese Komponente auf Ihren Geräten zu konfigurieren.

`lambdaParams`

Ein Objekt, das die Parameter für die Lambda-Funktion dieser Komponente enthält. Dieses Objekt enthält die folgenden Informationen:

`EnvironmentVariables`

Ein Objekt, das die Parameter der Lambda-Funktion enthält. Dieses Objekt enthält die folgenden Informationen:

`PUBLISH_INTERVAL`

(Optional) Die maximale Anzahl von Sekunden, die gewartet werden muss, bevor die Komponente Batch-Metriken für einen bestimmten Namespace veröffentlicht. Um die Komponente so zu konfigurieren, dass Metriken veröffentlicht werden, sobald sie sie empfangen, d. h. ohne Stapelverarbeitung, geben Sie an`0`.

Die Komponente veröffentlicht in , CloudWatch nachdem sie 20 Metriken im selben Namespace oder nach dem von Ihnen angegebenen Intervall erhalten hat.

> **Note**
>
> Die Komponente garantiert nicht die Reihenfolge, in der Ereignisse veröffentlicht werden.

Dieser Wert kann maximal 900 Sekunden betragen.

Standard: 10 Sekunden

`MAX_METRICS_TO_RETAIN`

(Optional) Die maximale Anzahl von Metriken für alle Namespaces, die im Speicher gespeichert werden sollen, bevor die Komponente sie durch neuere Metriken ersetzt.

Dieses Limit gilt, wenn das Core-Gerät keine Verbindung zum Internet hat, sodass die Komponente die Metriken puffert, um sie später zu veröffentlichen. Wenn der Puffer voll ist, ersetzt die Komponente die ältesten Metriken durch neuere. Metriken in einem bestimmten Namespace ersetzen nur Metriken im selben Namespace.

> **Note**
>
> Wenn der Host-Prozess für die Komponente unterbrochen wird, speichert die Komponente keine Metriken. Dies kann beispielsweise während einer Bereitstellung oder beim Neustart des Core-Geräts geschehen.

Dieser Wert muss mindestens 2 000 Metriken betragen.

Standard: 5 000 Metriken

`PUBLISH_REGION`

(Optional) Die AWS-Region , in der CloudWatch Metriken veröffentlicht werden sollen. Dieser Wert überschreibt die Standardregion für das Core-Gerät. Dieser Parameter ist nur für regionsübergreifende Metriken erforderlich.

`containerMode`

(Optional) Der Containerisierungsmodus für diese Komponente. Wählen Sie aus den folgenden Optionen aus:

- `NoContainer` – Die Komponente wird nicht in einer isolierten Laufzeitumgebung ausgeführt.
- `GreengrassContainer` – Die Komponente wird in einer isolierten Laufzeitumgebung innerhalb des AWS IoT Greengrass Containers ausgeführt.

Standard: `GreengrassContainer`

`containerParams`

(Optional) Ein Objekt, das die Containerparameter für diese Komponente enthält. Die Komponente verwendet diese Parameter, wenn Sie `GreengrassContainer` für angeben`containerMode`.

Dieses Objekt enthält die folgenden Informationen:

`memorySize`

(Optional) Die Speichermenge (in Kilobyte), die der Komponente zugewiesen werden soll.

Der Standardwert ist 64 MB (65.535 KB).

`pubsubTopics`

(Optional) Ein Objekt, das die Themen enthält, in denen die Komponente den Empfang von Nachrichten abonniert. Sie können jedes Thema angeben und angeben, ob die Komponente MQTT-Themen von AWS IoT Core oder lokale Veröffentlichungs-/Abonnementthemen abonniert.

Dieses Objekt enthält die folgenden Informationen:

`0` – Dies ist ein Array-Index als Zeichenfolge.

Ein Objekt, das die folgenden Informationen enthält:

`type`

(Optional) Der Typ des Veröffentlichungs-/Abonnement-Messagings, den diese Komponente zum Abonnieren von Nachrichten verwendet. Wählen Sie aus den folgenden Optionen aus:

- PUB_SUB — Abonnieren Sie lokale Veröffentlichen/Abonnement-Nachrichten. Wenn Sie diese Option wählen, darf das Thema keine MQTT-Platzhalter enthalten. Weitere Informationen zum Senden von Nachrichten von einer benutzerdefinierten Komponente, wenn Sie diese Option angeben, finden Sie unter Lokale Nachrichten veröffentlichen/abonnieren.
- `IOT_CORE` – Abonnieren Sie AWS IoT Core MQTT-Nachrichten. Wenn Sie diese Option wählen, kann das Thema MQTT-Platzhalter enthalten. Weitere Informationen zum Senden von Nachrichten von benutzerdefinierten Komponenten, wenn Sie diese Option angeben, finden Sie unter MQTT-Nachrichten veröffentlichen/abonnieren AWS IoT Core.

Standard: `PUB_SUB`

`topic`

(Optional) Das Thema, das die Komponente abonniert, um Nachrichten zu empfangen. Wenn Sie `IotCore` für angeben`type`, können Sie in diesem Thema MQTT-Platzhalter (+ und #) verwenden.

Example Beispiel: Aktualisierung der Konfigurationszusammenführung (Container-Modus)

```
{
  "containerMode": "GreengrassContainer"
}
```

Example Beispiel: Aktualisierung der Konfigurationszusammenführung (kein Containermodus)

```
{
  "containerMode": "NoContainer"
}
```

Eingabedaten

Diese Komponente akzeptiert Metriken zum folgenden Thema und veröffentlicht die Metriken in CloudWatch. Standardmäßig abonniert diese Komponente lokale Veröffentlichungs-/Abonnementnachrichten. Weitere Informationen zum Veröffentlichen von Nachrichten in dieser Komponente aus Ihren benutzerdefinierten Komponenten finden Sie unter Lokale Nachrichten veröffentlichen/abonnieren.

Ab Komponentenversion v3.0.0 können Sie diese Komponente optional so konfigurieren, dass sie ein MQTT-Thema abonniert, indem Sie den `PubSubToIoTCore` Konfigurationsparameter auf setzen`true`. Weitere Informationen zum Veröffentlichen von Nachrichten in einem MQTT-Thema in Ihren benutzerdefinierten Komponenten finden Sie unter MQTT-Nachrichten veröffentlichen/abonnieren AWS IoT Core.

Standardthema: `cloudwatch/metric/put`

Die Nachricht akzeptiert die folgenden Eigenschaften. Eingabenachrichten müssen im JSON-Format vorliegen.

`request`

Die Metrik in dieser Nachricht.

Das Anforderungsobjekt enthält die metrischen Daten, die an CloudWatch veröffentlicht werden sollen. Die Metrikwerte müssen den Spezifikationen der `PutMetricData` Operation entsprechen.

Typ: `object`, der die folgenden Informationen 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/.

Typ: `string`

Gültiges Muster: `[^:].*`

`metricData`

Die Daten für die Metrik.

Typ: `object`, der die folgenden Informationen enthält:

`metricName`

Name der Metrik.

Typ: `string`

`value`

Der Wert für die Metrik.

> **Note**
>
> CloudWatch lehnt Werte ab, die zu klein oder zu groß sind. Der Wert muss zwischen `8.515920e-109` und `1.174271e+108` (Base 10) oder `2e-360` und `2e360` (Base 2) liegen. unterstützt CloudWatch keine speziellen Werte wie `NaN`, `+Infinity`und `-Infinity`.

Typ: `double`

`dimensions`

(Optional) Die Dimensionen für die Metrik. Dimensionen liefern zusätzliche Informationen über die Metrik und ihre Daten. Eine Metrik kann bis zu 10 Dimensionen definieren.

Diese Komponente enthält automatisch eine Dimension mit dem Namen `coreName`, wobei der Wert der Name des Core-Geräts ist.

Typ: `array` der Objekte, die jeweils die folgenden Informationen enthalten:

`name`

(Optional) Der Dimensionsname.

Typ: `string`

`value`

(Optional) Der Dimensionswert.

Typ: `string`

`timestamp`

(Optional) Die Zeit, zu der die Metrikdaten empfangen wurden, ausgedrückt in Sekunden in Unix-Epochenzeit.

Standardmäßig ist dies der Zeitpunkt, zu dem die Komponente die Nachricht empfängt.

Typ: `double`

> **Note**
>
> Wenn Sie zwischen den Versionen 2.0.3 und 2.0.7 dieser Komponente verwenden, empfehlen wir, den Zeitstempel für jede Metrik separat abzurufen, wenn Sie mehrere Metriken aus einer einzigen Quelle senden. Verwenden Sie keine Variable, um den Zeitstempel zu speichern.

`unit`

(Optional) Die Einheit der Metrik.

Typ: `string`

Gültige Werte: `Seconds`, `Microseconds`, `MillisecondsBytes`, `Kilobytes`, `MegabytesGigabytes`, `Terabytes`, `Bits`, `Kilobits`, `Megabits`, `Gigabits`, `Terabits`, `Percent`, `Count`, `Bytes/Second`, `Kilobytes/Second`, `Megabytes/Second`, , `Gigabytes/Second`, , `Terabytes/Second`, `Kilobits/Second`, `Bits/Second`, `Megabits/Second`, , , `Gigabits/Second`, , , `Terabits/Second`, `Count/Second`, `None`

Standardeinstellung: `None`.

> **Note**
>
> Alle Kontingente, die für die CloudWatch `PutMetricData` API gelten, gelten für Metriken, die Sie mit dieser Komponente veröffentlichen. Die folgenden Kontingente sind besonders wichtig:
>
> - Limit von 40 KB für die API-Nutzlast
> - 20 Metriken pro API-Anforderung
> - 150 Transaktionen pro Sekunde (TPS) für die `PutMetricData`-API
>
> Weitere Informationen finden Sie unter -CloudWatch Servicekontingente im CloudWatch - Benutzerhandbuch.

Example Beispieleingabe

```
{
  "request": {
    "namespace": "Greengrass",
    "metricData": {
      "metricName": "latency",
      "dimensions": [
        {
          "name": "hostname",
          "value": "test_hostname"
        }
      ],
      "timestamp": 1539027324,
      "value": 123.0,
      "unit": "Seconds"
    }
  }
}
```

Ausgabedaten

Diese Komponente veröffentlicht Antworten standardmäßig als Ausgabedaten zum folgenden lokalen Veröffentlichungs-/Abonnementthema. Weitere Informationen zum Abonnieren von Nachrichten zu diesem Thema in Ihren benutzerdefinierten Komponenten finden Sie unter Lokale Nachrichten veröffentlichen/abonnieren.

Sie können diese Komponente optional für die Veröffentlichung in einem MQTT-Thema konfigurieren, indem Sie den `PubSubToIoTCore` Konfigurationsparameter auf setzen`true`. Weitere Informationen zum Abonnieren von Nachrichten zu einem MQTT-Thema in Ihren benutzerdefinierten Komponenten finden Sie unter MQTT-Nachrichten veröffentlichen/abonnieren AWS IoT Core.

> **Note**
>
> Komponentenversionen 2.0.x veröffentlichen Antworten standardmäßig als Ausgabedaten für ein MQTT-Thema. Sie müssen das Thema als `subject` in der Konfiguration für die Legacy-Abonnement-Routerkomponente angeben.

Standardthema: `cloudwatch/metric/put/status`

Example 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"
  }
}
```

Example Beispielausgabe: Fehler

```
{
  "response" : {
    "namespace": "Greengrass",
    "error": "InvalidInputException",
    "error_message": "cw metric is invalid",
    "status": "fail"
  }
}
```

> **Note**
>
> Wenn die Komponente einen Fehler erkennt, der erneut versucht werden kann, z. B. einen Verbindungsfehler, wiederholt sie die Veröffentlichung im nächsten Batch.

Lizenzen

Diese Komponente 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

Diese Komponente wird gemäß dem Greengrass Core Software License Agreement veröffentlicht.

Lokale Protokolldatei

Diese Komponente verwendet die folgende Protokolldatei.

Linux

```
/greengrass/v2/logs/aws.greengrass.Cloudwatch.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.Cloudwatch.log
```

So zeigen Sie die Protokolle dieser Komponente an

- Führen Sie den folgenden Befehl auf dem Core-Gerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

  Linux

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

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.Cloudwatch.log -Tail 10 -Wait
  ```

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in jeder Version der Komponente beschrieben.

v3.x

Version	Änderungen
3.1.0	Fehlerbehebungen und Verbesserungen • Fügt Unterstützung für HTTPS-Netzwerk-Proxy-Konfigurationen hinzu. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy und Ermöglichen Sie dem Kerngerät, einem HTTPS-Proxy zu vertrauen.
3.0.0	Diese Version der CloudWatch Metrikkomponente erwartet andere Konfigurationsparameter als Version 2.x. Wenn Sie eine nicht standardm äßige Konfiguration für Version 2.x verwenden und ein Upgrade von v2.x auf v3.x durchführen möchten, müssen Sie die Konfiguration der Komponente aktualisieren. Weitere Informationen finden Sie unter CloudWatch Konfiguration der Metrikkomponente . Neue Features • Fügt Unterstützung für -Core-Geräte hinzu, auf denen Windows ausgeführt wird. • Ändert den Komponententyp von einer Lambda-Komponente in eine generische Komponente. Diese Komponente hängt jetzt nicht mehr von der Legacy-Abonnement-Routerkomponente ab, um Abonnemen ts zu erstellen. • Fügt einen neuen `InputTopic` Konfigurationsparameter hinzu, um das Thema anzugeben, das die Komponente abonniert, um Nachricht en zu empfangen. • Fügt einen neuen `OutputTopic` Konfigurationsparameter hinzu, um das Thema anzugeben, zu dem die Komponente Statusantworten veröffentlicht. • Fügt einen neuen `PubSubToIoTCore` Konfigurationsparameter hinzu, um anzugeben, ob AWS IoT Core MQTT-Themen veröffentlicht und abonniert werden sollen. • Fügt den neuen `UseInstaller` Konfigurationsparameter hinzu, mit dem Sie optional das Installationsskript deaktivieren können, das Komponentenabhängigkeiten installiert.

Version	Änderungen
	Fehlerbehebungen und Verbesserungen Fügt Unterstützung für doppelte Zeitstempel in Eingabedaten hinzu.

v2.x

Version	Änderungen
2.1.3	Version für Greengrass-Kern Version 2.11.0 aktualisiert.
2.1.2	Version für Greengrass-Kern Version 2.7.0 aktualisiert.
2.1.1	Version für Greengrass-Kern Version 2.6.0 aktualisiert.
2.1.0	Neue Features • Fügt Unterstützung für HTTPS-Netzwerk-Proxy-Konfigurationen hinzu. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy und Ermöglichen Sie dem Kerngerät, einem HTTPS-Proxy zu vertrauen.
2.0.8	Fehlerbehebungen und Verbesserungen • Fügt Unterstützung für doppelte Zeitstempel in Eingabedaten hinzu. • Version für Greengrass-Kern Version 2.5.0 aktualisiert.
2.0.7	Version für Greengrass-Kern Version 2.4.0 aktualisiert.
2.0.6	Version für Greengrass-Kern Version 2.3.0 aktualisiert.
2.0.5	Version für Greengrass-Kern Version 2.2.0 aktualisiert.
2.0.4	Version für Greengrass-Kern Version 2.1.0 aktualisiert.
2.0.3	Erste Version

Weitere Informationen finden Sie auch unter

- Verwenden von Amazon CloudWatch-Metriken im Amazon CloudWatch -Benutzerhandbuch
- PutMetricData in der Amazon CloudWatch -API-Referenz zu

AWS IoT Device Defender

Die AWS IoT Device Defender Komponente (`aws.greengrass.DeviceDefender`) benachrichtigt Administratoren über Änderungen des Zustands von Greengrass-Core-Geräten. Dies kann helfen, ungewöhnliches Verhalten zu erkennen, das auf ein gefährdetes Gerät hinweisen könnte. Weitere Informationen finden Sie unter AWS IoT Device Defender im AWS IoT Core -Entwicklerhandbuch.

Diese Komponente liest Systemmetriken auf dem Core-Gerät. Anschließend werden die Metriken in veröffentlicht AWS IoT Device Defender. Weitere Informationen zum Lesen und Interpretieren der Metriken, die diese Komponente meldet, finden Sie unter Dokumentspezifikation für Gerätemetriken im AWS IoT Core -Entwicklerhandbuch.

> **Note**
>
> Diese Komponente bietet ähnliche Funktionen wie der Device-Defender-Konnektor in AWS IoT Greengrass V1. Weitere Informationen finden Sie unter Device Defender Connector im AWS IoT Greengrass V1 -Entwicklerhandbuch.

Themen

- Versionen
- Typ
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Konfiguration
- Eingabedaten
- Ausgabedaten
- Lokale Protokolldatei
- Lizenzen

- Änderungsprotokoll

Versionen

Diese Komponente hat die folgenden Versionen:

- 3.1.x
- 3.0.x
- 2.0.x

Informationen zu Änderungen in jeder Version der Komponente finden Sie im Änderungsprotokoll .

Typ

v3.x

Diese Komponente ist eine generische Komponente (`aws.greengrass.generic`). Der Greengrass-Kern führt die Lebenszyklusskripte der Komponente aus.

v2.x

Diese Komponente ist eine Lambda-Komponente (`aws.greengrass.lambda`). Der Greengrass-Kernus führt die Lambda-Funktion dieser Komponente mit der Lambda-Launcher-Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

v3.x

Diese Komponente kann auf -Core-Geräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

v2.x

Diese Komponente kann nur auf Linux-Core-Geräten installiert werden.

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

v3.x

- Python Version 3.7 ist auf dem Core-Gerät installiert und der PATH-Umgebungsvariablen hinzugefügt.
- AWS IoT Device Defender konfiguriert für die Verwendung der Detect-Funktion zur Überwachung von Verstößen. Weitere Informationen finden Sie unter Erkennen im AWS IoT Core -Entwicklerhandbuch.

v2.x

- Ihr Core-Gerät muss die Anforderungen für die Ausführung von Lambda-Funktionen erfüllen. Wenn Sie möchten, dass das Core-Gerät containerisierte Lambda-Funktionen ausführt, muss das Gerät die Anforderungen dafür erfüllen. Weitere Informationen finden Sie unter Anforderungen an die Lambda-Funktion.
- Python Version 3.7 ist auf dem Core-Gerät installiert und der PATH-Umgebungsvariablen hinzugefügt.
- AWS IoT Device Defender konfiguriert für die Verwendung der Detect-Funktion zur Überwachung von Verstößen. Weitere Informationen finden Sie unter Erkennen im AWS IoT Core -Entwicklerhandbuch.
- Die auf dem Core-Gerät installierte psutil-Bibliothek. Version 5.7.0 ist die neueste Version, die verifiziert wurde, dass sie mit der Komponente funktioniert.
- Die auf dem Core-Gerät installierte cbor-Bibliothek. Version 1.0.0 ist die neueste Version, die verifiziert wurde, dass sie mit der Komponente funktioniert.
- Um Ausgabedaten von dieser Komponente zu erhalten, müssen Sie das folgende Konfigurationsupdate für die Legacy-Abonnement-Routerkomponente (`aws.greengrass.LegacySubscriptionRouter`) zusammenführen, wenn Sie diese Komponente bereitstellen. Diese Konfiguration gibt das Thema an, in dem diese Komponente Antworten veröffentlicht.

Legacy subscription router v2.1.x

```
{
  "subscriptions": {
    "aws-greengrass-device-defender": {
      "id": "aws-greengrass-device-defender",
      "source": "component:aws.greengrass.DeviceDefender",
      "subject": "$aws/things/+/defender/metrics/json",
      "target": "cloud"
    }
  }
}
```

Legacy subscription router v2.0.x

```
{
  "subscriptions": {
    "aws-greengrass-device-defender": {
      "id": "aws-greengrass-device-defender",
      "source": "arn:aws:lambda:region:aws:function:aws-greengrass-device-
defender:version",
      "subject": "$aws/things/+/defender/metrics/json",
      "target": "cloud"
    }
  }
}
```

- Ersetzen Sie *region* durch die AWS-Region , die Sie verwenden.
- Ersetzen Sie *Version* durch die Version der Lambda-Funktion, die diese Komponente ausführt. Um die Version der Lambda-Funktion zu finden, müssen Sie das Rezept für die Version dieser Komponente anzeigen, die Sie bereitstellen möchten. Öffnen Sie die Detailseite dieser Komponente in der AWS IoT Greengrass Konsole und suchen Sie nach dem Schlüssel-Wert-Paar der Lambda-Funktion. Dieses Schlüssel-Wert-Paar enthält den Namen und die Version der Lambda-Funktion.

> **Important**
>
> Sie müssen die Lambda-Funktionsversion auf dem Legacy-Abonnement-Router jedes Mal aktualisieren, wenn Sie diese Komponente bereitstellen. Dadurch wird

sichergestellt, dass Sie die richtige Lambda-Funktionsversion für die von Ihnen bereitgestellte Komponentenversion verwenden.

Weitere Informationen finden Sie unter Erstellen von Bereitstellungen.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und alle ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitstellen zu können. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente und die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste Abhängigkeiten.

3.1.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 3.1.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <3.0.0	Weich
Token-Exchange-Service	>=0.0.0	Hart

3.0.0 - 3.0.2

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 3.0.0 bis 3.0.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <3.0.0	Weich
Token-Exchange-Service	>=0.0.0	Hart

2.0.10 and 2.0.11

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.10 und 2.0.11 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.8.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.0.9

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.9 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.7.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.0.8

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.8 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.6.0	Hart
Lambda-Launcher	^2.0.0	Hart

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.0.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.5.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.0.6

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.4.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.0.5

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.3.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.0.4

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.2.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.0.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.3 <2.1.0	Hart
Lambda-Launcher	>=1.0.0	Hart
Lambda-Laufzeiten	>=1.0.0	Weich
Token-Exchange-Service	>=1.0.0	Hart

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zum Komponentenrezept .

Konfiguration

Diese Komponente bietet die folgenden Konfigurationsparameter, die Sie anpassen können, wenn Sie die Komponente bereitstellen.

v3.x

`PublishRetryCount`

Gibt an, wie oft die Veröffentlichung erneut versucht wird. Diese Funktion ist in Version 3.1.1 verfügbar.

Das Minimum ist 0.

Das Maximum ist 72.

Standard: 5

`SampleIntervalSeconds`

(Optional) Die Zeit in Sekunden zwischen jedem Zyklus, in dem die Komponente Metriken sammelt und meldet.

Die Mindestwert beträgt 300 Sekunden (5 Minuten).

Standard: 300 Sekunden

`UseInstaller`

(Optional) Boolescher Wert, der definiert, ob das Installationsskript in dieser Komponente zum Installieren der Abhängigkeiten dieser Komponente verwendet werden soll.

Setzen Sie diesen Wert auf , `false` wenn Sie ein benutzerdefiniertes Skript zum Installieren von Abhängigkeiten verwenden möchten, oder wenn Sie Laufzeitabhängigkeiten in ein vorgefertigtes Linux-Image aufnehmen möchten. Um diese Komponente zu verwenden, müssen Sie die folgenden Bibliotheken, einschließlich aller Abhängigkeiten, installieren und sie dem Standard-Greengrass-Systembenutzer zur Verfügung stellen.

- AWS IoT Device SDK v2 für Python
- cbor-Bibliothek. Version 1.0.0 ist die neueste Version, die verifiziert wurde, dass sie mit der Komponente funktioniert.

- psutil-Bibliothek. Version 5.7.0 ist die neueste Version, die verifiziert wurde, dass sie mit der Komponente funktioniert.

> **Note**
>
> Wenn Sie Version 3.0.0 oder 3.0.1 dieser Komponente auf Core-Geräten verwenden, die Sie für die Verwendung eines HTTPS-Proxys konfigurieren, müssen Sie diesen Wert auf setzen`false`. Das Installationsskript unterstützt in diesen Versionen dieser Komponente keine Operation hinter einem HTTPS-Proxy.

Standard: `true`

v2.x

> **Note**
>
> Die Standardkonfiguration dieser Komponente enthält Lambda-Funktionsparameter. Wir empfehlen Ihnen, nur die folgenden Parameter zu bearbeiten, um diese Komponente auf Ihren Geräten zu konfigurieren.

`lambdaParams`

Ein Objekt, das die Parameter für die Lambda-Funktion dieser Komponente enthält. Dieses Objekt enthält die folgenden Informationen:

`EnvironmentVariables`

Ein Objekt, das die Parameter der Lambda-Funktion enthält. Dieses Objekt enthält die folgenden Informationen:

`PROCFS_PATH`

(Optional) Der Pfad zum `/proc` Ordner.

- Um diese Komponente in einem Container auszuführen, verwenden Sie den Standardwert `/host-proc`. Die Komponente wird standardmäßig in einem Container ausgeführt.
- Um diese Komponente im Containermodus auszuführen, geben Sie `/proc` für diesen Parameter an.

Standard: `/host-proc`. Dies ist der Standardpfad, in dem diese Komponente den `/proc` Ordner im Container mountet.

> **Note**
>
> Diese Komponente hat schreibgeschützten Zugriff auf diesen Ordner.

`SAMPLE_INTERVAL_SECONDS`

(Optional) Die Zeit in Sekunden zwischen jedem Zyklus, in dem die Komponente Metriken sammelt und meldet.

Die Mindestwert beträgt 300 Sekunden (5 Minuten).

Standard: 300 Sekunden

`containerMode`

(Optional) Der Containerisierungsmodus für diese Komponente. Wählen Sie aus den folgenden Optionen aus:

- `GreengrassContainer` – Die Komponente wird in einer isolierten Laufzeitumgebung innerhalb des AWS IoT Greengrass Containers ausgeführt.
- `NoContainer` – Die Komponente wird nicht in einer isolierten Laufzeitumgebung ausgeführt.

  Wenn Sie diese Option angeben, müssen Sie `/proc` für den `PROCFS_PATH` Umgebungsvariablenparameter angeben.

Standard: `GreengrassContainer`

`containerParams`

(Optional) Ein Objekt, das die Containerparameter für diese Komponente enthält. Die Komponente verwendet diese Parameter, wenn Sie `GreengrassContainer` für angeben`containerMode`.

Dieses Objekt enthält die folgenden Informationen:

`memorySize`

(Optional) Die Menge an Arbeitsspeicher (in Kilobyte), die der Komponente zugewiesen werden soll.

Standardmäßig 50.000 KB.

`pubsubTopics`

(Optional) Ein Objekt, das die Themen enthält, in denen die Komponente den Empfang von Nachrichten abonniert. Sie können jedes Thema angeben und angeben, ob die Komponente MQTT-Themen von AWS IoT Core oder lokale Veröffentlichungs-/Abonnementthemen abonniert.

Dieses Objekt enthält die folgenden Informationen:

`0` – Dies ist ein Array-Index als Zeichenfolge.

Ein Objekt, das die folgenden Informationen enthält:

`type`

(Optional) Der Typ des Veröffentlichungs-/Abonnement-Messagings, den diese Komponente zum Abonnieren von Nachrichten verwendet. Wählen Sie aus den folgenden Optionen aus:

- `PUB_SUB` — Abonnieren Sie lokale Veröffentlichen/Abonnement-Nachrichten. Wenn Sie diese Option wählen, darf das Thema keine MQTT-Platzhalter enthalten. Weitere Informationen zum Senden von Nachrichten von einer benutzerdefinierten Komponente, wenn Sie diese Option angeben, finden Sie unter Lokale Nachrichten veröffentlichen/abonnieren.
- `IOT_CORE` – Abonnieren Sie AWS IoT Core MQTT-Nachrichten. Wenn Sie diese Option wählen, kann das Thema MQTT-Platzhalter enthalten. Weitere Informationen zum Senden von Nachrichten von benutzerdefinierten Komponenten, wenn Sie diese Option angeben, finden Sie unter MQTT-Nachrichten veröffentlichen/abonnieren AWS IoT Core.

Standard: `PUB_SUB`

`topic`

(Optional) Das Thema, das die Komponente abonniert, um Nachrichten zu empfangen. Wenn Sie `IotCore` für angeben`type`, können Sie in diesem Thema MQTT-Platzhalter (+ und #) verwenden.

Example Beispiel: Aktualisierung der Konfigurationszusammenführung (Container-Modus)

```
{
```

```
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "PROCFS_PATH": "/host_proc"
    }
  },
  "containerMode": "GreengrassContainer"
}
```

Example Beispiel: Aktualisierung der Konfigurationszusammenführung (kein Containermodus)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "PROCFS_PATH": "/proc"
    }
  },
  "containerMode": "NoContainer"
}
```

Eingabedaten

Diese Komponente akzeptiert keine Nachrichten als Eingabedaten.

Ausgabedaten

Diese Komponente veröffentlicht Sicherheitsmetriken im folgenden reservierten Thema für AWS IoT Device Defender. Diese Komponente ersetzt beim Veröffentlichen *coreDeviceName* der Metriken durch den Namen des Core-Geräts.

Thema (AWS IoT Core MQTT): $aws/things/*coreDeviceName*/defender/metrics/json

Example Beispielausgabe

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

Weitere Informationen zu den Metriken, die diese Komponente meldet, finden Sie unter Dokumentspezifikation für Gerätemetriken im AWS IoT Core -Entwicklerhandbuch.

Lokale Protokolldatei

Diese Komponente verwendet die folgende Protokolldatei.

Linux

```
/greengrass/v2/logs/aws.greengrass.DeviceDefender.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.DeviceDefender.log
```

So zeigen Sie die Protokolle dieser Komponente an

- Führen Sie den folgenden Befehl auf dem Core-Gerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

  Linux

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

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.DeviceDefender.log -Tail 10 -
  Wait
  ```

Lizenzen

Diese Komponente wird gemäß dem Greengrass Core Software License Agreement veröffentlicht.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in jeder Version der Komponente beschrieben.

v3.x

Version	Änderungen
3.1.1	Fehlerbehebungen und Verbesserungen • Fügt Wiederholungen für die Client-Verbindung hinzu, wenn die Verbindung nach einem Netzwerkausfall nicht wiederhergestellt werden kann. • Fügt einen konfigurierbaren Wiederholungsversuch zum Veröffent lichen von Metriken hinzu.
3.1.0	Fehlerbehebungen und Verbesserungen • Fügt Unterstützung für HTTPS-Netzwerk-Proxy-Konfigurationen hinzu. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy und Ermöglichen Sie dem Kerngerät, einem HTTPS-Proxy zu vertrauen.
3.0.1	Behebt ein Problem damit, wie die Komponente Deltawerte für Metriken berechnet.
3.0.0	**Warning** Diese Version ist nicht mehr verfügbar. Die Verbesserungen in dieser Version sind in späteren Versionen dieser Komponente verfügbar. Erste Version

v2.x

Version	Änderungen
2.0.11	Version für Greengrass-Kern Version 2.11.0 aktualisiert.
2.0.10	Version für Greengrass-Kern Version 2.7.0 aktualisiert.
2.0.9	Version für Greengrass-Kern Version 2.6.0 aktualisiert.
2.0.8	Version für Greengrass-Kern Version 2.5.0 aktualisiert.
2.0.7	Version für Greengrass-Kern Version 2.4.0 aktualisiert.
2.0.6	Version für Greengrass-Kern Version 2.3.0 aktualisiert.
2.0.5	Version für Greengrass-Kern Version 2.2.0 aktualisiert.
2.0.4	Version für Greengrass-Kern Version 2.1.0 aktualisiert.
2.0.3	Erste Version

Festplatten-Spooler

Die Laufwerks-Spooler-Komponente (`aws.greengrass.DiskSpooler`) bietet eine persistente Speicheroption für Nachrichten, die von Greengrass-Core-Geräten zu gepoolt werden AWS IoT Core. Diese Komponente speichert diese ausgehenden Nachrichten auf der Festplatte.

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 1.0.x

Typ

Diese Komponente ist eine Plugin-Komponente (`aws.greengrass.plugin`). Der Greengrass-Kern führt diese Komponente in derselben Java Virtual Machine (JVM) wie der Kern aus. Der Kern wird neu gestartet, wenn Sie die Version dieser Komponente auf dem Core-Gerät ändern.

Diese Komponente verwendet dieselbe Protokolldatei wie der Greengrass-Kern. Weitere Informationen finden Sie unter Überwachen von AWS IoT Greengrass Protokollen.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf -Core-Geräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- `storageType` sollte auf gesetzt werden`Disk`, um diese Komponente zu verwenden. Sie können dies in der Greengrass-Kernkonfiguration festlegen.
- `maxSizeInBytes` darf nicht so konfiguriert sein, dass sie größer als der verfügbare Speicherplatz auf dem Gerät ist. Sie können dies in der Greengrass-Kernkonfiguration festlegen.
- Die Datenträger-Spooler-Komponente wird für die Ausführung in einer VPC unterstützt.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und alle

ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitstellen zu können. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente und die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste Abhängigkeiten.

1.0.1 – 1.0.3

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 1.0.1 bis 1.0.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.11.0 <2.13.0	Hart

1.0.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.0.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.11.0 <2.12.0	Hart

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zum Komponentenrezept .

Verwendung

Um die Disk-Spooler-Komponente verwenden zu können, `aws.greengrass.DiskSpooler` muss bereitgestellt werden.

Um diese Komponente zu konfigurieren und `pluginName` zu verwenden, müssen Sie auf setzen`aws.greengrass.DiskSpooler`.

Lokale Protokolldatei

Diese Komponente verwendet dieselbe Protokolldatei wie die Greengrass-Kernkomponente.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

So zeigen Sie die Protokolle dieser Komponente an

- Führen Sie den folgenden Befehl auf dem Core-Gerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

  Linux

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

  Windows (PowerShell)

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

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in jeder Version der Komponente beschrieben.

Version	Änderungen
1.0.3	Fehlerbehebungen und Verbesserungen Verbessert die Leistung durch die Wiederverwendung von Datenbank verbindungen.
1.0.2	Fehlerbehebungen und Verbesserungen Behebt ein Problem, bei dem das MQTT-Nachrichtenformatfeld in bestimmten Fällen nicht beibehalten wird.

Version	Änderungen
1.0.1	Version für Greengrass-Kern Version 2.12.0 aktualisiert.
1.0.0	Erste Version

Docker-Anwendungsmanager

Die Docker-Anwendungsmanager-Komponente (`aws.greengrass.DockerApplicationManager`) ermöglicht es AWS IoT Greengrass, Docker-Images aus öffentlichen Image-Registrys und privaten Registrierungen herunterzuladen, die in Amazon Elastic Container Registry (Amazon ECR) gehostet werden. Außerdem kann AWS IoT Greengrass Anmeldeinformationen automatisch verwalten, um Images sicher aus privaten Repositorys in Amazon ECR herunterzuladen.

Wenn Sie eine benutzerdefinierte Komponente entwickeln, die einen Docker-Container ausführt, schließen Sie den Docker-Anwendungsmanager als Abhängigkeit ein, um die Docker-Images herunterzuladen, die als Artefakte in Ihrer Komponente angegeben sind. Weitere Informationen finden Sie unter Führen Sie einen Docker-Container aus.

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.0.x

Typ

Diese Komponente ist eine generische Komponente (`aws.greengrass.generic`). Der Greengrass-Kern führt die Lebenszyklusskripte der Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf -Core-Geräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Docker Engine 1.9.1 oder höher ist auf dem Greengrass-Core-Gerät installiert. Version 20.10 ist die neueste Version, für die verifiziert wurde, dass sie mit der AWS IoT Greengrass Core-Software funktioniert. Sie müssen Docker direkt auf dem Core-Gerät installieren, bevor Sie Komponenten bereitstellen, die Docker-Container ausführen.
- Der Docker-Daemon wurde gestartet und wird auf dem Core-Gerät ausgeführt, bevor Sie diese Komponente bereitstellen.
- Docker-Images, die in einer der folgenden unterstützten Image-Quellen gespeichert sind:
  - Öffentliche und private Image-Repositorys in Amazon Elastic Container Registry (Amazon ECR)
  - Öffentliches Docker-Hub-Repository
  - Öffentliche vertrauenswürdige Docker-Registrierung
- Docker-Images, die als Artefakte in Ihren benutzerdefinierten Docker-Containerkomponenten enthalten sind. Verwenden Sie die folgenden URI-Formate, um Ihre Docker-Images anzugeben:
  - Privates Amazon-ECR-Image: `docker:`*`account-id`*`.dkr.ecr.`*`region`*`.amazonaws.com/`*`repository`*`/`*`image`*`[`*`:tag|@digest`*`]`

- Öffentliches Amazon-ECR-Image: `docker:public.ecr.aws/`*`repository`*`/`*`image`*`[:tag|@digest]`
- Öffentliches Docker-Hub-Image: `docker:`*`name`*`[:tag|@digest]`

Weitere Informationen finden Sie unter Führen Sie einen Docker-Container aus.

> **Note**
>
> Wenn Sie das Image-Tag oder den Image-Digest nicht im Artefakt-URI für ein Image angeben, ruft der Docker-Anwendungsmanager die neueste verfügbare Version dieses Images ab, wenn Sie Ihre benutzerdefinierte Docker-Containerkomponente bereitstellen. Um sicherzustellen, dass alle Ihre Core-Geräte dieselbe Version eines Images ausführen, empfehlen wir Ihnen, das Image-Tag oder den Image-Digest in den Artefakt-URI aufzunehmen.

- Der Systembenutzer, der eine Docker-Containerkomponente ausführt, muss über Root- oder Administratorberechtigungen verfügen, oder Sie müssen Docker so konfigurieren, dass es als Nicht-Root- oder Nicht-Administratorbenutzer ausgeführt wird.
  - Auf Linux-Geräten können Sie der `docker` Gruppe einen Benutzer hinzufügen, um `docker` Befehle ohne aufzurufen`sudo`.
  - Auf Windows-Geräten können Sie der `docker-users` Gruppe einen Benutzer hinzufügen, um `docker` Befehle ohne Administratorrechte aufzurufen.

  Linux or Unix

  Um oder den Nicht-Root-Benutzer`ggc_user`, den Sie zum Ausführen von Docker-Containerkomponenten verwenden, zur `docker` Gruppe hinzuzufügen, führen Sie den folgenden Befehl aus.

  ```
  sudo usermod -aG docker ggc_user
  ```

  Weitere Informationen finden Sie unter Verwalten von Docker als Nicht-Root-Benutzer.

  Windows Command Prompt (CMD)

  Um oder den Benutzer`ggc_user`, den Sie zum Ausführen von Docker-Containerkomponenten verwenden, zur `docker-users` Gruppe hinzuzufügen, führen Sie den folgenden Befehl als Administrator aus.

```
net localgroup docker-users ggc_user /add
```

Windows PowerShell

Um oder den Benutzer`ggc_user`, den Sie zum Ausführen von Docker-Containerkomponenten verwenden, zur `docker-users` Gruppe hinzuzufügen, führen Sie den folgenden Befehl als Administrator aus.

```
Add-LocalGroupMember -Group docker-users -Member ggc_user
```

- Wenn Sie die AWS IoT Greengrass Core-Software für die Verwendung eines Netzwerk-Proxys konfigurieren, müssen Sie Docker für die Verwendung desselben Proxy-Servers konfigurieren.
- Wenn Ihre Docker-Images in einer privaten Amazon-ECR-Registrierung gespeichert sind, müssen Sie die Token-Exchange-Servicekomponente als Abhängigkeit in die Docker-Containerkomponente aufnehmen. Außerdem muss die Greengrass-Geräterolle die `ecr:GetDownloadUrlForLayer` Aktionen `ecr:GetAuthorizationTokenecr:BatchGetImage`, und zulassen, wie in der folgenden Beispiel-IAM-Richtlinie gezeigt.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

- Die Docker-Application-Manager-Komponente wird für die Ausführung in einer VPC unterstützt. Um diese Komponente in einer VPC bereitzustellen, ist Folgendes erforderlich.

- Die Docker-Application-Manager-Komponente muss über Konnektivität verfügen, um Images herunterzuladen. Wenn Sie beispielsweise ECR verwenden, müssen Sie über Konnektivität zu den folgenden Endpunkten verfügen.
  - `*.dkr.ecr.region.amazonaws.com` (VPC-Endpunkt `com.amazonaws.region.ecr.dkr`)
  - `api.ecr.region.amazonaws.com` (VPC-Endpunkt `com.amazonaws.region.ecr.api`)

Endpunkte und Ports

Diese Komponente muss in der Lage sein, ausgehende Anfragen an die folgenden Endpunkte und Ports auszuführen, zusätzlich zu den für den grundlegenden Betrieb erforderlichen Endpunkten und Ports. Weitere Informationen finden Sie unter Zulassen von Gerätedatenverkehr über einen Proxy oder eine Firewall.

Endpunkt	Port	Erforderlich	Beschreibung
`ecr.region.amazonaws.com`	443	Nein	Erforderlich, wenn Sie Docker-Images von Amazon ECR herunterladen.

`hub.docker.com`
 	
`registry.hub.docker.com/v1`
	443	Nein	Erforderlich, wenn Sie Docker-Images von Docker Hub herunterladen.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und alle ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitzustellen. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente und die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste Abhängigkeiten.

2.0.11

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.11 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.1.0 <2.13.0	Weich

2.0.10

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.10 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.1.0 <2.12.0	Weich

2.0.9

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.9 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.1.0 <2.11.0	Weich

2.0.8

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.8 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.1.0 <2.10.0	Weich

2.0.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.1.0 <2.9.0	Weich

2.0.6

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.1.0 <2.8.0	Weich

2.0.5

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.1.0 <2.7.0	Weich

2.0.4

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.1.0 <2.6.0	Weich

2.0.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.1.0 <2.5.0	Weich

2.0.2

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.1.0 <2.4.0	Weich

2.0.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.1.0 <2.3.0	Weich

2.0.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.1.0 <2.2.0	Weich

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zum Komponentenrezept .

Konfiguration

Diese Komponente hat keine Konfigurationsparameter.

Lokale Protokolldatei

Diese Komponente verwendet dieselbe Protokolldatei wie die Greengrass-Kernkomponente.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

So zeigen Sie die Protokolle dieser Komponente an

- Führen Sie den folgenden Befehl auf dem Core-Gerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

  Linux

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

  Windows (PowerShell)

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

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in jeder Version der Komponente beschrieben.

Version	Änderungen
2.0.11	Version für Greengrass-Kern Version 2.12.0 aktualisiert.
2.0.10	Version für Greengrass-Kern Version 2.11.0 aktualisiert.
2.0.9	Version für Greengrass-Kernversion 2.10.0 aktualisiert.
2.0.8	Version für Greengrass-Kern Version 2.9.0 aktualisiert.
2.0.7	Version für Greengrass-Kern Version 2.8.0 aktualisiert.
2.0.6	Version für Greengrass-Kern Version 2.7.0 aktualisiert.
2.0.5	Version für Greengrass-Kern Version 2.6.0 aktualisiert.
2.0.4	Version für Greengrass-Kern Version 2.5.0 aktualisiert.
2.0.3	Version für Greengrass-Kern Version 2.4.0 aktualisiert.
2.0.2	Version für Greengrass-Kern Version 2.3.0 aktualisiert.
2.0.1	Version für Greengrass-Kern Version 2.2.0 aktualisiert.
2.0.0	Erste Version

Weitere Informationen finden Sie auch unter

- Führen Sie einen Docker-Container aus

Edge-Anschluss für Kinesis Video Streams

Der Edge-Connector für die Kinesis Video Streams-Komponente (`aws.iot.EdgeConnectorForKVS`) liest Video-Feeds von lokalen Kameras und veröffentlicht die Streams in Kinesis Video Streams. Sie können diese Komponente so konfigurieren, dass sie Video-Feeds von IP-Kameras (Internet Protocol) liest, die das Real Time Streaming Protocol (RTSP) verwenden. Anschließend können Sie Dashboards in Amazon Managed Grafana oder lokalen Grafana-Servern einrichten, um die Videostreams zu überwachen und mit ihnen zu interagieren.

Sie können diese Komponente integrieren AWS IoT TwinMaker , um Videostreams in Grafana-Dashboards anzuzeigen und zu steuern. AWS IoT TwinMaker ist ein AWS Service, mit dem Sie betriebsbereite digitale Zwillinge physischer Systeme erstellen können. Sie können AWS IoT TwinMaker damit Daten von Sensoren, Kameras und Unternehmensanwendungen visualisieren, sodass Sie Ihre physischen Fabriken, Gebäude oder Industrieanlagen verfolgen können. Sie können diese Daten auch verwenden, um Abläufe zu überwachen, Fehler zu diagnostizieren und Fehler zu reparieren. Weitere Informationen finden Sie unter Was ist AWS IoT TwinMaker? im AWS IoT TwinMaker Benutzerhandbuch.

Diese Komponente speichert ihre Konfiguration in AWS IoT SiteWise. Dabei handelt es sich um einen AWS Dienst, der Industriedaten modelliert und speichert. AWS IoT SiteWise In stehen Anlagen für Objekte wie Geräte, Ausrüstung oder Gruppen anderer Objekte. Um diese Komponente zu konfigurieren und zu verwenden, erstellen Sie ein AWS IoT SiteWise Asset für jedes Greengrass-Core-Gerät und für jede IP-Kamera, die mit jedem Core-Gerät verbunden ist. Jedes Asset verfügt über Eigenschaften, die Sie konfigurieren, um Funktionen wie Live-Streaming, On-Demand-Upload und lokales Caching zu steuern. Um die URL für jede Kamera anzugeben, erstellen Sie ein Geheimnis AWS Secrets Manager , das die URL der Kamera enthält. Wenn für die Kamera eine Authentifizierung erforderlich ist, geben Sie in der URL auch einen Benutzernamen und ein Passwort an. Anschließend geben Sie dieses Geheimnis in einer Asset-Eigenschaft für die IP-Kamera an.

Diese Komponente lädt den Videostream jeder Kamera in einen Kinesis-Videostream hoch. Sie geben den Namen des Kinesis-Ziel-Videostreams in der AWS IoT SiteWise Asset-Konfiguration für jede Kamera an. Wenn der Kinesis-Videostream nicht existiert, erstellt diese Komponente ihn für Sie.

AWS IoT TwinMaker stellt ein Skript bereit, das Sie ausführen können, um diese AWS IoT SiteWise Assets und Secrets Manager zu erstellen. Weitere Informationen zur Erstellung dieser Ressourcen und zur Installation, Konfiguration und Verwendung dieser Komponente finden Sie unter AWS IoT TwinMaker Videointegration im AWS IoT TwinMaker Benutzerhandbuch.

> **Note**
>
> Der Edge-Connector für die Kinesis Video Streams Streams-Komponente ist nur in den folgenden AWS-Regionen Fällen verfügbar:
>
> - USA Ost (Nord-Virginia)
> - USA West (Oregon)
> - Europe (Frankfurt)
> - Europa (Irland)

- Asien-Pazifik (Singapur)

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 1.0.x

Typ

Diese Komponente ist eine generische Komponente (`aws.greengrass.generic`). Der Greengrass-Kern führt die Lebenszyklusskripte der Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann nur auf Linux-Core-Geräten installiert werden.

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Sie können diese Komponente nur auf Single-Core-Geräten bereitstellen, da die Komponentenkonfiguration für jedes Core-Gerät eindeutig sein muss. Sie können diese Komponente nicht für Gruppen von Kerngeräten bereitstellen.
- GStreamer 1.18.4 oder höher ist auf dem Kerngerät installiert. Weitere Informationen finden Sie unter GStreamer installieren.

  Auf einem Gerät mit können Sie die folgenden Befehle ausführen`apt`, um GStreamer zu installieren.

  ```
  sudo apt install -y libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev
   gstreamer1.0-plugins-base-apps
  sudo apt install -y gstreamer1.0-libav
  sudo apt install -y gstreamer1.0-plugins-bad gstreamer1.0-plugins-good gstreamer1.0-
  plugins-ugly gstreamer1.0-tools
  ```
- Ein AWS IoT SiteWise Asset für jedes Kerngerät. Dieses AWS IoT SiteWise Asset stellt das Kerngerät dar. Weitere Informationen zur Erstellung dieses Assets finden Sie im AWS IoT TwinMaker Benutzerhandbuch unter AWS IoT TwinMaker Videointegration.
- Ein AWS IoT SiteWise Asset für jede IP-Kamera, die Sie mit jedem Kerngerät verbinden. Diese AWS IoT SiteWise Ressourcen stellen die Kameras dar, die Videos auf jedes Kerngerät streamen. Das Asset jeder Kamera muss dem Asset für das Kerngerät zugeordnet werden, das mit der Kamera verbunden ist. Kamera-Assets verfügen über Eigenschaften, die Sie konfigurieren können, um einen Kinesis-Videostream, ein Authentifizierungsgeheimnis und Videostreaming-Parameter anzugeben. Weitere Informationen zum Erstellen und Konfigurieren von Kameraobjekten finden Sie im AWS IoT TwinMaker Benutzerhandbuch unter AWS IoT TwinMaker Videointegration.
- Ein AWS Secrets Manager Geheimnis für jede IP-Kamera. Dieses Geheimnis muss ein Schlüssel-Wert-Paar definieren, wobei sich der Schlüssel befindet `RTSPStreamUrl` und der Wert die URL für die Kamera ist. Wenn für die Kamera eine Authentifizierung erforderlich ist, geben Sie den Benutzernamen und das Passwort in diese URL ein. Sie können ein Skript verwenden, um ein Geheimnis zu erstellen, wenn Sie die Ressourcen erstellen, die diese Komponente benötigt. Weitere Informationen finden Sie im AWS IoT TwinMaker Benutzerhandbuch unter AWS IoT TwinMaker Videointegration.

  Sie können auch die Secrets Manager Manager-Konsole und die API verwenden, um zusätzliche Geheimnisse zu erstellen. Weitere Informationen finden Sie im AWS Secrets Manager Benutzerhandbuch unter Create a Secret.

- Die Greengrass-Token-Austauschrolle muss die folgenden AWS Secrets Manager AWS IoT SiteWise, und Kinesis Video Streams Streams-Aktionen zulassen, wie in der folgenden IAM-Beispielrichtlinie gezeigt.

> **Note**
>
> Diese Beispielrichtlinie ermöglicht es dem Gerät, den Wert von Geheimnissen mit dem Namen und abzurufen. **`IPCamera1Url`** **`IPCamera2Url`** Wenn Sie jede IP-Kamera konfigurieren, geben Sie ein Geheimnis an, das die URL für diese Kamera enthält. Wenn für die Kamera eine Authentifizierung erforderlich ist, geben Sie in der URL auch einen Benutzernamen und ein Passwort an. Die Token-Austauschfunktion des Kerngeräts muss den Zugriff auf das Geheimnis ermöglichen, damit jede IP-Kamera eine Verbindung herstellen kann.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:secretsmanager:region:account-id:secret:IPCamera1Url",
        "arn:aws:secretsmanager:region:account-id:secret:IPCamera2Url"
      ]
    },
    {
      "Action": [
        "iotsitewise:BatchPutAssetPropertyValue",
        "iotsitewise:DescribeAsset",
        "iotsitewise:DescribeAssetModel",
        "iotsitewise:DescribeAssetProperty",
        "iotsitewise:GetAssetPropertyValue",
        "iotsitewise:ListAssetRelationships",
        "iotsitewise:ListAssets",
        "iotsitewise:ListAssociatedAssets",
        "kinesisvideo:CreateStream",
        "kinesisvideo:DescribeStream",
        "kinesisvideo:GetDataEndpoint",
```

```
        "kinesisvideo:PutMedia",
        "kinesisvideo:TagStream"
      ],
      "Effect": "Allow",
      "Resource": [
        "*"
      ]
    }
  ]
}
```

> **Note**
>
> Wenn Sie einen vom Kunden verwalteten AWS Key Management Service Schlüssel zum Verschlüsseln von Geheimnissen verwenden, muss die Geräterolle die `kms:Decrypt` Aktion ebenfalls zulassen.

Endpunkte und Ports

Diese Komponente muss in der Lage sein, ausgehende Anfragen an die folgenden Endpunkte und Ports auszuführen, zusätzlich zu den Endpunkten und Ports, die für den Basisbetrieb erforderlich sind. Weitere Informationen finden Sie unter Zulassen von Gerätedatenverkehr über einen Proxy oder eine Firewall.

Endpunkt	Port	Erforderlich	Beschreibung
`kinesisvideo.` *`region`*`.amazonaws.com`	443	Ja	Laden Sie Daten in Kinesis Video Streams hoch.
`data.iotsitewise.` *`region`*`.amazonaws.com`	443	Ja	Veröffentlichen Sie Videostream-Metada

Endpunkt	Port	Erforderlich	Beschreibung
			ten auf AWS IoT SiteWise.
`secretsmanager. region.amazonaws.com`	443	Ja	Laden Sie die Geheimnisse der Kamera-URL auf das Kerngerät herunter.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt sie AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und all ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitstellen zu können. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente sowie die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste der Abhängigkeiten.

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 1.0.0 bis 1.0.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Token-Austauschdienst	>=2.0.3	Hart
Stream-Manager	>=2.0.9	Hart

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zu den Komponentenrezepten.

Konfiguration

Diese Komponente stellt die folgenden Konfigurationsparameter bereit, die Sie bei der Bereitstellung der Komponente anpassen können.

`SiteWiseAssetIdForHub`

Die ID des AWS IoT SiteWise Assets, das dieses Kerngerät darstellt. Weitere Informationen darüber, wie Sie dieses Asset erstellen und es für die Interaktion mit dieser Komponente verwenden, finden Sie im AWS IoT TwinMaker Benutzerhandbuch unter AWS IoT TwinMaker Videointegration.

Example Beispiel: Aktualisierung der Zusammenführung von Konfigurationen

```
{
  "SiteWiseAssetIdForHub": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
}
```

Lizenzen

Diese Komponente umfasst die folgende Software/Lizenzierung von Drittanbietern:

- Quartz Job Scheduler//Apache License 2.0
- Java-Bindungen für GStreamer 1.x//GNU Lesser General Public License v3.0

Verwendung

Um diese Komponente zu konfigurieren und mit ihr zu interagieren, können Sie Eigenschaften für die Komponenten festlegen, die das Kerngerät und AWS IoT SiteWise die IP-Kameras repräsentieren, mit denen es verbunden ist. Sie können auch Videostreams in Grafana-Dashboards über visualisieren und mit ihnen interagieren. AWS IoT TwinMaker Weitere Informationen finden Sie unter AWS IoT TwinMaker Videointegration im AWS IoT TwinMaker Benutzerhandbuch.

Lokale Protokolldatei

Diese Komponente verwendet die folgende Protokolldatei.

```
/greengrass/v2/logs/aws.iot.EdgeConnectorForKVS.log
```

Um die Protokolle dieser Komponente einzusehen

- Führen Sie den folgenden Befehl auf dem Kerngerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. */greengrass/v2*Ersetzen Sie es durch den Pfad zum AWS IoT Greengrass Stammordner.

  ```
  sudo tail -f /greengrass/v2/logs/aws.iot.EdgeConnectorForKVS.log
  ```

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
1.0.5	Allgemeine Fehlerbehebungen und Verbesserungen.
1.0.4	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, das dazu führte, dass das Live-Hochladen gestoppt wurde.
1.0.3	Allgemeine Fehlerbehebungen und Verbesserungen.
1.0.1	Allgemeine Fehlerbehebungen und Verbesserungen.
1.0.0	Erste Version

Weitere Informationen finden Sie auch unter

- Was ist AWS IoT TwinMaker? im AWS IoT TwinMaker Benutzerhandbuch
- AWS IoT TwinMaker Videointegration im AWS IoT TwinMaker Benutzerhandbuch
- Was ist AWS IoT SiteWise? im AWS IoT SiteWise Benutzerhandbuch
- Aktualisierung von Attributwerten im AWS IoT SiteWise Benutzerhandbuch
- Was ist AWS Secrets Manager? im AWS Secrets Manager Benutzerhandbuch

- Im AWS Secrets Manager Benutzerhandbuch können Sie Geheimnisse erstellen und verwalten

Greengrass CLI

Die Greengrass-CLI-Komponente (`aws.greengrass.Cli`) bietet eine lokale Befehlszeilenschnittstelle, die Sie auf Kerngeräten verwenden können, um Komponenten lokal zu entwickeln und zu debuggen. Mit der Greengrass-CLI können Sie beispielsweise lokale Bereitstellungen erstellen und Komponenten auf dem Kerngerät neu starten.

Sie können diese Komponente bei der Installation der AWS IoT Greengrass Core-Software installieren. Weitere Informationen finden Sie unter Tutorial: Erste Schritte mit AWS IoT Greengrass V2.

> **Important**
>
> Wir empfehlen, diese Komponente nur in Entwicklungsumgebungen und nicht in Produktionsumgebungen zu verwenden. Diese Komponente bietet Zugriff auf Informationen und Operationen, die Sie in einer Produktionsumgebung normalerweise nicht benötigen. Folgen Sie dem Prinzip der geringsten Rechte, indem Sie diese Komponente nur dort einsetzen, wo Sie sie benötigen.

Führen Sie nach der Installation dieser Komponente den folgenden Befehl aus, um die zugehörige Hilfedokumentation aufzurufen. Bei der Installation dieser Komponente wird dem */greengrass/v2*`/bin` Ordner ein symbolischer Link hinzugefügt. `greengrass-cli` Sie können die Greengrass-CLI von diesem Pfad aus ausführen oder sie zu Ihrer `PATH` Umgebungsvariablen hinzufügen, um sie `greengrass-cli` ohne ihren absoluten Pfad auszuführen.

Linux or Unix

```
/greengrass/v2/bin/greengrass-cli help
```

Windows

```
C:\greengrass\v2\bin\greengrass-cli help
```

Der folgende Befehl startet beispielsweise eine Komponente mit dem Namen `com.example.HelloWorld` neu.

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli component restart --names
 "com.example.HelloWorld"
```

Windows

```
C:\greengrass\v2\bin\greengrass-cli component restart --names
 "com.example.HelloWorld"
```

Weitere Informationen finden Sie unter Greengrass-Befehlszeilenschnittstelle.

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.12.x
- 2.11.x
- 2.10.x
- 2.9.x
- 2.8.x

- 2.7.x
- 2.6.x
- 2.5.x
- 2.4.x
- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Typ

Diese Komponente ist eine Plugin-Komponente (`aws.greengrass.plugin`). Der Greengrass-Kern führt diese Komponente in derselben Java Virtual Machine (JVM) wie der Nucleus aus. Der Nucleus wird neu gestartet, wenn Sie die Version dieser Komponente auf dem Core-Gerät ändern.

Diese Komponente verwendet dieselbe Protokolldatei wie der Greengrass-Kern. Weitere Informationen finden Sie unter Überwachen von AWS IoT Greengrass Protokollen.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf Kerngeräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Sie müssen autorisiert sein, die Greengrass-CLI zu verwenden, um mit der AWS IoT Greengrass Core-Software zu interagieren. Führen Sie einen der folgenden Schritte aus, um die Greengrass-CLI zu verwenden:
  - Verwenden Sie den Systembenutzer, der die AWS IoT Greengrass Core-Software ausführt.

- Verwenden Sie einen Benutzer mit Root- oder Administratorrechten. Auf Linux-Core-Geräten können Sie diese Option verwenden, um `sudo` Root-Rechte zu erhalten.
- Verwenden Sie einen Systembenutzer, der zu einer Gruppe gehört, die Sie bei der Bereitstellung der Komponente in den `AuthorizedWindowsGroups` Konfigurationsparametern `AuthorizedPosixGroups` oder angeben. Weitere Informationen finden Sie unter Konfiguration der Greengrass-CLI-Komponente.

- Die Greengrass-CLI-Komponente wird für die Ausführung in einer VPC unterstützt.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt sie AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und all ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitstellen zu können. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente sowie die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste der Abhängigkeiten.

2.12.0 – 2.12.6

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.12.0 bis 2.12.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.12.0 <2.13.0	Weich

2.11.0 – 2.11.3

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.11.0 bis 2.11.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.11.0 <2.12.0	Weich

2.10.0 – 2.10.3

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.10.0 bis 2.10.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2,5,0 <2,11,0	Weich

2.9.0 – 2.9.6

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.9.0 bis 2.9.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.5.0 <2.10.0	Weich

2.8.0 – 2.8.1

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.8.0 und 2.8.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2,5,0 <2,9,0	Weich

2.7.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.7.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2,5,0 <2,8,0	Weich

2.6.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.6.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2,5,0 <2,7,0	Weich

2.5.0 – 2.5.6

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.5.0 bis 2.5.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2,5,0 <2,6,0	Weich

2.4.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.4.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.1.0 <2.5.0	Weich

2.3.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.3.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.1.0 <2.4.0	Weich

2.2.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.2.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.1.0 <2.3.0	Weich

2.1.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.1.0 <2.2.0	Weich

2.0.x

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.x dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.1.0	Weich

> **Note**
>
> Die minimale kompatible Version des Greengrass-Nukleus entspricht der Patch-Version der Greengrass-CLI-Komponente.

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zu den Komponentenrezepten.

Konfiguration

Diese Komponente stellt die folgenden Konfigurationsparameter bereit, die Sie bei der Bereitstellung der Komponente anpassen können.

2.5.x - 2.12.x

`AuthorizedPosixGroups`

(Optional) Eine Zeichenfolge, die eine durch Kommas getrennte Liste von Systemgruppen enthält. Sie autorisieren diese Systemgruppen, die Greengrass-CLI für die Interaktion mit der AWS IoT Greengrass Core-Software zu verwenden. Sie können Gruppennamen oder Gruppen-IDs angeben. `group1,1002,group3`Autorisiert beispielsweise drei Systemgruppen (`group1`, und`group3`)`1002`, die Greengrass-CLI zu verwenden.

Wenn Sie keine zu autorisierenden Gruppen angeben, können Sie die Greengrass-CLI als Root-Benutzer (`sudo`) oder als Systembenutzer verwenden, der die AWS IoT Greengrass Core-Software ausführt.

`AuthorizedWindowsGroups`

(Optional) Eine Zeichenfolge, die eine durch Kommas getrennte Liste von Systemgruppen enthält. Sie autorisieren diese Systemgruppen, die Greengrass-CLI für die Interaktion mit der AWS IoT Greengrass Core-Software zu verwenden. Sie können Gruppennamen oder Gruppen-IDs angeben. `group1,1002,group3`Autorisiert beispielsweise drei Systemgruppen (`group1`, und`group3`)`1002`, die Greengrass-CLI zu verwenden.

Wenn Sie keine zu autorisierenden Gruppen angeben, können Sie die Greengrass-CLI als Administrator oder als Systembenutzer verwenden, der die AWS IoT Greengrass Core-Software ausführt.

Example Beispiel: Aktualisierung der Konfigurationszusammenführung

Die folgende Beispielkonfiguration legt fest, dass drei POSIX-Systemgruppen (`group11002`, und`group3`) und zwei Windows-Benutzergruppen (`Device OperatorsundQA Engineers`) zur Verwendung der Greengrass-CLI autorisiert werden sollen.

```
{
  "AuthorizedPosixGroups": "group1,1002,group3",
  "AuthorizedWindowsGroups": "Device Operators,QA Engineers"
}
```

2.4.x - 2.0.x

`AuthorizedPosixGroups`

(Optional) Eine Zeichenfolge, die eine durch Kommas getrennte Liste von Systemgruppen enthält. Sie autorisieren diese Systemgruppen, die Greengrass-CLI für die Interaktion mit der AWS IoT Greengrass Core-Software zu verwenden. Sie können Gruppennamen oder Gruppen-IDs angeben. `group1,1002,group3`Autorisiert beispielsweise drei Systemgruppen (`group1`, und`group3`)`1002`, die Greengrass-CLI zu verwenden.

Wenn Sie keine zu autorisierenden Gruppen angeben, können Sie die Greengrass-CLI als Root-Benutzer (`sudo`) oder als Systembenutzer verwenden, der die AWS IoT Greengrass Core-Software ausführt.

Example Beispiel: Aktualisierung der Konfigurationszusammenführung

Die folgende Beispielkonfiguration legt fest, dass drei Systemgruppen (`group1`1002, und`group3`) zur Verwendung der Greengrass-CLI autorisiert werden sollen.

```
{
  "AuthorizedPosixGroups": "group1,1002,group3"
}
```

Lokale Protokolldatei

Diese Komponente verwendet dieselbe Protokolldatei wie die Greengrass Nucleus-Komponente.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

Um die Protokolle dieser Komponente einzusehen

- Führen Sie den folgenden Befehl auf dem Kerngerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

  Linux

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

  Windows (PowerShell)

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

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
2.12.6	Die Version wurde für die Version 2.12.6 von Greengrass Nucleus aktualisiert.
2.12.5	Die Version wurde für die Version 2.12.5 von Greengrass Nucleus aktualisiert.
2.12.4	Die Version wurde für die Version 2.12.4 von Greengrass Nucleus aktualisiert.
2.12.3	**Warning** Diese Version ist nicht mehr verfügbar. Die Verbesserungen in dieser Version sind in späteren Versionen dieser Komponente verfügbar. Die Version wurde für die Version 2.12.3 von Greengrass Nucleus aktualisiert.
2.12.2	Die Version wurde für die Version 2.12.2 von Greengrass Nucleus aktualisiert.
2.12.1	Die Version wurde für die Version 2.12.1 von Greengrass Nucleus aktualisiert.

Version	Änderungen
2.12.0	Die Version wurde für die Version 2.12.0 von Greengrass Nucleus aktualisiert.
2.11.3	Die Version wurde für die Version 2.11.3 von Greengrass Nucleus aktualisiert.
2.11.2	Die Version wurde für die Version 2.11.2 von Greengrass Nucleus aktualisiert.
2.11.1	Die Version wurde für die Version 2.11.1 von Greengrass Nucleus aktualisiert.
2.11.0	Neue Features • Ermöglicht es Ihnen, eine lokale Bereitstellung abzubrechen. • Ermöglicht die Konfiguration einer Fehlerbehandlungsrichtlinie für eine lokale Bereitstellung. • Verbessert die detaillierte Berichterstattung zum Bereitstellungsstatus.
2.10.3	Die Version wurde für die Version 2.10.3 von Greengrass Nucleus aktualisiert.
2.10.2	Die Version wurde für die Version 2.10.2 von Greengrass Nucleus aktualisiert.
2.10.1	Die Version wurde für die Version 2.10.1 von Greengrass Nucleus aktualisiert.
2.10.0	Die Version wurde für die Version 2.10.0 von Greengrass Nucleus aktualisiert.
2.9.6	Die Version wurde für die Version 2.9.6 von Greengrass Nucleus aktualisiert.
2.9.5	Die Version wurde für die Version 2.9.5 von Greengrass Nucleus aktualisiert.
2.9.4	Die Version wurde für die Version 2.9.4 von Greengrass Nucleus aktualisiert.
2.9.3	Die Version wurde für die Version 2.9.3 von Greengrass Nucleus aktualisiert.
2.9.2	Die Version wurde für die Version 2.9.2 von Greengrass Nucleus aktualisiert.
2.9.1	Die Version wurde für die Version 2.9.1 von Greengrass Nucleus aktualisiert.
2.9.0	Die Version wurde für die Version 2.9.0 von Greengrass Nucleus aktualisiert.
2.8.1	Die Version wurde für die Version 2.8.1 von Greengrass Nucleus aktualisiert.
2.8.0	Die Version wurde für die Version 2.8.0 von Greengrass Nucleus aktualisiert.

Version	Änderungen
2.7.0	Die Version wurde für die Version 2.7.0 von Greengrass Nucleus aktualisiert.
2.6.0	Neue Features • Integriert die Unterstützung für benutzerdefinierte Komponenten zum Aufrufen von Interprozesskommunikationsoperationen (IPC), die von der Greengrass-CLI verwendet werden. Sie können diese IPC-Operationen verwenden, um lokale Bereitstellungen zu verwalten, Komponentendetails anzuzeigen und ein Passwort zu generieren, mit dem Sie sich bei der lokalen Debug-Konsole anmelden können. Weitere Informationen finden Sie unter IPC: Lokale Bereitstellungen und Komponenten verwalten. Fehlerkorrekturen und Verbesserungen • Zusätzliche kleinere Korrekturen und Verbesserungen.
2.5.6	Die Version wurde für die Version 2.5.6 von Greengrass Nucleus aktualisiert.
2.5.5	Die Version wurde für die Version 2.5.5 von Greengrass Nucleus aktualisiert.
2.5.4	Die Version wurde für die Version 2.5.4 von Greengrass Nucleus aktualisiert.
2.5.3	Die Version wurde für die Version 2.5.3 von Greengrass Nucleus aktualisiert.
2.5.2	Die Version wurde für die Version 2.5.2 von Greengrass Nucleus aktualisiert.
2.5.1	Die Version wurde für die Version 2.5.1 von Greengrass Nucleus aktualisiert.

Version	Änderungen
2.5.0	Neue Features • Fügt Unterstützung für Kerngeräte hinzu, auf denen Windows ausgeführt wird. • Fügt den neuen `AuthorizedWindowsGroups` Konfigurationsparameter hinzu, den Sie angeben können, um Systemgruppen zur Verwendung der Greengrass-CLI auf Windows-Geräten zu autorisieren. • Fügt den `windowsUser` Parameter für lokale Bereitstellungen hinzu. Mit diesem Parameter können Sie den Benutzer angeben, der zum Ausführen von Komponenten auf einem Windows-Core-Gerät verwendet werden soll.
2.4.0	Neue Features • Fügt Unterstützung für Systemressourcenlimits hinzu. Wenn Sie eine lokale Bereitstellung erstellen, können Sie die maximale CPU- und RAM-Auslastung konfigurieren, die die Prozesse der einzelnen Komponenten auf dem Kerngerät verwenden können. Weitere Informationen finden Sie unter Konfigurieren Sie die Systemressourcenlimits für Komponenten und dem Befehl deployment create.
2.3.0	Die Version wurde für die Version 2.3.0 von Greengrass Nucleus aktualisiert.
2.2.0	Die Version wurde für die Version 2.2.0 von Greengrass Nucleus aktualisiert.
2.1.0	Die Version wurde für die Version 2.1.0 von Greengrass Nucleus aktualisiert.
2.0.5	Die Version wurde für die Version 2.0.5 von Greengrass Nucleus aktualisiert.
2.0.4	Die Version wurde für die Version 2.0.4 von Greengrass Nucleus aktualisiert.
2.0.3	Erste Version

IP-Detektor

Die IP-Detektorkomponente (`aws.greengrass.clientdevices.IPDetector`) macht Folgendes:

- Überwacht die Netzwerkverbindungsinformationen des Greengrass-Core-Geräts. Zu diesen Informationen gehören die Netzwerkendpunkte des Kerngeräts und der Port, an dem ein MQTT-Broker betrieben wird.
- Aktualisiert die Konnektivitätsinformationen des Kerngeräts im AWS IoT Greengrass Cloud-Dienst.

Client-Geräte können Greengrass Cloud Discovery verwenden, um die Verbindungsinformationen der zugehörigen Kerngeräte abzurufen. Anschließend können Client-Geräte versuchen, eine Verbindung zu jedem Kerngerät herzustellen, bis sie erfolgreich eine Verbindung herstellen.

> **Note**
>
> Client-Geräte sind lokale IoT-Geräte, die eine Verbindung zu einem Greengrass-Core-Gerät herstellen, um MQTT-Nachrichten und Daten zur Verarbeitung zu senden. Weitere Informationen finden Sie unter Interagieren mit lokalen IoT-Geräten.

Die IP-Detektorkomponente ersetzt die vorhandenen Konnektivitätsinformationen eines Kerngeräts durch die Informationen, die es erkennt. Da diese Komponente vorhandene Informationen entfernt, können Sie entweder die IP-Detektorkomponente verwenden oder die Verbindungsinformationen manuell verwalten.

> **Note**
>
> Die IP-Detektorkomponente erkennt nur IPv4-Adressen.

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.1.x
- 2.0.x

Typ

Diese Komponente ist eine Plugin-Komponente (`aws.greengrass.plugin`). Der Greengrass-Kern führt diese Komponente in derselben Java Virtual Machine (JVM) wie der Nucleus aus. Der Nucleus wird neu gestartet, wenn Sie die Version dieser Komponente auf dem Core-Gerät ändern.

Diese Komponente verwendet dieselbe Protokolldatei wie der Greengrass-Kern. Weitere Informationen finden Sie unter Überwachen von AWS IoT Greengrass Protokollen.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf Kerngeräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Die Greengrass-Servicerolle muss mit Ihren Berechtigungen verknüpft sein AWS-Konto `iot:GetThingShadow` und die `iot:UpdateThingShadow` Berechtigungen zulassen.
- Die AWS IoT Richtlinie des Kerngeräts muss die `greengrass:UpdateConnectivityInfo` Genehmigung zulassen. Weitere Informationen finden Sie unter AWS IoT-Richtlinien für Operationen auf Datenebene und Minimale AWS IoT Richtlinie zur Unterstützung von Client-Geräten.
- Wenn Sie die MQTT-Broker-Komponente des Kerngeräts so konfigurieren, dass sie einen anderen Port als den Standardport 8883 verwendet, müssen Sie den IP-Detektor v2.1.0 oder höher verwenden. Konfigurieren Sie es so, dass der Port gemeldet wird, an dem der Broker arbeitet.

- Wenn Sie ein komplexes Netzwerk-Setup haben, kann die IP-Detektorkomponente möglicherweise nicht die Endpunkte identifizieren, an denen Client-Geräte eine Verbindung zum Kerngerät herstellen können. Wenn die IP-Detektorkomponente die Endpunkte nicht verwalten kann, müssen Sie stattdessen die Endpunkte der Kerngeräte manuell verwalten. Wenn sich das Kerngerät beispielsweise hinter einem Router befindet, der den MQTT-Broker-Port an ihn weiterleitet, müssen Sie die IP-Adresse des Routers als Endpunkt für das Kerngerät angeben. Weitere Informationen finden Sie unter Verwalten von -Core-Geräteendpunkten.
- Die IP-Detektorkomponente wird für die Ausführung in einer VPC unterstützt.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt sie AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und all ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitstellen zu können. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente sowie die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste der Abhängigkeiten.

2.1.8 – 2.1.9

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.1.8 und 2.1.9 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.2.0 <2.13.0	Weich

2.1.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.2.0 <2.12.0	Weich

2.1.6

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.2.0 <2.11.0	Weich

2.1.5

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.2.0 <2.10.0	Weich

2.1.4

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.2.0 <2.9.0	Weich

2.1.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.2.0 <2.8.0	Weich

2.1.2

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.2.0 <2.7.0	Weich

2.1.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.2.0 <2.6.0	Weich

2.1.0 and 2.0.2

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.1.0 und 2.0.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.2.0 <2.5.0	Weich

2.0.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.2.0 <2.4.0	Weich

2.0.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.2.0 <2.3.0	Weich

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zu den Komponentenrezepten.

Konfiguration

Diese Komponente stellt die folgenden Konfigurationsparameter bereit, die Sie bei der Bereitstellung der Komponente anpassen können.

2.1.x

`defaultPort`

(Optional) Der MQTT-Broker-Port, der gemeldet werden soll, wenn diese Komponente IP-Adressen erkennt. Sie müssen diesen Parameter angeben, wenn Sie den MQTT-Broker so konfigurieren, dass er einen anderen Port als den Standardport 8883 verwendet.

Standard: `8883`

`includeIPv4LoopbackAddrs`

(Optional) Sie können diese Option aktivieren, um IPv4-Loopback-Adressen zu erkennen und zu melden. Dies sind IP-Adressen, mit denen ein Gerät `localhost` beispielsweise mit sich selbst kommunizieren kann. Verwenden Sie diese Option in Testumgebungen, in denen das Kerngerät und das Client-Gerät auf demselben System ausgeführt werden.

Standard: `false`

`includeIPv4LinkLocalAddrs`

(Optional) Sie können diese Option aktivieren, um link-lokale IPv4-Adressen zu erkennen und zu melden. Verwenden Sie diese Option, wenn das Netzwerk des Kerngeräts nicht über DHCP (Dynamic Host Configuration Protocol) oder statisch zugewiesene IP-Adressen verfügt.

Standard: `false`

2.0.x

`includeIPv4LoopbackAddrs`

(Optional) Sie können diese Option aktivieren, um IPv4-Loopback-Adressen zu erkennen und zu melden. Dies sind IP-Adressen, mit denen ein Gerät `localhost` beispielsweise mit sich

selbst kommunizieren kann. Verwenden Sie diese Option in Testumgebungen, in denen das Kerngerät und das Client-Gerät auf demselben System ausgeführt werden.

Standard: `false`

`includeIPv4LinkLocalAddrs`

(Optional) Sie können diese Option aktivieren, um link-lokale IPv4-Adressen zu erkennen und zu melden. Verwenden Sie diese Option, wenn das Netzwerk des Kerngeräts nicht über DHCP (Dynamic Host Configuration Protocol) oder statisch zugewiesene IP-Adressen verfügt.

Standard: `false`

Lokale Protokolldatei

Diese Komponente verwendet dieselbe Protokolldatei wie die Greengrass Nucleus-Komponente.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

Um die Protokolle dieser Komponente einzusehen

- Führen Sie den folgenden Befehl auf dem Kerngerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

  Linux

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

  Windows (PowerShell)

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

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
2.1.9	Fehlerkorrekturen und Verbesserungen • Passt den Schritt zur Erfassung der IP-Adresse so an, dass nur Protokolle auf Debug-Protokollebene gesendet werden.
2.1.8	Die Version wurde für die Version 2.12.0 von Greengrass Nucleus aktualisiert.
2.1.7	Die Version wurde für die Version 2.11.0 von Greengrass Nucleus aktualisiert.
2.1.6	Die Version wurde für die Version 2.10.0 von Greengrass Nucleus aktualisiert.
2.1.5	Die Version wurde für die Version 2.9.0 von Greengrass Nucleus aktualisiert.
2.1.4	Die Version wurde für die Version 2.8.0 von Greengrass Nucleus aktualisiert.
2.1.3	Die Version wurde für die Version 2.7.0 von Greengrass Nucleus aktualisiert.
2.1.2	Fehlerkorrekturen und Verbesserungen • Verbessert die Fehlermeldungen, die diese Komponente in bestimmten Szenarien protokolliert. • Die Version wurde für die Version 2.6.0 von Greengrass Nucleus aktualisiert.
2.1.1	Die Version wurde für die Version 2.5.0 von Greengrass Nucleus aktualisiert.
2.1.0	Verbesserungen • Fügt den `defaultPort` Parameter hinzu, der es Ihnen ermöglicht, einen nicht standardmäßigen MQTT-Broker-Port zu verwenden. • Aktualisierungen, um Protokollnachrichten übersichtlicher zu machen.
2.0.2	Die Version wurde für die Version 2.4.0 von Greengrass Nucleus aktualisiert.
2.0.1	Die Version wurde für die Version 2.3.0 von Greengrass Nucleus aktualisiert.

Version	Änderungen
2.0.0	Erste Version

Firehose

Die Firehose-Komponente (`aws.greengrass.KinesisFirehose`) veröffentlicht Daten über Amazon-Data-Firehose-Bereitstellungsdatenströme an Ziele wie Amazon S3, Amazon Redshift und Amazon OpenSearch Service. Weitere Informationen finden Sie unter Was ist Amazon Data Firehose? im Entwicklerhandbuch für Amazon Data Firehose.

Um mit dieser Komponente in einem Kinesis-Bereitstellungs-Stream zu veröffentlichen, veröffentlichen Sie eine Nachricht zu einem Thema, in dem diese Komponente abonniert wird. Standardmäßig abonniert diese Komponente die `kinesisfirehose/message/binary/#` Themen `kinesisfirehose/message` und lokales Veröffentlichen/Abonnieren. Sie können bei der Bereitstellung dieser Komponente andere Themen angeben, einschließlich AWS IoT Core MQTT-Themen.

> **Note**
>
> Diese Komponente bietet ähnliche Funktionen wie der Firehose-Konnektor in AWS IoT Greengrass V1. Weitere Informationen finden Sie unter Firehose-Konnektor im AWS IoT Greengrass V1-Entwicklerhandbuch.

Themen

- Lizenzen
- Änderungsprotokoll
- Weitere Informationen finden Sie auch unter

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.1.x
- 2.0.x

Typ

Diese Komponente ist eine Lambda-Komponente (`aws.greengrass.lambda`). Der Greengrass-Kern führt die Lambda-Funktion dieser Komponente mit der Lambda-Launcher-Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann nur auf Linux-Core-Geräten installiert werden.

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Ihr Core-Gerät muss die Anforderungen erfüllen, um Lambda-Funktionen auszuführen. Wenn Sie möchten, dass das Core-Gerät containerisierte Lambda-Funktionen ausführt, muss das Gerät die Voraussetzungen dafür erfüllen. Weitere Informationen finden Sie unter Anforderungen an die Lambda-Funktion.
- Python Version 3.7 ist auf dem Core-Gerät installiert und der PATH-Umgebungsvariablen hinzugefügt.
- Die Greengrass-Geräterolle muss die `firehose:PutRecordBatch` Aktionen `firehose:PutRecord` und zulassen, wie in der folgenden Beispiel-IAM-Richtlinie gezeigt.

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Action": [
        "firehose:PutRecord",
        "firehose:PutRecordBatch"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:firehose:region:account-id:deliverystream/stream-name"
      ]
    }
  ]
}
```

Sie können den Standard-Bereitstellungsstream in der Nutzlast der Eingabenachricht für diese Komponente dynamisch überschreiben. Wenn Ihre Anwendung diese Funktion verwendet, muss die IAM-Richtlinie alle Zielstreams als Ressourcen enthalten. Sie können Ressourcen granularen oder bedingten Zugriff gewähren (etwa mit einem Benennungsschema mit Platzhaltern *).

- Um Ausgabedaten von dieser Komponente zu erhalten, müssen Sie das folgende Konfigurationsupdate für die Legacy-Abonnement-Routerkomponente (`aws.greengrass.LegacySubscriptionRouter`) zusammenführen, wenn Sie diese Komponente bereitstellen. Diese Konfiguration gibt das Thema an, in dem diese Komponente Antworten veröffentlicht.

  Legacy subscription router v2.1.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-kinesisfirehose": {
        "id": "aws-greengrass-kinesisfirehose",
        "source": "component:aws.greengrass.KinesisFirehose",
        "subject": "kinesisfirehose/message/status",
        "target": "cloud"
      }
    }
  }
  ```

  Legacy subscription router v2.0.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-kinesisfirehose": {
  ```

```
      "id": "aws-greengrass-kinesisfirehose",
      "source": "arn:aws:lambda:region:aws:function:aws-greengrass-
kinesisfirehose:version",
      "subject": "kinesisfirehose/message/status",
      "target": "cloud"
    }
  }
}
```

- Ersetzen Sie *region* durch die AWS-Region , die Sie verwenden.
- Ersetzen Sie *Version* durch die Version der Lambda-Funktion, die diese Komponente ausführt. Um die Version der Lambda-Funktion zu finden, müssen Sie das Rezept für die Version dieser Komponente anzeigen, die Sie bereitstellen möchten. Öffnen Sie die Detailseite dieser Komponente in der AWS IoT Greengrass Konsole und suchen Sie nach dem Schlüssel-Wert-Paar der Lambda-Funktion. Dieses Schlüssel-Wert-Paar enthält den Namen und die Version der Lambda-Funktion.

> **Important**
>
> Sie müssen die Lambda-Funktionsversion auf dem Legacy-Abonnement-Router jedes Mal aktualisieren, wenn Sie diese Komponente bereitstellen. Dadurch wird sichergestellt, dass Sie die richtige Lambda-Funktionsversion für die von Ihnen bereitgestellte Komponentenversion verwenden.

Weitere Informationen finden Sie unter Erstellen von Bereitstellungen.

- Die Firehose-Komponente wird für die Ausführung in einer VPC unterstützt. Um diese Komponente in einer VPC bereitzustellen, ist Folgendes erforderlich.
  - Die Firehose-Komponente muss über Konnektivität mit `firehose.region.amazonaws.com` dem VPC-Endpunkt verfügen`com.amazonaws.region.kinesis-firehose`.

Endpunkte und Ports

Diese Komponente muss in der Lage sein, ausgehende Anfragen an die folgenden Endpunkte und Ports auszuführen, zusätzlich zu den für den grundlegenden Betrieb erforderlichen Endpunkten und Ports. Weitere Informationen finden Sie unter Zulassen von Gerätedatenverkehr über einen Proxy oder eine Firewall.

Endpunkt	Port	Erforderlich	Beschreibung
`firehose. region.amazonaws.com`	443	Ja	Laden Sie Daten in Firehose hoch.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und alle ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitzustellen. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente und die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste Abhängigkeiten.

2.1.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.13.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.1.6

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.12.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.1.5

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.11.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.1.4

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.10.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.1.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.9.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.1.2

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.8.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.1.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.7.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Token-Exchange-Service	^2.0.0	Hart

2.0.8 - 2.1.0

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.0.8 und 2.1.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.6.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.0.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.5.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.0.6

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.4.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.0.5

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.3.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.0.4

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.2.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.0.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.3 <2.1.0	Hart
Lambda-Launcher	>=1.0.0	Hart
Lambda-Laufzeiten	>=1.0.0	Weich
Token-Exchange-Service	>=1.0.0	Hart

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zum Komponentenrezept .

Konfiguration

Diese Komponente bietet die folgenden Konfigurationsparameter, die Sie anpassen können, wenn Sie die Komponente bereitstellen.

> **Note**
>
> Die Standardkonfiguration dieser Komponente enthält Lambda-Funktionsparameter. Wir empfehlen Ihnen, nur die folgenden Parameter zu bearbeiten, um diese Komponente auf Ihren Geräten zu konfigurieren.

`lambdaParams`

Ein Objekt, das die Parameter für die Lambda-Funktion dieser Komponente enthält. Dieses Objekt enthält die folgenden Informationen:

`EnvironmentVariables`

Ein Objekt, das die Parameter der Lambda-Funktion enthält. Dieses Objekt enthält die folgenden Informationen:

`DEFAULT_DELIVERY_STREAM_ARN`

Der ARN des standardmäßigen Firehose-Bereitstellungs-Streams, an den die Komponente Daten sendet. Sie können den Ziel-Stream mit der -`delivery_stream_arn`Eigenschaft in der Nutzlast der Eingabenachricht überschreiben.

> **Note**
>
> Die Rolle des Core-Geräts muss die erforderlichen Aktionen für alle Zielbereitstellungsdatenströme zulassen. Weitere Informationen finden Sie unter Voraussetzungen.

`PUBLISH_INTERVAL`

(Optional) Die maximale Anzahl von Sekunden, die gewartet werden muss, bevor die Komponente Batch-Daten in Firehose veröffentlicht. Um die Komponente so zu konfigurieren, dass Metriken veröffentlicht werden, sobald sie sie empfangen, d. h. ohne Stapelverarbeitung, geben Sie an`0`.

Dieser Wert kann maximal 900 Sekunden betragen.

Standard: 10 Sekunden

`DELIVERY_STREAM_QUEUE_SIZE`

(Optional) Die maximale Anzahl von Datensätzen, die im Speicher aufbewahrt werden sollen, bevor die Komponente neue Datensätze für denselben Bereitstellungsdatenstrom ablehnt.

Dieser Wert muss mindestens 2 000 Datensätze umfassen.

Standard: 5 000 Datensätze

`containerMode`

(Optional) Der Containerisierungsmodus für diese Komponente. Wählen Sie aus den folgenden Optionen aus:

- `NoContainer` – Die Komponente wird nicht in einer isolierten Laufzeitumgebung ausgeführt.
- `GreengrassContainer` – Die Komponente wird in einer isolierten Laufzeitumgebung innerhalb des AWS IoT Greengrass Containers ausgeführt.

Standard: `GreengrassContainer`

`containerParams`

(Optional) Ein Objekt, das die Containerparameter für diese Komponente enthält. Die Komponente verwendet diese Parameter, wenn Sie `GreengrassContainer` für angeben`containerMode`.

Dieses Objekt enthält die folgenden Informationen:

`memorySize`

(Optional) Die Menge an Arbeitsspeicher (in Kilobyte), die der Komponente zugewiesen werden soll.

Der Standardwert ist 64 MB (65 535 KB).

`pubsubTopics`

(Optional) Ein Objekt, das die Themen enthält, in denen die Komponente den Empfang von Nachrichten abonniert. Sie können jedes Thema angeben und angeben, ob die Komponente MQTT-Themen von AWS IoT Core oder lokale Veröffentlichungs-/Abonnementthemen abonniert.

Dieses Objekt enthält die folgenden Informationen:

`0` – Dies ist ein Array-Index als Zeichenfolge.

Ein Objekt, das die folgenden Informationen enthält:

`type`

(Optional) Der Typ des Veröffentlichungs-/Abonnement-Messagings, den diese Komponente zum Abonnieren von Nachrichten verwendet. Wählen Sie aus den folgenden Optionen aus:

- `PUB_SUB` — Abonnieren Sie lokale Veröffentlichen/Abonnement-Nachrichten. Wenn Sie diese Option wählen, darf das Thema keine MQTT-Platzhalter enthalten. Weitere Informationen zum Senden von Nachrichten von einer benutzerdefinierten Komponente, wenn Sie diese Option angeben, finden Sie unter Lokale Nachrichten veröffentlichen/abonnieren.
- `IOT_CORE` – Abonnieren Sie AWS IoT Core MQTT-Nachrichten. Wenn Sie diese Option wählen, kann das Thema MQTT-Platzhalter enthalten. Weitere Informationen zum Senden von Nachrichten von benutzerdefinierten Komponenten, wenn Sie diese Option angeben, finden Sie unter MQTT-Nachrichten veröffentlichen/abonnieren AWS IoT Core.

Standard: PUB_SUB

`topic`

(Optional) Das Thema, das die Komponente abonniert, um Nachrichten zu empfangen. Wenn Sie `IotCore` für angeben`type`, können Sie in diesem Thema MQTT-Platzhalter (+ und #) verwenden.

Example Beispiel: Aktualisierung der Konfigurationszusammenführung (Container-Modus)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_DELIVERY_STREAM_ARN": "arn:aws:firehose:us-
west-2:123456789012:deliverystream/mystream"
    }
  },
  "containerMode": "GreengrassContainer"
}
```

Example Beispiel: Aktualisierung der Konfigurationszusammenführung (kein Containermodus)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_DELIVERY_STREAM_ARN": "arn:aws:firehose:us-
west-2:123456789012:deliverystream/mystream"
    }
  },
  "containerMode": "NoContainer"
}
```

Eingabedaten

Diese Komponente akzeptiert Stream-Inhalte zu den folgenden Themen und sendet den Inhalt an den Zielbereitstellungs-Stream. Die Komponente akzeptiert zwei Arten von Eingabedaten:

- JSON-Daten zum `kinesisfirehose/message`-Thema.
- Binäre Daten zum `kinesisfirehose/message/binary/#`-Thema.

Standardthema für JSON-Daten (lokales Veröffentlichen/Abonnieren): `kinesisfirehose/message`

Die Nachricht akzeptiert die folgenden Eigenschaften. Eingabenachrichten müssen im JSON-Format vorliegen.

`request`

Die zu sendenden Daten an den Lieferstrom und den Ziel-Lieferstrom, falls sie sich vom Standard-Stream unterscheiden.

Typ: `object`, der die folgenden Informationen enthält:

`data`

Die Daten, die an den Lieferstrom gesendet werden sollen.

Typ: `string`

`delivery_stream_arn`

(Optional) Der ARN des Ziel-Firehose-Bereitstellungsdatenstroms. Geben Sie diese Eigenschaft an, um den Standard-Bereitstellungs-Stream zu überschreiben.

Typ: `string`

`id`

Eine willkürliche ID für die Anforderung. Verwenden Sie diese Eigenschaft, um eine Eingabeanforderung einer Ausgabeantwort zuzuordnen. Wenn Sie diese Eigenschaft angeben, legt die Komponente die `id` Eigenschaft im Antwortobjekt auf diesen Wert fest.

Typ: `string`

Example Beispieleingabe

```
{
  "request": {
    "delivery_stream_arn": "arn:aws:firehose:region:account-id:deliverystream/
stream2-name",
    "data": "Data to send to the delivery stream."
  },
  "id": "request123"
}
```

Standardthema für Binärdaten (lokale Veröffentlichung/Abonnierung): `kinesisfirehose/message/binary/#`

Verwenden Sie dieses Thema, um eine Nachricht zu senden, die binäre Daten enthält. Die Komponente analysiert keine Binärdaten. Die Komponente streamt die Daten unverändert.

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, den abschließenden Schrägstrich () einzuschließen/.

Ausgabedaten

Diese Komponente veröffentlicht Antworten standardmäßig als Ausgabedaten zum folgenden MQTT-Thema. Sie müssen dieses Thema als `subject` in der Konfiguration für die Legacy-Abonnement-Routerkomponente angeben. Weitere Informationen zum Abonnieren von Nachrichten zu diesem Thema in Ihren benutzerdefinierten Komponenten finden Sie unter MQTT-Nachrichten veröffentlichen/abonnieren AWS IoT Core.

Standardthema (AWS IoT Core MQTT): `kinesisfirehose/message/status`

Example 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 die Komponente einen Fehler erkennt, der erneut versucht werden kann, z. B. einen Verbindungsfehler, wiederholt sie die Veröffentlichung im nächsten Batch.

Lokale Protokolldatei

Diese Komponente verwendet die folgende Protokolldatei.

```
/greengrass/v2/logs/aws.greengrass.KinesisFirehose.log
```

So zeigen Sie die Protokolle dieser Komponente an

- Führen Sie den folgenden Befehl auf dem Core-Gerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie durch */greengrass/v2* den Pfad zum AWS IoT Greengrass Stammordner.

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

Lizenzen

Diese Komponente 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

Diese Komponente wird gemäß dem Greengrass Core Software License Agreement veröffentlicht.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in jeder Version der Komponente beschrieben.

Version	Änderungen
2.1.7	Version für Greengrass-Kern Version 2.12.0 aktualisiert.
2.1.6	Version für Greengrass-Kern Version 2.11.0 aktualisiert.
2.1.5	Version für Greengrass-Kern Version 2.10.0 aktualisiert.
2.1.4	Version für Greengrass-Kern Version 2.9.0 aktualisiert.
2.1.3	Version für Greengrass-Kern Version 2.8.0 aktualisiert.
2.1.2	Version für Greengrass-Kern Version 2.7.0 aktualisiert.
2.1.1	Version für Greengrass-Kern Version 2.6.0 aktualisiert.
2.1.0	Neue Features • Fügt Unterstützung für HTTPS-Netzwerk-Proxy-Konfigurationen hinzu. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy und Ermöglichen Sie dem Kerngerät, einem HTTPS-Proxy zu vertrauen.
2.0.8	Version für Greengrass-Kern Version 2.5.0 aktualisiert.
2.0.7	Version für Greengrass-Kern Version 2.4.0 aktualisiert.
2.0.6	Version für Greengrass-Kern Version 2.3.0 aktualisiert.
2.0.5	Version für Greengrass-Kern Version 2.2.0 aktualisiert.

Version	Änderungen
2.0.4	Version für Greengrass-Kern Version 2.1.0 aktualisiert.
2.0.3	Erste Version

Weitere Informationen finden Sie auch unter

- Was ist Amazon Data Firehose? im Entwicklerhandbuch für Amazon Data Firehose

Lambda-Launcher

Die Lambda-Launcher-Komponente (`aws.greengrass.LambdaLauncher`) startet und stoppt AWS Lambda Funktionen auf AWS IoT Greengrass -Core-Geräten. Diese Komponente richtet auch jede Containerisierung ein und führt Prozesse als die von Ihnen angegebenen Benutzer aus.

> **Note**
>
> Wenn Sie eine Lambda-Funktionskomponente auf einem Core-Gerät bereitstellen, enthält die Bereitstellung auch diese Komponente. Weitere Informationen finden Sie unter Ausführen von -AWS LambdaFunktionen.

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.0.x

Typ

Diese Komponente ist eine generische Komponente (`aws.greengrass.generic`). Der Greengrass-Kernus führt die Lebenszyklusskripte der Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann nur auf Linux-Core-Geräten installiert werden.

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Ihr Core-Gerät muss die Anforderungen erfüllen, um Lambda-Funktionen auszuführen. Wenn Sie möchten, dass das Core-Gerät containerisierte Lambda-Funktionen ausführt, muss das Gerät die Voraussetzungen dafür erfüllen. Weitere Informationen finden Sie unter Anforderungen an die Lambda-Funktion.
- Die Lambda-Launcher-Komponente wird für die Ausführung in einer VPC unterstützt.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und alle ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitzustellen. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente und die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste Abhängigkeiten.

2.0.11 – 2.0.13

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.0.11 bis 2.0.13 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Lambda-Manager	>=2.0.0 <2.4.0	Hart

2.0.9 – 2.0.10

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.0.9 bis 2.0.10 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Lambda-Manager	>=2.0.0 <2.3.0	Hart

2.0.4 - 2.0.8

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.0.4 bis 2.0.8 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Lambda-Manager	>=2.0.0 <2.2.0	Hart

2.0.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Lambda-Manager	>=2.0.3 <2.1.0	Hart

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zum Komponentenrezept .

Konfiguration

Diese Komponente hat keine Konfigurationsparameter.

Lokale Protokolldatei

Diese Komponente verwendet die folgende Protokolldatei.

```
/greengrass/v2/logs/lambdaFunctionComponentName.log
```

So zeigen Sie die Protokolle dieser Komponente an

- Führen Sie den folgenden Befehl auf dem Core-Gerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* durch den Pfad zum AWS IoT Greengrass Stammordner und ersetzen Sie *lambdaFunctionComponentName* durch den Namen der Lambda-Funktionskomponente, die diese Komponente startet.

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

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in jeder Version der Komponente beschrieben.

Version	Änderungen
2.0.13	Fehlerbehebungen und Verbesserungen Allgemeine Fehlerbehebungen und Verbesserungen.
2.0.12	Fehlerbehebungen und Verbesserungen Behebt ein Problem, bei dem der Lambda-Launcher einen Fehler ausgeben konnte, wenn der vorherige Prozess nicht ordnungsgemäß gestoppt wurde.
2.0.11	Support für Lambda Manager 2.3.0.

Version	Änderungen
2.0.10	Fehlerbehebungen und Verbesserungen • Allgemeine Fehlerbehebungen und Verbesserungen.
2.0.9	Version für Greengrass-Kern Version 2.5.0 aktualisiert.
2.0.8	Version für Greengrass-Kern Version 2.4.0 aktualisiert.
2.0.7	Version für Greengrass-Kern Version 2.3.0 aktualisiert.
2.0.6	Allgemeine Leistungssteigerungen und Bugfixes.
2.0.4	Fehlerbehebungen und Verbesserungen • Behebt ein Problem, bei dem die Komponente nicht korrekt `AddGroupOwner` an den Lambda-Funktionscontainer übergeben wurde.
2.0.3	Erste Version

Lambda-Manager

Die Lambda-Manager-Komponente (`aws.greengrass.LambdaManager`) verwaltet Arbeitselemente und die Interprozesskommunikation für AWS Lambda Funktionen, die auf dem Greengrass-Core-Gerät ausgeführt werden.

> **Note**
>
> Wenn Sie eine Lambda-Funktionskomponente auf einem Kerngerät bereitstellen, umfasst die Bereitstellung auch diese Komponente. Weitere Informationen finden Sie unter Ausführen von -AWS LambdaFunktionen.

Themen

- Versionen
- Betriebssystem
- Typ
- Voraussetzungen

- Abhängigkeiten
- Konfiguration
- Lokale Protokolldatei
- Änderungsprotokoll

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Betriebssystem

Diese Komponente kann nur auf Linux-Core-Geräten installiert werden.

Typ

Diese Komponente ist eine Plugin-Komponente (`aws.greengrass.plugin`). Der Greengrass-Kern führt diese Komponente in derselben Java Virtual Machine (JVM) wie der Nucleus aus. Der Nucleus wird neu gestartet, wenn Sie die Version dieser Komponente auf dem Core-Gerät ändern.

Diese Komponente verwendet dieselbe Protokolldatei wie der Greengrass-Kern. Weitere Informationen finden Sie unter Überwachen von AWS IoT Greengrass Protokollen.

Weitere Informationen finden Sie unter Komponententypen.

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Ihr Kerngerät muss die Anforderungen für die Ausführung von Lambda-Funktionen erfüllen. Wenn Sie möchten, dass das Kerngerät containerisierte Lambda-Funktionen ausführt, muss das Gerät die entsprechenden Anforderungen erfüllen. Weitere Informationen finden Sie unter Anforderungen an die Lambda-Funktion.
- Die Lambda-Manager-Komponente wird für die Ausführung in einer VPC unterstützt.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt sie AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und all ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitstellen zu können. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente sowie die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste der Abhängigkeiten.

2.3.2 and 2.3.3

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.3.2 und 2.3.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.13.0	Weich

2.2.10 and 2.3.1

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.2.10 und 2.3.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.12.0	Weich

2.2.8 and 2.2.9

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.2.8 und 2.2.9 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.11.0	Weich

2.2.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.2.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.10.0	Weich

2.2.6

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.2.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.9.0	Weich

2.2.5

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.2.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.8.0	Weich

2.2.4

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.2.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.7.0	Weich

2.2.1 - 2.2.3

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.2.1 bis 2.2.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.6.0	Weich

2.2.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.2.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2,5,0 <2,6,0	Weich

2.1.3 and 2.1.4

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.1.3 und 2.1.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.5.0	Weich

2.1.2

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.4.0	Weich

2.1.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.3.0	Weich

2.1.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.2.0	Weich

2.0.x

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.x dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.3 <2.1.0	Weich

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zu den Komponentenrezepten.

Konfiguration

Diese Komponente stellt die folgenden Konfigurationsparameter bereit, die Sie bei der Bereitstellung der Komponente anpassen können.

`logHandlerMode`

> **Note**
>
> Nur für Lambda Manager-Versionen 2.3.0 und höher

Wird verwendet, um die zu verwendende Implementierung des Lambda-Log-Managers auszuwählen. Setzen Sie den Wert auf, `optimized` um weniger Threads zum Lesen von Lambda-Logs zu verwenden.

`getResultTimeoutInSecond`

(Optional) Die maximale Zeit in Sekunden, die Lambda-Funktionen ausgeführt werden können, bevor ihr Timeout eintritt.

Standard: `60`

Lokale Protokolldatei

Diese Komponente verwendet dieselbe Protokolldatei wie die Greengrass Nucleus-Komponente.

```
/greengrass/v2/logs/greengrass.log
```

Um die Protokolle dieser Komponente einzusehen

- Führen Sie den folgenden Befehl auf dem Kerngerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. */greengrass/v2*Ersetzen Sie durch den Pfad zum AWS IoT Greengrass Stammordner.

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

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
2.3.3	Fehlerkorrekturen und Verbesserungen • Allgemeine Fehlerbehebungen und Verbesserungen.
2.3.2	Die Version wurde für die Version 2.12.0 von Greengrass Nucleus aktualisiert.
2.3.1	Fehlerkorrekturen und Verbesserungen • Passt die Protokollebenen für bestimmte Fehler an.
2.3.0	Neue Features • Der Log-Handler wurde optimiert, um die CPU-Last zu reduzieren. Verwenden Sie diese Funktion, indem Sie die Konfigurationsoption `logHandlerMode` auf setzen`optimized` .

Version	Änderungen
	Fehlerkorrekturen und Verbesserungen • Protokolliert nicht mehr den vollständigen Stacktrace für`WorkQueue FullException` , was die Logs und die Leistung verbessert. • Legt das Lambda-Shutdown-Timeout von 15 Sekunden auf 300 Sekunden fest, um Timeouts beim Herunterfahren zu verhindern. • Behebt ein Problem, bei dem On-Demand-Lambdas nach einer Änderung der Konfiguration möglicherweise nicht neu gestartet werden können.
2.2.11	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem die LegacySubscriptionRouter Konfigura tion nicht aktualisiert wird, wenn sich die Lambda-Konfiguration ändert.
2.2.10	Die Version wurde für die Version 2.11.0 von Greengrass Nucleus aktualisiert.
2.2.9	Fehlerkorrekturen und Verbesserungen Behebt ein Problem, bei dem die Portnummer aufgrund einer schiefen Uhr beschädigt ist.
2.2.8	Die Version wurde für die Version 2.10.0 von Greengrass Nucleus aktualisiert.
2.2.7	Die Version wurde für die Version 2.9.0 von Greengrass Nucleus aktualisiert.
2.2.6	Die Version wurde für die Version 2.8.0 von Greengrass Nucleus aktualisiert.
2.2.5	Neue Features • Fügt Unterstützung für MQTT-Themen-Platzhalter in Ereignisquellen hinzu, in denen Sie lokale Veröffentlichungs-/Abonnementnachrichten abonnieren. Für diese Funktion ist Version 2.6.0 oder höher der Greengrass Nucleus-Komponente erforderlich. • Die Version wurde für die Version 2.7.0 von Greengrass Nucleus aktualisiert.

Version	Änderungen
2.2.4	Die Version wurde für die Version 2.6.0 von Greengrass Nucleus aktualisiert.
2.2.3	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem sich mehrere Instanzen einer Lambda-Funktion eine einzige Cgroup teilen. Diese Komponente verwendet Cgroups, um die Ressourcennutzung für Lambda-Funktionen zu verwalten.
2.2.2	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem angeheftete Lambda-Funktionsko mponenten in bestimmten Szenarien unerwartet neu gestartet werden.
2.2.1	Fehlerkorrekturen und Verbesserungen • Ändert die Versionsbeschränkungen der Greengrass Nucleus-A bhängigkeit dieser Komponente, um ein Problem mit der Abhängigk eitsauflösung zu beheben.
2.2.0	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem Lambda-Funktionen nach einem Neustart keine Protokolle schreiben konnten. • Behebt ein Problem, bei dem der ältere Abonnement-Router doppelte Nachrichten sendet, wenn das Thema Platzhalter enthält. • Behebt ein Problem, bei dem nicht angeheftete Lambda-Funktionen die Greengrass Interprocess Communication (IPC) -Bibliothek in der nicht verwenden konnten. AWS IoT Device SDK
2.1.4	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, das dazu führte, dass Lambda-Funktionen, die NodeJS-Laufzeiten verwenden, nur eine Nachricht verarbeiteten. • Die Version wurde für die Version 2.5.0 von Greengrass Nucleus aktualisiert.
2.1.3	Die Version wurde für die Version 2.4.0 von Greengrass Nucleus aktualisiert.
2.1.2	Die Version wurde für die Version 2.3.0 von Greengrass Nucleus aktualisiert.

Version	Änderungen
2.1.1	Die Version wurde für die Version 2.2.0 von Greengrass Nucleus aktualisiert.
2.1.0	Die Version wurde für die Version 2.1.0 von Greengrass Nucleus aktualisiert.
2.0.3	Erste Version

Lambda-Laufzeiten

Die Lambda-Laufzeitenkomponente (`aws.greengrass.LambdaRuntimes`) stellt die Laufzeiten bereit, die Greengrass-Core-Geräte zum Ausführen von AWS Lambda Funktionen verwenden.

> **Note**
>
> Wenn Sie eine Lambda-Funktionskomponente auf einem Core-Gerät bereitstellen, enthält die Bereitstellung auch diese Komponente. Weitere Informationen finden Sie unter Ausführen von -AWS LambdaFunktionen.

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.0.x

Typ

Diese Komponente ist eine generische Komponente (`aws.greengrass.generic`). Der Greengrass-Kern führt die Lebenszyklusskripte der Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann nur auf Linux-Core-Geräten installiert werden.

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Ihr Core-Gerät muss die Anforderungen für die Ausführung von Lambda-Funktionen erfüllen. Wenn Sie möchten, dass das Core-Gerät containerisierte Lambda-Funktionen ausführt, muss das Gerät die Anforderungen dafür erfüllen. Weitere Informationen finden Sie unter Anforderungen an die Lambda-Funktion.
- Die Lambda-Laufzeitenkomponente wird für die Ausführung in einer VPC unterstützt.

Abhängigkeiten

Diese Komponente hat keine Abhängigkeiten.

Konfiguration

Diese Komponente hat keine Konfigurationsparameter.

Lokale Protokolldatei

Diese Komponente gibt keine Protokolle aus.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in jeder Version der Komponente beschrieben.

Version	Änderungen
2.0.8	Version für Greengrass-Kern Version 2.5.0 aktualisiert.

Version	Änderungen
2.0.7	Version für Greengrass-Kern Version 2.4.0 aktualisiert.
2.0.6	Version für Greengrass-Kern Version 2.3.0 aktualisiert.
2.0.5	Version für Greengrass-Kern Version 2.2.0 aktualisiert.
2.0.4	Version für Greengrass-Kern Version 2.1.0 aktualisiert.
2.0.3	Erste Version

Legacy-Abonnement-Router

Der Legacy-Abonnement-Router (`aws.greengrass.LegacySubscriptionRouter`) verwaltet Abonnements auf dem Greengrass-Core-Gerät. Abonnements sind eine Funktion von AWS IoT Greengrass V1, die die Themen definiert, die Lambda-Funktionen für MQTT-Nachrichten auf einem Core-Gerät verwenden können. Weitere Informationen finden Sie unter Verwaltete Abonnements im MQTT-Messaging-Workflow im AWS IoT Greengrass V1-Entwicklerhandbuch.

Sie können diese Komponente verwenden, um Abonnements für Konnektorkomponenten und Lambda-Funktionskomponenten zu aktivieren, die das AWS IoT Greengrass Core SDK verwenden.

> **Note**
>
> Die Legacy-Abonnement-Routerkomponente ist nur erforderlich, wenn Ihre Lambda-Funktion die `publish()` Funktion im AWS IoT Greengrass Core-SDK verwendet. Wenn Sie Ihren Lambda-Funktionscode aktualisieren, um die Interprocess Communication (IPC)-Schnittstelle in der AWS IoT Device SDK V2 zu verwenden, müssen Sie die Legacy-Abonnement-Routerkomponente nicht bereitstellen. Weitere Informationen finden Sie unter den folgenden prozessübergreifenden Kommunikationsservices:
>
> - Lokale Nachrichten veröffentlichen/abonnieren
> - MQTT-Nachrichten veröffentlichen/abonnieren AWS IoT Core

Themen

- Versionen

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.1.x
- 2.0.x

Typ

Diese Komponente ist eine generische Komponente (`aws.greengrass.generic`). Der Greengrass-Kern führt die Lebenszyklusskripte der Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann nur auf Linux-Core-Geräten installiert werden.

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Der Legacy-Abonnement-Router wird für die Ausführung in einer VPC unterstützt.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und

alle ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitzustellen. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente und die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste Abhängigkeiten.

2.1.11

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.11 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.13.0	Weich

2.1.10

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.10 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.12.0	Weich

2.1.9

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.9 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.11.0	Weich

2.1.8

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.8 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.10.0	Weich

2.1.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.9.0	Weich

2.1.6

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.8.0	Weich

2.1.5

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.7.0	Weich

2.1.4

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.6.0	Weich

2.1.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.5.0	Weich

2.1.2

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.4.0	Weich

2.1.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.3.0	Weich

2.1.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.2.0	Weich

2.0.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.3 <2.1.0	Weich

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zum Komponentenrezept .

Konfiguration

Diese Komponente bietet die folgenden Konfigurationsparameter, die Sie anpassen können, wenn Sie die Komponente bereitstellen.

v2.1.x

`subscriptions`

(Optional) Die Abonnements, die auf dem Core-Gerät aktiviert werden sollen. Dies ist ein Objekt, bei dem jeder Schlüssel eine eindeutige ID ist und jeder Wert ein Objekt ist, das das Abonnement für diesen Konnektor definiert. Sie müssen ein Abonnement konfigurieren, wenn Sie eine V1-Konnektorkomponente oder eine Lambda-Funktion bereitstellen, die das AWS IoT Greengrass Core-SDK verwendet.

Jedes Abonnementobjekt enthält die folgenden Informationen:

`id`

Die eindeutige ID dieses Abonnements. Diese ID muss mit dem Schlüssel für dieses Abonnementobjekt übereinstimmen.

`source`

Die Lambda-Funktion, die das AWS IoT Greengrass Core SDK verwendet, um MQTT-Nachrichten zu den Themen zu veröffentlichen, die Sie in angeben`subject`. Geben Sie eines der folgenden Elemente an:

- Der Name einer Lambda-Funktionskomponente auf dem Core-Gerät. Geben Sie den Komponentennamen mit dem `component:` Präfix an, z. B. **`component:com.example.HelloWorldLambda`**.
- Der Amazon-Ressourcenname (ARN) einer Lambda-Funktion auf dem Core-Gerät.

> **Important**
>
> Wenn sich die Version der Lambda-Funktion ändert, müssen Sie das Abonnement mit der neuen Version der Funktion konfigurieren. Andernfalls leitet diese Komponente die Nachrichten erst weiter, wenn die Version mit dem Abonnement übereinstimmt.
> Sie müssen einen Amazon-Ressourcennamen (ARN) angeben, der die Version der zu importierenden Funktion enthält. Sie können keine Versions-Aliase wie `$LATEST` verwenden.

Um ein Abonnement für eine V1-Konnektor-Komponente bereitzustellen, geben Sie den Namen der Komponente oder den ARN der Lambda-Funktion der Konnektor-Komponente an.

`subject`

Das MQTT-Thema oder der Themenfilter, zu dem Quelle und Ziel Nachrichten veröffentlichen und empfangen können. Dieser Wert unterstützt die # Themen-Platzhalter + und .

`target`

Das Ziel, das die MQTT-Nachrichten zu den Themen empfängt, die Sie in angeben`subject`. Das Abonnement gibt an, dass die `source` Funktion MQTT-Nachrichten an AWS IoT Core oder an eine Lambda-Funktion auf dem Core-Gerät veröffentlicht. Geben Sie eines der folgenden Elemente an:

- `cloud`. Die `source` Funktion veröffentlicht MQTT-Nachrichten in AWS IoT Core.
- Der Name einer Lambda-Funktionskomponente auf dem Core-Gerät.
  Geben Sie den Komponentennamen mit dem `component:` Präfix an, z. B. **`component:com.example.HelloWorldLambda`**.
- Der Amazon-Ressourcenname (ARN) einer Lambda-Funktion auf dem Core-Gerät.

> **Important**
>
> Wenn sich die Version der Lambda-Funktion ändert, müssen Sie das Abonnement mit der neuen Version der Funktion konfigurieren. Andernfalls leitet diese Komponente die Nachrichten erst weiter, wenn die Version mit dem Abonnement übereinstimmt.

Sie müssen einen Amazon-Ressourcennamen (ARN) angeben, der die Version der zu importierenden Funktion enthält. Sie können keine Versions-Aliase wie `$LATEST` verwenden.

Standard: Keine Abonnements

Example Beispiel für ein Konfigurationsupdate (Definition eines Abonnements für AWS IoT Core)

Das folgende Beispiel gibt an, dass die `com.example.HelloWorldLambda` Lambda-Funktionskomponente MQTT-Nachrichten zu AWS IoT Core dem `hello/world` Thema in veröffentlicht.

```
{
  "subscriptions": {
    "Greengrass_HelloWorld_to_cloud": {
      "id": "Greengrass_HelloWorld_to_cloud",
      "source": "component:com.example.HelloWorldLambda",
      "subject": "hello/world",
      "target": "cloud"
    }
  }
}
```

Example Beispiel für ein Konfigurationsupdate (Definition eines Abonnements für eine andere Lambda-Funktion)

Das folgende Beispiel gibt an, dass die `com.example.HelloWorldLambda` Lambda-Funktionskomponente MQTT-Nachrichten an die `com.example.MessageRelay` Lambda-Funktionskomponente im `hello/world` Thema veröffentlicht.

```
{
  "subscriptions": {
    "Greengrass_HelloWorld_to_MessageRelay": {
      "id": "Greengrass_HelloWorld_to_MessageRelay",
      "source": "component:com.example.HelloWorldLambda",
      "subject": "hello/world",
      "target": "component:com.example.MessageRelay"
    }
  }
}
```

v2.0.x

`subscriptions`

(Optional) Die Abonnements, die auf dem Core-Gerät aktiviert werden sollen. Dies ist ein Objekt, bei dem jeder Schlüssel eine eindeutige ID ist und jeder Wert ein Objekt ist, das das Abonnement für diesen Konnektor definiert. Sie müssen ein Abonnement konfigurieren, wenn Sie eine V1-Konnektorkomponente oder eine Lambda-Funktion bereitstellen, die das AWS IoT Greengrass Core-SDK verwendet.

Jedes Abonnementobjekt enthält die folgenden Informationen:

`id`

Die eindeutige ID dieses Abonnements. Diese ID muss mit dem Schlüssel für dieses Abonnementobjekt übereinstimmen.

`source`

Die Lambda-Funktion, die das AWS IoT Greengrass Core SDK verwendet, um MQTT-Nachrichten zu den Themen zu veröffentlichen, die Sie in angeben`subject`. Machen Sie folgende Angaben:

- Der Amazon-Ressourcenname (ARN) einer Lambda-Funktion auf dem Core-Gerät.

> **Important**
>
> Wenn sich die Version der Lambda-Funktion ändert, müssen Sie das Abonnement mit der neuen Version der Funktion konfigurieren. Andernfalls leitet diese Komponente die Nachrichten erst weiter, wenn die Version mit dem Abonnement übereinstimmt.
> Sie müssen einen Amazon-Ressourcennamen (ARN) angeben, der die Version der zu importierenden Funktion enthält. Sie können keine Versions-Aliase wie `$LATEST` verwenden.

Um ein Abonnement für eine V1-Konnektor-Komponente bereitzustellen, geben Sie den ARN der Lambda-Funktion der Konnektor-Komponente an.

`subject`

Das MQTT-Thema oder der Themenfilter, zu dem Quelle und Ziel Nachrichten veröffentlichen und empfangen können. Dieser Wert unterstützt die # Themen-Platzhalter + und .

`target`

Das Ziel, das die MQTT-Nachrichten zu den Themen empfängt, die Sie in angeben`subject`. Das Abonnement gibt an, dass die `source` Funktion MQTT-Nachrichten an AWS IoT Core oder an eine Lambda-Funktion auf dem Core-Gerät veröffentlicht. Geben Sie eines der folgenden Elemente an:

- `cloud`. Die `source` Funktion veröffentlicht MQTT-Nachrichten in AWS IoT Core.
- Der Amazon-Ressourcenname (ARN) einer Lambda-Funktion auf dem Core-Gerät.

> **Important**
>
> Wenn sich die Version der Lambda-Funktion ändert, müssen Sie das Abonnement mit der neuen Version der Funktion konfigurieren. Andernfalls leitet diese Komponente die Nachrichten erst weiter, wenn die Version mit dem Abonnement übereinstimmt.
> Sie müssen einen Amazon-Ressourcennamen (ARN) angeben, der die Version der zu importierenden Funktion enthält. Sie können keine Versions-Aliase wie `$LATEST` verwenden.

Standard: Keine Abonnements

Example Beispiel für ein Konfigurationsupdate (Definition eines Abonnements für AWS IoT Core)

Das folgende Beispiel gibt an, dass die `Greengrass_HelloWorld` Funktion eine MQTT-Nachricht zu AWS IoT Core dem `hello/world` Thema in veröffentlicht.

```
"subscriptions": {
  "Greengrass_HelloWorld_to_cloud": {
    "id": "Greengrass_HelloWorld_to_cloud",
    "source": "arn:aws:lambda:us-
west-2:123456789012:function:Greengrass_HelloWorld:5",
    "subject": "hello/world",
    "target": "cloud"
```

```
  }
}
```

Example Beispiel für ein Konfigurationsupdate (Definition eines Abonnements für eine andere Lambda-Funktion)

Das folgende Beispiel gibt an, dass die `Greengrass_HelloWorld` Funktion MQTT-Nachrichten im `Greengrass_MessageRelay` zum `hello/world` Thema veröffentlicht.

```
"subscriptions": {
  "Greengrass_HelloWorld_to_MessageRelay": {
    "id": "Greengrass_HelloWorld_to_MessageRelay",
    "source": "arn:aws:lambda:us-
west-2:123456789012:function:Greengrass_HelloWorld:5",
    "subject": "hello/world",
    "target": "arn:aws:lambda:us-
west-2:123456789012:function:Greengrass_MessageRelay:5"
  }
}
```

Lokale Protokolldatei

Diese Komponente gibt keine Protokolle aus.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in jeder Version der Komponente beschrieben.

Version	Änderungen
2.1.11	Version für Greengrass-Kern Version 2.12.0 aktualisiert.
2.1.10	Version für Greengrass-Kern Version 2.11.0 aktualisiert.
2.1.9	Version für Greengrass-Kern Version 2.10.0 aktualisiert.
2.1.8	Version für Greengrass-Kern Version 2.9.0 aktualisiert.
2.1.7	Version für Greengrass-Kern Version 2.8.0 aktualisiert.
2.1.6	Version für Greengrass-Kern Version 2.7.0 aktualisiert.

Version	Änderungen
2.1.5	Version für Greengrass-Kern Version 2.6.0 aktualisiert.
2.1.4	Version für Greengrass-Kern Version 2.5.0 aktualisiert.
2.1.3	Version für Greengrass-Kern Version 2.4.0 aktualisiert.
2.1.2	Version für Greengrass-Kern Version 2.3.0 aktualisiert.
2.1.1	Version für Greengrass-Kern Version 2.2.0 aktualisiert.
2.1.0	Fehlerbehebungen und Verbesserungen • Fügt Unterstützung für die Angabe von Komponentennamen anstelle von ARNs für `source` und hinzu`target`. Wenn Sie einen Komponent ennamen für ein Abonnement angeben, müssen Sie das Abonnement nicht jedes Mal neu konfigurieren, wenn sich die Version der Lambda-Fu nktion ändert.
2.0.3	Erste Version

Lokale Debug-Konsole

Die lokale Debug-Konsolenkomponente (`aws.greengrass.LocalDebugConsole`) bietet ein lokales Dashboard, das Informationen über Ihre AWS IoT Greengrass Kerngeräte und deren Komponenten anzeigt. Sie können dieses Dashboard verwenden, um Ihr Kerngerät zu debuggen und lokale Komponenten zu verwalten.

> **Important**
>
> Wir empfehlen, diese Komponente nur in Entwicklungsumgebungen und nicht in Produktionsumgebungen zu verwenden. Diese Komponente bietet Zugriff auf Informationen und Operationen, die Sie in einer Produktionsumgebung normalerweise nicht benötigen. Folgen Sie dem Prinzip der geringsten Rechte, indem Sie diese Komponente nur dort einsetzen, wo Sie sie benötigen.

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Typ

Diese Komponente ist eine Plugin-Komponente (`aws.greengrass.plugin`). Der Greengrass-Kern führt diese Komponente in derselben Java Virtual Machine (JVM) wie der Nucleus aus. Der Nucleus wird neu gestartet, wenn Sie die Version dieser Komponente auf dem Core-Gerät ändern.

Diese Komponente verwendet dieselbe Protokolldatei wie der Greengrass-Kern. Weitere Informationen finden Sie unter Überwachen von AWS IoT Greengrass Protokollen.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf Kerngeräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Sie verwenden einen Benutzernamen und ein Passwort, um sich beim Dashboard anzumelden. Der Benutzername, der lautet`debug`, wird Ihnen zur Verfügung gestellt. Sie müssen die AWS IoT Greengrass CLI verwenden, um ein temporäres Passwort zu erstellen, das Sie beim Dashboard auf einem Core-Gerät authentifiziert. Sie müssen in der Lage sein, die AWS IoT Greengrass CLI zu verwenden, um die lokale Debug-Konsole zu verwenden. Weitere Informationen finden Sie in den Greengrass-CLI-Anforderungen. Weitere Informationen zum Generieren des Kennworts und zur Anmeldung finden Sie unter Verwendung der lokalen Debug-Konsolenkomponente.
- Die lokale Debug-Konsolenkomponente wird für die Ausführung in einer VPC unterstützt.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt sie AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und all ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitstellen zu können. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente sowie die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste der Abhängigkeiten.

2.4.1 – 2.4.2

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.4.1 bis 2.4.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.10.0 <2.13.0	Hart
Greengrass CLI	>=2.10.0 <2.13.0	Hart

2.4.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.4.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.10.0 <2.12.0	Hart
Greengrass CLI	>=2.10.0 <2.12.0	Hart

2.3.0 and 2.3.1

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.3.0 und 2.3.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.10.0 <2.12.0	Hart
Greengrass CLI	>=2.10.0 <2.12.0	Hart

2.2.9

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.2.9 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.1.0 <2.12.0	Hart
Greengrass CLI	>=2.1.0 <2.12.0	Hart

2.2.8

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.2.8 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.1.0 <2.11.0	Hart
Greengrass CLI	>=2.1.0 <2.11.0	Hart

2.2.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.2.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.1.0 <2.10.0	Hart
Greengrass CLI	>=2.1.0 <2.10.0	Hart

2.2.6

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.2.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.1.0 <2.9.0	Hart
Greengrass CLI	>=2.1.0 <2.9.0	Hart

2.2.5

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.2.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.1.0 <2.8.0	Hart
Greengrass CLI	>=2.1.0 <2.8.0	Hart

2.2.4

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.2.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.1.0 <2.7.0	Hart
Greengrass CLI	>=2.1.0 <2.7.0	Hart

2.2.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.2.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.1.0 <2.6.0	Hart
Greengrass CLI	>=2.1.0 <2.6.0	Hart

2.2.2

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.2.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.1.0 <2.5.0	Hart
Greengrass CLI	>=2.1.0 <2.5.0	Hart

2.2.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.2.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.1.0 <2.4.0	Hart

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Greengrass CLI	>=2.1.0 <2.4.0	Hart

2.2.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.2.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.1.0 <2.3.0	Hart
Greengrass CLI	>=2.1.0 <2.3.0	Hart

2.1.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.1.0 <2.2.0	Hart
Greengrass CLI	>=2.1.0 <2.2.0	Hart

2.0.x

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.x dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.3 <2.1.0	Weich
Greengrass CLI	>=2.0.3 <2.1.0	Weich

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zu den Komponentenrezepten.

Konfiguration

Diese Komponente stellt die folgenden Konfigurationsparameter bereit, die Sie bei der Bereitstellung der Komponente anpassen können.

v2.1.x - v2.4.x

`httpsEnabled`

(Optional) Sie können die HTTPS-Kommunikation für die lokale Debug-Konsole aktivieren. Wenn Sie die HTTPS-Kommunikation aktivieren, erstellt die lokale Debug-Konsole ein selbstsigniertes Zertifikat. Webbrowser zeigen Sicherheitswarnungen für Websites an, die selbstsignierte Zertifikate verwenden. Sie müssen das Zertifikat daher manuell überprüfen. Dann können Sie die Warnung umgehen. Weitere Informationen finden Sie unter Verwendung.

Standard: `true`

`port`

(Optional) Der Port, an dem die lokale Debug-Konsole bereitgestellt werden soll.

Standard: `1441`

`websocketPort`

(Optional) Der Websocket-Port, der für die lokale Debug-Konsole verwendet werden soll.

Standard: `1442`

`bindHostname`

(Optional) Der Hostname, der für die lokale Debug-Konsole verwendet werden soll.

Wenn Sie die AWS IoT Greengrass Core-Software in einem Docker-Container ausführen, setzen Sie diesen Parameter auf`0.0.0.0`, damit Sie die lokale Debug-Konsole außerhalb des Docker-Containers öffnen können.

Standard: `localhost`

Example Beispiel: Update zur Zusammenführung von Konfigurationen

Die folgende Beispielkonfiguration legt fest, dass die lokale Debug-Konsole an nicht standardmäßigen Ports geöffnet und HTTPS deaktiviert werden soll.

```
{
  "httpsEnabled": false,
  "port": "10441",
  "websocketPort": "10442"
}
```

v2.0.x

`port`

(Optional) Der Port, an dem die lokale Debug-Konsole bereitgestellt werden soll.

Standard: `1441`

`websocketPort`

(Optional) Der Websocket-Port, der für die lokale Debug-Konsole verwendet werden soll.

Standard: `1442`

`bindHostname`

(Optional) Der Hostname, der für die lokale Debug-Konsole verwendet werden soll.

Wenn Sie die AWS IoT Greengrass Core-Software in einem Docker-Container ausführen, setzen Sie diesen Parameter auf`0.0.0.0`, damit Sie die lokale Debug-Konsole außerhalb des Docker-Containers öffnen können.

Standard: `localhost`

Example Beispiel: Update zur Zusammenführung von Konfigurationen

Die folgende Beispielkonfiguration legt fest, dass die lokale Debug-Konsole auf nicht standardmäßigen Ports geöffnet werden soll.

```
{
  "port": "10441",
  "websocketPort": "10442"
}
```

Verwendung

Um die lokale Debug-Konsole zu verwenden, erstellen Sie eine Sitzung über die Greengrass-CLI. Wenn Sie eine Sitzung erstellen, stellt die Greengrass-CLI einen Benutzernamen und ein temporäres Passwort bereit, mit denen Sie sich bei der lokalen Debug-Konsole anmelden können.

Folgen Sie diesen Anweisungen, um die lokale Debug-Konsole auf Ihrem Core-Gerät oder auf Ihrem Entwicklungscomputer zu öffnen.

v2.1.x - v2.4.x

In den Versionen 2.1.0 und höher verwendet die lokale Debug-Konsole standardmäßig HTTPS. Wenn HTTPS aktiviert ist, erstellt die lokale Debug-Konsole ein selbstsigniertes Zertifikat, um die Verbindung zu sichern. Ihr Webbrowser zeigt aufgrund dieses selbstsignierten Zertifikats eine Sicherheitswarnung an, wenn Sie die lokale Debug-Konsole öffnen. Wenn Sie eine Sitzung mit der Greengrass-CLI erstellen, enthält die Ausgabe die Fingerabdrücke des Zertifikats, sodass Sie überprüfen können, ob das Zertifikat legitim und die Verbindung sicher ist.

Sie können HTTPS deaktivieren. Weitere Informationen finden Sie unter Konfiguration der lokalen Debug-Konsole.

Um die lokale Debug-Konsole zu öffnen

1. (Optional) Um die lokale Debug-Konsole auf Ihrem Entwicklungscomputer anzuzeigen, können Sie den Port der Konsole über SSH weiterleiten. Sie müssen die `AllowTcpForwarding` Option jedoch zuerst in der SSH-Konfigurationsdatei Ihres Kerngeräts aktivieren. Diese Option ist standardmäßig aktiviert. Führen Sie den folgenden Befehl auf Ihrem Entwicklungscomputer aus, um das Dashboard `localhost:1441` auf Ihrem Entwicklungscomputer anzuzeigen.

   ```
   ssh -L 1441:localhost:1441 -L 1442:localhost:1442 username@core-device-ip-address
   ```

   > **Note**
   >
   > Sie können die Standardports über `1441` und ändern`1442`. Weitere Informationen finden Sie unter Konfiguration der lokalen Debug-Konsole.

2. Erstellen Sie eine Sitzung, um die lokale Debug-Konsole zu verwenden. Wenn Sie eine Sitzung erstellen, generieren Sie ein Passwort, das Sie zur Authentifizierung verwenden. Für die lokale Debug-Konsole ist zur Erhöhung der Sicherheit ein Passwort erforderlich, da Sie diese Komponente verwenden können, um wichtige Informationen anzuzeigen und Operationen auf dem Kerngerät auszuführen. Die lokale Debug-Konsole erstellt außerdem ein Zertifikat, um die Verbindung zu sichern, wenn Sie HTTPS in der Komponentenkonfiguration aktivieren. HTTPS ist standardmäßig aktiviert.

   Verwenden Sie die AWS IoT Greengrass CLI, um die Sitzung zu erstellen. Dieser Befehl generiert ein zufälliges 43-stelliges Passwort, das nach 8 Stunden abläuft. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass V2 Stammordner.

   Linux or Unix

   ```
   sudo /greengrass/v2/bin/greengrass-cli get-debug-password
   ```

   Windows

   ```
   C:\greengrass\v2\bin\greengrass-cli get-debug-password
   ```

   Die Befehlsausgabe sieht wie das folgende Beispiel aus, wenn Sie die lokale Debug-Konsole für die Verwendung von HTTPS konfiguriert haben. Sie verwenden die Fingerabdrücke des Zertifikats, um zu überprüfen, ob die Verbindung sicher ist, wenn Sie die lokale Debug-Konsole öffnen.

   ```
   Username: debug
   Password: bEDp3MOHdj8ou2w5de_sCBI2XAaguy3a8XxREXAMPLE
   Password expires at: 2021-04-01T17:01:43.921999931-07:00
   The local debug console is configured to use TLS security. The certificate is
    self-signed so you will need to bypass your web browser's security warnings to
    open the console.
   Before you bypass the security warning, verify that the certificate fingerprint
    matches the following fingerprints.
   SHA-256: 15 0B 2C E2 54 8B 22 DE 08 46 54 8A B1 2B 25 DE FB 02 7D 01 4E 4A 56 67
    96 DA A6 CC B1 D2 C4 1B
   SHA-1: BC 3E 16 04 D3 80 70 DA E0 47 25 F9 90 FA D6 02 80 3E B5 C1
   ```

Die Debug-View-Komponente erstellt eine Sitzung, die 8 Stunden dauert. Danach müssen Sie ein neues Passwort generieren, um die lokale Debug-Konsole wieder aufrufen zu können.

3. Öffnen Sie das Dashboard und melden Sie sich dort an. Sehen Sie sich das Dashboard auf Ihrem Greengrass-Core-Gerät oder auf Ihrem Entwicklungscomputer an, wenn Sie den Port über SSH weiterleiten. Führen Sie eine der folgenden Aktionen aus:

   - Wenn Sie HTTPS in der lokalen Debug-Konsole aktiviert haben, was die Standardeinstellung ist, gehen Sie wie folgt vor:

     a. Öffnen Sie `https://localhost:1441` auf Ihrem Core-Gerät oder auf Ihrem Entwicklungscomputer, wenn Sie den Port über SSH weitergeleitet haben.

        Ihr Browser zeigt möglicherweise eine Sicherheitswarnung über ein ungültiges Sicherheitszertifikat an.

     b. Wenn Ihr Browser eine Sicherheitswarnung anzeigt, überprüfen Sie, ob das Zertifikat legitim ist, und umgehen Sie die Sicherheitswarnung. Gehen Sie wie folgt vor:

        i. Suchen Sie den SHA-256- oder SHA-1-Fingerabdruck für das Zertifikat und stellen Sie sicher, dass er mit dem SHA-256- oder SHA-1-Fingerabdruck übereinstimmt, den der Befehl zuvor gedruckt hat. `get-debug-password` Ihr Browser stellt möglicherweise einen oder beide Fingerabdrücke bereit. Schlagen Sie in der Dokumentation Ihres Browsers nach, um das Zertifikat und die zugehörigen Fingerabdrücke einzusehen. In einigen Browsern wird der Fingerabdruck des Zertifikats als Fingerabdruck bezeichnet.

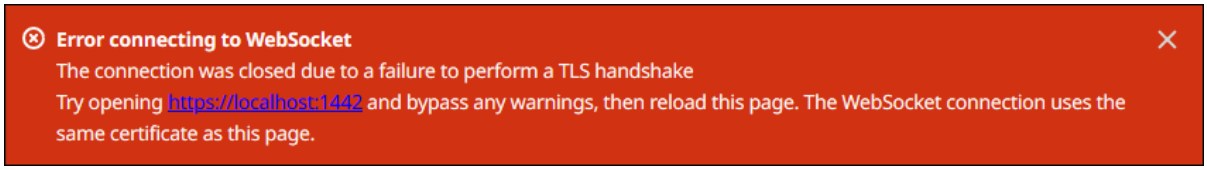
        > **Note**
        >
        > Wenn der Fingerabdruck des Zertifikats nicht übereinstimmt, gehen Sie Step 2 zu Eine neue Sitzung erstellen. Wenn der Fingerabdruck des Zertifikats immer noch nicht übereinstimmt, ist Ihre Verbindung möglicherweise unsicher.

        ii. Wenn der Fingerabdruck des Zertifikats übereinstimmt, umgehen Sie die Sicherheitswarnung Ihres Browsers, um die lokale Debug-Konsole zu öffnen. Schlagen Sie in der Dokumentation Ihres Browsers nach, um die Sicherheitswarnung des Browsers zu umgehen.

c. Melden Sie sich auf der Website mit dem Benutzernamen und dem Passwort an, die der `get-debug-password` Befehl zuvor gedruckt hat.

   Die lokale Debug-Konsole wird geöffnet.

d. Wenn die lokale Debug-Konsole einen Fehler anzeigt, der besagt, dass WebSocket aufgrund eines fehlgeschlagenen TLS-Handshakes keine Verbindung zur hergestellt werden kann, müssen Sie die selbstsignierte Sicherheitswarnung für die URL umgehen. WebSocket

   Gehen Sie wie folgt vor:

   i. Öffnen Sie `https://localhost:1442` in demselben Browser, in dem Sie die lokale Debug-Konsole geöffnet haben.

   ii. Überprüfen Sie das Zertifikat und umgehen Sie die Sicherheitswarnung.

   Ihr Browser zeigt möglicherweise eine HTTP 404-Seite an, nachdem Sie die Warnung umgangen haben.

   iii. `https://localhost:1441`Erneut öffnen.

   Die lokale Debug-Konsole zeigt Informationen über das Kerngerät an.

- Wenn Sie HTTPS in der lokalen Debug-Konsole deaktiviert haben, gehen Sie wie folgt vor:

  a. Öffnen Sie es `http://localhost:1441` auf Ihrem Core-Gerät oder öffnen Sie es auf Ihrem Entwicklungscomputer, wenn Sie den Port über SSH weitergeleitet haben.

  b. Melden Sie sich auf der Website mit dem Benutzernamen und dem Passwort an, die der `get-debug-password` Befehl zuvor gedruckt hat.

  Die lokale Debug-Konsole wird geöffnet.

v2.0.x

Um die lokale Debug-Konsole zu öffnen

1. (Optional) Um die lokale Debug-Konsole auf Ihrem Entwicklungscomputer anzuzeigen, können Sie den Port der Konsole über SSH weiterleiten. Sie müssen die `AllowTcpForwarding` Option jedoch zuerst in der SSH-Konfigurationsdatei Ihres Kerngeräts aktivieren. Diese Option ist standardmäßig aktiviert. Führen Sie den folgenden Befehl auf Ihrem Entwicklungscomputer aus, um das Dashboard `localhost:1441` auf Ihrem Entwicklungscomputer anzuzeigen.

   ```
   ssh -L 1441:localhost:1441 -L 1442:localhost:1442 username@core-device-ip-address
   ```

   > **Note**
   >
   > Sie können die Standardports über `1441` und ändern`1442`. Weitere Informationen finden Sie unter Konfiguration der lokalen Debug-Konsole.

2. Erstellen Sie eine Sitzung, um die lokale Debug-Konsole zu verwenden. Wenn Sie eine Sitzung erstellen, generieren Sie ein Passwort, das Sie zur Authentifizierung verwenden. Für die lokale Debug-Konsole ist zur Erhöhung der Sicherheit ein Passwort erforderlich, da Sie diese Komponente verwenden können, um wichtige Informationen anzuzeigen und Operationen auf dem Kerngerät auszuführen.

   Verwenden Sie die AWS IoT Greengrass CLI, um die Sitzung zu erstellen. Dieser Befehl generiert ein zufälliges 43-stelliges Passwort, das nach 8 Stunden abläuft. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass V2 Stammordner.

   Linux or Unix

   ```
   sudo /greengrass/v2/bin/greengrass-cli get-debug-password
   ```

   Windows

   ```
   C:\greengrass\v2\bin\greengrass-cli get-debug-password
   ```

Die Befehlsausgabe sieht wie das folgende Beispiel aus.

```
Username: debug
Password: bEDp3MOHdj8ou2w5de_sCBI2XAaguy3a8XxREXAMPLE
Password will expire at: 2021-04-01T17:01:43.921999931-07:00
```

Die Debug-View-Komponente erstellt eine Sitzung, die 4 Stunden dauert. Anschließend müssen Sie ein neues Passwort generieren, um die lokale Debug-Konsole erneut aufrufen zu können.

3. Öffnen Sie es `http://localhost:1441` auf Ihrem Hauptgerät oder öffnen Sie es auf Ihrem Entwicklungscomputer, wenn Sie den Port über SSH weitergeleitet haben.
4. Melden Sie sich auf der Website mit dem Benutzernamen und dem Passwort an, die der `get-debug-password` Befehl zuvor gedruckt hat.

   Die lokale Debug-Konsole wird geöffnet.

Lokale Protokolldatei

Diese Komponente verwendet dieselbe Protokolldatei wie die Greengrass Nucleus-Komponente.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

Um die Protokolle dieser Komponente einzusehen

- Führen Sie den folgenden Befehl auf dem Kerngerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

Linux

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

Windows (PowerShell)

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

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
2.4.2	Fehlerkorrekturen und Verbesserungen • Allgemeine Fehlerbehebungen und Verbesserungen.
2.4.1	Die Version wurde für die Version 2.12.0 von Greengrass Nucleus aktualisiert.
2.4.0	Neue Features • Fügt die Stream Manager-Debugging-Konsole hinzu.
2.3.1	Die Version wurde für die Version 2.11.0 von Greengrass Nucleus aktualisiert.
2.3.0	Die Version wurde für die Version 2.10.0 von Greengrass Nucleus aktualisiert. Neue Features • Beinhaltet einen PubSub AWS IoT Core MQTT-Debug-Client.
2.2.7	Die Version wurde für die Version 2.9.0 von Greengrass Nucleus aktualisiert.
2.2.6	Die Version wurde für die Version 2.8.0 von Greengrass Nucleus aktualisiert.
2.2.5	Die Version wurde für die Version 2.7.0 von Greengrass Nucleus aktualisiert.
2.2.4	Die Version wurde für die Version 2.6.0 von Greengrass Nucleus aktualisiert.

Version	Änderungen
2.2.3	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, das den Start verhinderte, wenn die Komponente den Keystore, der den privaten SSL-Schlüssel enthält, nicht entschlüsseln konnte. • Die Version wurde für die Version 2.5.0 von Greengrass Nucleus aktualisiert.
2.2.2	Die Version wurde für die Version 2.4.0 von Greengrass Nucleus aktualisiert.
2.2.1	Die Version wurde für die Version 2.3.0 von Greengrass Nucleus aktualisiert.
2.2.0	Die Version wurde für die Version 2.2.0 von Greengrass Nucleus aktualisiert.
2.1.0	Neue Features • Verwendet HTTPS, um Ihre Verbindung zur lokalen Debug-Konsole zu sichern. HTTPS ist standardmäßig aktiviert. Fehlerkorrekturen und Verbesserungen • Sie können Flashbar-Meldungen im Konfigurationseditor verwerfen.
2.0.3	Erste Version

Protokollmanager

Die Log Manager-Komponente (`aws.greengrass.LogManager`) lädt Protokolle von AWS IoT Greengrass Core-Geräten auf Amazon CloudWatch Logs hoch. Sie können Protokolle aus dem Greengrass-Kern, anderen Greengrass-Komponenten und anderen Anwendungen und Services hochladen, die keine Greengrass-Komponenten sind. Weitere Informationen zur Überwachung von Protokollen in - CloudWatch Protokollen und im lokalen Dateisystem finden Sie unter Überwachen von AWS IoT Greengrass Protokollen.

Die folgenden Überlegungen gelten, wenn Sie die Log Manager-Komponente verwenden, um in CloudWatch Protokolle zu schreiben:

- Protokollverzögerungen

> **Note**
>
> Wir empfehlen Ihnen, auf Log Manager Version 2.3.0 zu aktualisieren, um Protokollverzögerungen für rotierte und aktive Protokolldateien zu reduzieren. Wenn Sie auf Log Manager 2.3.0 aktualisieren, empfehlen wir Ihnen, auch auf Greengrass-Kern 2.9.1 zu aktualisieren.

Die Log Manager-Komponente Version 2.2.8 (und früher) verarbeitet und lädt Protokolle nur aus rotierten Protokolldateien hoch. Standardmäßig rotiert die AWS IoT Greengrass Core-Software Protokolldateien stündlich oder nach 1 024 KB. Daher lädt die Log Manager-Komponente Protokolle erst hoch, nachdem die -AWS IoT GreengrassCore-Software oder eine Greengrass-Komponente Protokolle mit mehr als 1.024 KB geschrieben hat. Sie können eine niedrigere Größenbeschränkung für Protokolldateien konfigurieren, damit Protokolldateien häufiger rotiert werden. Dies führt dazu, dass die Protokollmanagerkomponente Protokolle häufiger in CloudWatch Protokolle hochlädt.

Die Log Manager-Komponente Version 2.3.0 (und höher) verarbeitet und lädt alle Protokolle hoch. Wenn Sie ein neues Protokoll schreiben, verarbeitet und lädt Log Manager Version 2.3.0 (und höher) diese aktive Protokolldatei direkt hoch, anstatt darauf zu warten, dass sie rotiert wird. Das bedeutet, dass Sie das neue Protokoll in 5 Minuten oder weniger anzeigen können.

Die Log Manager-Komponente lädt regelmäßig neue Protokolle hoch. Standardmäßig lädt die Log Manager-Komponente alle 5 Minuten neue Protokolle hoch. Sie können ein niedrigeres Upload-Intervall konfigurieren, sodass die Protokollmanagerkomponente Protokolle häufiger in CloudWatch Protokolle hochlädt, indem Sie die konfigurieren`periodicUploadIntervalSec`. Weitere Informationen zur Konfiguration dieses regelmäßigen Intervalls finden Sie unter Konfiguration .

Protokolle können nahezu in Echtzeit aus demselben Greengrass-Dateisystem hochgeladen werden. Wenn Sie Protokolle in Echtzeit beobachten müssen, sollten Sie die Verwendung von Dateisystemprotokollen in Betracht ziehen.

> **Note**
>
> Wenn Sie verschiedene Dateisysteme verwenden, um Protokolle zu schreiben, kehrt Log Manager zum Verhalten in den Versionen 2.2.8 und früher des Protokollmanagers

zurück. Informationen zum Zugriff auf Dateisystemprotokolle finden Sie unter Zugriff auf Dateisystemprotokolle.

- Zeitverzerrung

  Die Log-Manager-Komponente verwendet den Standard-Signaturprozess von Signature Version 4, um API-Anfragen an CloudWatch Logs zu erstellen. Wenn die Systemzeit auf einem Core-Gerät um mehr als 15 Minuten nicht synchron ist, lehnt CloudWatch Logs die Anforderungen ab. Weitere Informationen finden Sie unter Signaturprozess mit Signaturversion 4 in derAllgemeine AWS-Referenz .

Informationen zu den Protokollgruppen und Protokollstreams, in die diese Komponente Protokolle hochlädt, finden Sie unter Verwendung.

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Typ

Diese Komponente ist eine Plugin-Komponente (`aws.greengrass.plugin`). Der Greengrass-Kern führt diese Komponente in derselben Java Virtual Machine (JVM) wie der Kern aus. Der Kern wird neu gestartet, wenn Sie die Version dieser Komponente auf dem Core-Gerät ändern.

Diese Komponente verwendet dieselbe Protokolldatei wie der Greengrass-Kern. Weitere Informationen finden Sie unter Überwachen von AWS IoT Greengrass Protokollen.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf -Core-Geräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Die Greengrass-Geräterolle muss die `logs:DescribeLogStreams` Aktionen `logs:CreateLogGroup`, `logs:CreateLogStreamlogs:PutLogEvents`, und zulassen, wie in der folgenden Beispiel-IAM-Richtlinie gezeigt.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:logs:*:*:*"
    }
```

```
  ]
}
```

> **Note**
>
> Die Greengrass-Geräterolle, die Sie bei der Installation der AWS IoT Greengrass Core-Software erstellen, enthält standardmäßig die Berechtigungen in dieser Beispielrichtlinie.

Weitere Informationen finden Sie unter Verwenden von identitätsbasierten Richtlinien (IAM-Richtlinien) für - CloudWatch Protokolle im Amazon- CloudWatch Logs-Benutzerhandbuch.

- Die Log Manager-Komponente wird für die Ausführung in einer VPC unterstützt. Um diese Komponente in einer VPC bereitzustellen, ist Folgendes erforderlich.
  - Die Log-Manager-Komponente muss über Konnektivität mit `logs.region.amazonaws.com` dem VPC-Endpunkt verfügen`com.amazonaws.us-east-1.logs`.

Endpunkte und Ports

Diese Komponente muss in der Lage sein, ausgehende Anfragen an die folgenden Endpunkte und Ports auszuführen, zusätzlich zu den für den grundlegenden Betrieb erforderlichen Endpunkten und Ports. Weitere Informationen finden Sie unter Zulassen von Gerätedatenverkehr über einen Proxy oder eine Firewall.

Endpunkt	Port	Erforderlich	Beschreibung
`logs.region.amazonaws.com`	443	Nein	Erforderlich, wenn Sie Protokolle in CloudWatch - Protokolle schreiben.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und alle ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitzustellen. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente und die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste Abhängigkeiten.

2.3.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.3.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.1.0 <2.13.0	Weich

2.3.5 and 2.3.6

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.3.5 und 2.3.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.1.0 <2.12.0	Weich

2.3.3 – 2.3.4

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.3.3 bis 2.3.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.1.0 <2.11.0	Weich

2.2.8 – 2.3.2

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.2.8 bis 2.3.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.1.0 <2.10.0	Weich

2.2.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.2.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.1.0 <2.9.0	Weich

2.2.6

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.2.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.1.0 <2.8.0	Weich

2.2.5

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.2.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.1.0 <2.7.0	Weich

2.2.1 - 2.2.4

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.2.1–2.2.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.1.0 <2.6.0	Weich

2.1.3 and 2.2.0

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.1.3 und 2.2.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.1.0 <2.5.0	Weich

2.1.2

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.1.0 <2.4.0	Weich

2.1.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.1.0 <2.3.0	Weich

2.1.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.1.0 <2.2.0	Weich

2.0.x

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.x dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.3 <2.1.0	Weich

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zum Komponentenrezept .

Konfiguration

Diese Komponente bietet die folgenden Konfigurationsparameter, die Sie anpassen können, wenn Sie die Komponente bereitstellen.

v2.3.6 – v2.3.7

`logsUploaderConfiguration`

(Optional) Die Konfiguration für Protokolle, die die Log-Manager-Komponente hochlädt. Dieses Objekt enthält die folgenden Informationen:

`systemLogsConfiguration`

(Optional) Die Konfiguration für -AWS IoT GreengrassCore-Softwaresystemprotokolle, die Protokolle aus dem Greengrass-Kern und Plugin-Komponenten enthalten. Geben Sie diese Konfiguration an, damit die Log Manager-Komponente Systemprotokolle verwalten kann. Dieses Objekt enthält die folgenden Informationen:

`uploadToCloudWatch`

(Optional) Sie können Systemprotokolle in CloudWatch -Protokolle hochladen.

Standard: `false`

`minimumLogLevel`

(Optional) Die Mindestebene der hochzuladenden Protokollnachrichten. Diese Mindestebene gilt nur, wenn Sie die Greengrass-Kernkomponente für die Ausgabe von Protokollen im JSON-Format konfigurieren. Um Protokolle im JSON-

Format zu aktivieren, geben Sie `JSON` für den Protokollierungsformatparameter (`logging.format`) an.

Wählen Sie aus den folgenden Protokollebenen aus, die hier in der Reihenfolge der Ebenen aufgeführt sind:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Standard: `INFO`

`diskSpaceLimit`

(Optional) Die maximale Gesamtgröße der Greengrass-Systemprotokolldateien in der Einheit, die Sie in angeben`diskSpaceLimitUnit`. Nachdem die Gesamtgröße der Greengrass-Systemprotokolldateien diese maximale Gesamtgröße überschreitet, löscht die AWS IoT Greengrass Core-Software die ältesten Greengrass-Systemprotokolldateien.

Dieser Parameter entspricht dem Parameter für die Protokollgrößenbeschränkung (`totalLogsSize`KB) der Greengrass-Kernkomponente . Die AWS IoT Greengrass Core-Software verwendet das Minimum der beiden Werte als maximale Gesamtgröße des Greengrass-Systemprotokolls.

`diskSpaceLimitUnit`

(Optional) Die Einheit für die `diskSpaceLimit`. Wählen Sie aus den folgenden Optionen aus:

- `KB` – Kilobyte
- `MB` – Megabyte
- `GB` – Gigabyte

Standard: `KB`

`deleteLogFileAfterCloudUpload`

(Optional) Sie können eine Protokolldatei löschen, nachdem die Protokollmanagerkomponente die Protokolle in CloudWatch Logs hochgeladen hat.

Standard: `false`

`componentLogsConfigurationMap`

(Optional) Eine Zuordnung von Protokollkonfigurationen für Komponenten auf dem Core-Gerät. Jedes `componentName` Objekt in dieser Zuordnung definiert die Protokollkonfiguration für die Komponente oder Anwendung. Die Log-Manager-Komponente lädt diese Komponentenprotokolle in CloudWatch Logs hoch.

> **Important**
>
> Wir empfehlen dringend, einen einzelnen Konfigurationsschlüssel pro Komponente zu verwenden. Sie sollten nur auf eine Gruppe von Dateien abzielen, die nur eine Protokolldatei haben, in die aktiv geschrieben wird, wenn Sie die verwenden`logFileRegex`. Wenn Sie diese Empfehlung nicht befolgen, können doppelte Protokolle in hochgeladen werden CloudWatch. Wenn Sie mehrere aktive Protokolldateien mit einem einzigen Regex anvisieren, empfehlen wir Ihnen, auf Log Manager v2.3.1 oder höher zu aktualisieren und Ihre Konfiguration mithilfe der Beispielkonfiguration zu ändern.

> **Note**
>
> Wenn Sie von einer Version des Protokollmanagers vor v2.2.0 aktualisieren, können Sie weiterhin die `componentLogsConfiguration` Liste anstelle von verwenden`componentLogsConfigurationMap`. Es wird jedoch dringend empfohlen, das Kartenformat zu verwenden, damit Sie Zusammenführungs- und Reset-Aktualisierungen verwenden können, um Konfigurationen für bestimmte Komponenten zu ändern. Informationen zum -`componentLogsConfiguration`Parameter finden Sie in den Konfigurationsparametern für v2.1.x dieser Komponente.

`componentName`

Die Protokollkonfiguration für die *`componentName`* Komponente oder Anwendung für diese Protokollkonfiguration. Sie können den Namen einer Greengrass-Komponente oder einen anderen Wert angeben, um diese Protokollgruppe zu identifizieren.

Jedes Objekt enthält die folgenden Informationen:

`minimumLogLevel`

(Optional) Die Mindestebene der hochzuladenden Protokollnachrichten. Diese Mindestebene gilt nur, wenn die Protokolle dieser Komponente ein bestimmtes JSON-Format verwenden, das Sie im AWS IoT Greengrass Protokollierungsmodul-Repository auf finden GitHub.

Wählen Sie aus den folgenden Protokollebenen aus, die hier in der Reihenfolge der Ebenen aufgeführt sind:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Standard: `INFO`

`diskSpaceLimit`

(Optional) Die maximale Gesamtgröße aller Protokolldateien für diese Komponente in der Einheit, die Sie in angeben`diskSpaceLimitUnit`. Nachdem die Gesamtgröße der Protokolldateien dieser Komponente diese maximale Gesamtgröße überschreitet, löscht die AWS IoT Greengrass Core-Software die ältesten Protokolldateien dieser Komponente.

Dieser Parameter bezieht sich auf den Parameter zur Begrenzung der Protokollgröße (`totalLogsSizeKB`) der Greengrass-Kernkomponente . Die AWS IoT Greengrass -Core-Software verwendet das Minimum der beiden Werte als maximale Gesamtprotokollgröße für diese Komponente.

`diskSpaceLimitUnit`

(Optional) Die Einheit für die `diskSpaceLimit`. Wählen Sie aus den folgenden Optionen aus:

- `KB` – Kilobyte
- `MB` – Megabyte
- `GB` – Gigabyte

Standard: `KB`

`logFileDirectoryPath`

(Optional) Der Pfad zu dem Ordner, der die Protokolldateien dieser Komponente enthält.

Sie müssen diesen Parameter nicht für Greengrass-Komponenten angeben, die in der Standardausgabe (stdout) und im Standardfehler (stderr) gedruckt werden.

Standard: *`/greengrass/v2`*`/logs`.

`logFileRegex`

(Optional) Ein regulärer Ausdruck, der das Format des Protokolldateinamens angibt, das die Komponente oder Anwendung verwendet. Die Log Manager-Komponente verwendet diesen regulären Ausdruck, um Protokolldateien im Ordner unter zu identifizieren`logFileDirectoryPath`.

Sie müssen diesen Parameter nicht für Greengrass-Komponenten angeben, die in der Standardausgabe (stdout) und im Standardfehler (stderr) gedruckt werden.

Wenn Ihre Komponente oder Anwendung Protokolldateien rotiert, geben Sie einen Regex an, der den rotierten Protokolldateinamen entspricht. Sie können beispielsweise angeben, **`hello_world\\\\w*.log`** um Protokolle für eine Hello World-Anwendung hochzuladen. Das `\\\\w*` Muster entspricht null oder mehr Wortzeichen, die alphanumerische Zeichen und Unterstriche enthalten. Dieser Regex gleicht Protokolldateien mit und ohne Zeitstempel im Namen ab. In diesem Beispiel lädt der Protokollmanager die folgenden Protokolldateien hoch:

- `hello_world.log` – Die neueste Protokolldatei für die Hello World-Anwendung.
- `hello_world_2020_12_15_17_0.log` – Eine ältere Protokolldatei für die Hello World-Anwendung.

Standard: *`componentName`*`\\\\w*.log`, wobei *`componentName`* der Name der Komponente für diese Protokollkonfiguration ist.

`deleteLogFileAfterCloudUpload`

(Optional) Sie können eine Protokolldatei löschen, nachdem die Protokollmanagerkomponente die Protokolle in CloudWatch Logs hochgeladen hat.

Standard: `false`

`multiLineStartPattern`

(Optional) Ein regulärer Ausdruck, der angibt, wann eine Protokollnachricht in einer neuen Zeile eine neue Protokollnachricht ist. Wenn der reguläre Ausdruck nicht mit der neuen Zeile übereinstimmt, hängt die Protokollmanagerkomponente die neue Zeile an die Protokollnachricht für die vorherige Zeile an.

Standardmäßig prüft die Protokollmanagerkomponente, ob die Zeile mit einem Leerzeichen beginnt, z. B. einer Registerkarte oder einem Leerzeichen. Ist dies nicht der Fall, behandelt der Protokollmanager diese Zeile als neue Protokollnachricht. Andernfalls wird diese Zeile an die aktuelle Protokollnachricht angehängt. Dieses Verhalten stellt sicher, dass die Protokollmanagerkomponente keine Nachrichten aufteilt, die sich über mehrere Zeilen erstrecken, z. B. Stack-Ablaufverfolgungen.

`periodicUploadIntervalSec`

(Optional) Der Zeitraum in Sekunden, in dem die Log Manager-Komponente nach neuen hochzuladenden Protokolldateien sucht.

Standard: `300` (5 Minuten)

Minimum: `0.000001` (1 Mikrosekunde)

`deprecatedVersionSupport`

Gibt an, ob der Protokollmanager Verbesserungen der Protokollierungsgeschwindigkeit verwenden soll, die in Log Manager v2.3.5 eingeführt wurden. Setzen Sie den Wert auf `false`, um die Verbesserungen zu verwenden.

Wenn Sie diesen Wert auf festlegen`false`, wenn Sie ein Upgrade von Log Manager v2.3.1 durchführen, können frühere doppelte Protokolleinträge hochgeladen werden.

Der Standardwert ist `true`.

Example Beispiel: Aktualisierung der Konfigurationszusammenführung

Die folgende Beispielkonfiguration gibt an, Systemprotokolle und `com.example.HelloWorld` Komponentenprotokolle in CloudWatch Protokolle hochzuladen.

```
{
  "logsUploaderConfiguration": {
```

```
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfigurationMap": {
      "com.example.HelloWorld": {
        "minimumLogLevel": "INFO",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  },
  "periodicUploadIntervalSec": "300",
  "deprecatedVersionSupport": "false"
}
```

Example Beispiel: Konfiguration zum Hochladen mehrerer aktiver Protokolldateien mit Log Manager v2.3.1

Die folgende Beispielkonfiguration ist das empfohlene Beispiel, wenn Sie mehrere aktive Protokolldateien anvisieren möchten. Diese Beispielkonfiguration gibt an, welche aktiven Protokolldateien Sie in hochladen möchten CloudWatch. Bei Verwendung dieser Konfigurationsbeispielkonfiguration werden auch alle rotierten Dateien hochgeladen, die mit dem übereinstimmen`logFileRegex`. Diese Beispielkonfiguration wird auf Log Manager v2.3.1 unterstützt.

```
{
  "logsUploaderConfiguration": {
    "componentLogsConfigurationMap": {
      "com.example.A": {
        "logFileRegex": "com.example.A\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
      "com.example.B": {
        "logFileRegex": "com.example.B\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
```

```
  },
  "periodicUploadIntervalSec": "10"
}
```

v2.3.x

`logsUploaderConfiguration`

(Optional) Die Konfiguration für Protokolle, die die Log-Manager-Komponente hochlädt. Dieses Objekt enthält die folgenden Informationen:

`systemLogsConfiguration`

(Optional) Die Konfiguration für -AWS IoT GreengrassCore-Softwaresystemprotokolle, die Protokolle aus dem Greengrass-Kern und den Plugin-Komponenten enthalten. Geben Sie diese Konfiguration an, damit die Log Manager-Komponente Systemprotokolle verwalten kann. Dieses Objekt enthält die folgenden Informationen:

`uploadToCloudWatch`

(Optional) Sie können Systemprotokolle in CloudWatch -Protokolle hochladen.

Standard: `false`

`minimumLogLevel`

(Optional) Die Mindestebene der hochzuladenden Protokollnachrichten. Diese Mindeststufe gilt nur, wenn Sie die Greengrass-Kernkomponente für die Ausgabe von Protokollen im JSON-Format konfigurieren. Um Protokolle im JSON-Format zu aktivieren, geben Sie `JSON` für den Protokollierungsformatparameter (`logging.format`) an.

Wählen Sie aus den folgenden Protokollebenen aus, die hier in der Reihenfolge der Ebenen aufgeführt sind:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Standard: `INFO`

`diskSpaceLimit`

(Optional) Die maximale Gesamtgröße der Greengrass-Systemprotokolldateien in der Einheit, die Sie in angeben`diskSpaceLimitUnit`. Nachdem die Gesamtgröße der Greengrass-Systemprotokolldateien diese maximale Gesamtgröße überschreitet, löscht die AWS IoT Greengrass Core-Software die ältesten Greengrass-Systemprotokolldateien.

Dieser Parameter entspricht dem Parameter zur Begrenzung der Protokollgröße (`totalLogsSize`KB) der Greengrass-Kernkomponente . Die AWS IoT Greengrass Core-Software verwendet das Minimum der beiden Werte als maximale Gesamtgröße des Greengrass-Systemprotokolls.

`diskSpaceLimitUnit`

(Optional) Die Einheit für die `diskSpaceLimit`. Wählen Sie aus den folgenden Optionen aus:

- `KB` – Kilobyte
- `MB` – Megabyte
- `GB` – Gigabyte

Standard: `KB`

`deleteLogFileAfterCloudUpload`

(Optional) Sie können eine Protokolldatei löschen, nachdem die Protokollmanagerkomponente die Protokolle in CloudWatch Logs hochgeladen hat.

Standard: `false`

`componentLogsConfigurationMap`

(Optional) Eine Zuordnung von Protokollkonfigurationen für Komponenten auf dem Core-Gerät. Jedes `componentName` Objekt in dieser Zuordnung definiert die Protokollkonfiguration für die Komponente oder Anwendung. Die Log-Manager-Komponente lädt diese Komponentenprotokolle in CloudWatch Logs hoch.

> **Important**
>
> Wir empfehlen dringend, einen einzelnen Konfigurationsschlüssel pro Komponente zu verwenden. Sie sollten nur auf eine Gruppe von Dateien abzielen, die

nur eine Protokolldatei haben, in die aktiv geschrieben wird, wenn Sie die verwenden`logFileRegex`. Wenn Sie diese Empfehlung nicht befolgen, können doppelte Protokolle in hochgeladen werden CloudWatch. Wenn Sie mehrere aktive Protokolldateien mit einem einzigen Regex anvisieren, empfehlen wir Ihnen, auf Log Manager v2.3.1 zu aktualisieren und Ihre Konfiguration mithilfe der Beispielkonfiguration zu ändern.

> **Note**
>
> Wenn Sie von einer Version des Protokollmanagers vor v2.2.0 aktualisieren, können Sie weiterhin die `componentLogsConfiguration` Liste anstelle von verwenden`componentLogsConfigurationMap`. Es wird jedoch dringend empfohlen, das Kartenformat zu verwenden, damit Sie Zusammenführungs- und Reset-Aktualisierungen verwenden können, um Konfigurationen für bestimmte Komponenten zu ändern. Informationen zum `componentLogsConfiguration` Parameter finden Sie in den Konfigurationsparametern für v2.1.x dieser Komponente.

`componentName`

Die Protokollkonfiguration für die *`componentName`* Komponente oder Anwendung für diese Protokollkonfiguration. Sie können den Namen einer Greengrass-Komponente oder einen anderen Wert angeben, um diese Protokollgruppe zu identifizieren.

Jedes Objekt enthält die folgenden Informationen:

`minimumLogLevel`

(Optional) Die Mindestebene der hochzuladenden Protokollnachrichten. Diese Mindestebene gilt nur, wenn die Protokolle dieser Komponente ein bestimmtes JSON-Format verwenden, das Sie im AWS IoT Greengrass Protokollierungsmodul-Repository auf finden GitHub.

Wählen Sie aus den folgenden Protokollebenen aus, die hier in der Reihenfolge der Ebenen aufgeführt sind:

- `DEBUG`
- `INFO`

- `WARN`
- `ERROR`

Standard: `INFO`

`diskSpaceLimit`

(Optional) Die maximale Gesamtgröße aller Protokolldateien für diese Komponente in der Einheit, die Sie in angeben`diskSpaceLimitUnit`. Nachdem die Gesamtgröße der Protokolldateien dieser Komponente diese maximale Gesamtgröße überschreitet, löscht die AWS IoT Greengrass Core-Software die ältesten Protokolldateien dieser Komponente.

Dieser Parameter bezieht sich auf den Parameter zur Begrenzung der Protokollgröße (`totalLogsSize`KB) der Greengrass-Kernkomponente . Die AWS IoT Greengrass Core-Software verwendet das Minimum der beiden Werte als maximale Gesamtprotokollgröße für diese Komponente.

`diskSpaceLimitUnit`

(Optional) Die Einheit für die `diskSpaceLimit`. Wählen Sie aus den folgenden Optionen aus:

- `KB` – Kilobyte
- `MB` – Megabyte
- `GB` – Gigabyte

Standard: `KB`

`logFileDirectoryPath`

(Optional) Der Pfad zu dem Ordner, der die Protokolldateien dieser Komponente enthält.

Sie müssen diesen Parameter nicht für Greengrass-Komponenten angeben, die in der Standardausgabe (stdout) und im Standardfehler (stderr) gedruckt werden.

Standard: */greengrass/v2*`/logs`.

`logFileRegex`

(Optional) Ein regulärer Ausdruck, der das Format des Protokolldateinamens angibt, das die Komponente oder Anwendung verwendet. Die Log Manager-Komponente

verwendet diesen regulären Ausdruck, um Protokolldateien im Ordner unter zu identifizieren`logFileDirectoryPath`.

Sie müssen diesen Parameter nicht für Greengrass-Komponenten angeben, die in der Standardausgabe (stdout) und im Standardfehler (stderr) gedruckt werden.

Wenn Ihre Komponente oder Anwendung Protokolldateien rotiert, geben Sie einen Regex an, der den rotierten Protokolldateinamen entspricht. Sie können beispielsweise angeben, **`hello_world\\\\w*.log`** um Protokolle für eine Hello World-Anwendung hochzuladen. Das `\\\\w*` Muster entspricht null oder mehr Wortzeichen, die alphanumerische Zeichen und Unterstriche enthalten. Dieser Regex gleicht Protokolldateien mit und ohne Zeitstempel im Namen ab. In diesem Beispiel lädt der Protokollmanager die folgenden Protokolldateien hoch:

- `hello_world.log` – Die neueste Protokolldatei für die Hello World-Anwendung.
- `hello_world_2020_12_15_17_0.log` – Eine ältere Protokolldatei für die Hello World-Anwendung.

Standard: *`componentName`*`\\\\w*.log`, wobei *`componentName`* der Name der Komponente für diese Protokollkonfiguration ist.

`deleteLogFileAfterCloudUpload`

(Optional) Sie können eine Protokolldatei löschen, nachdem die Protokollmanagerkomponente die Protokolle in CloudWatch Logs hochgeladen hat.

Standard: `false`

`multiLineStartPattern`

(Optional) Ein regulärer Ausdruck, der angibt, wann eine Protokollnachricht in einer neuen Zeile eine neue Protokollnachricht ist. Wenn der reguläre Ausdruck nicht mit der neuen Zeile übereinstimmt, hängt die Protokollmanagerkomponente die neue Zeile an die Protokollnachricht für die vorherige Zeile an.

Standardmäßig prüft die Log-Manager-Komponente, ob die Zeile mit einem Leerzeichen beginnt, z. B. einer Registerkarte oder einem Leerzeichen. Andernfalls behandelt der Protokollmanager diese Zeile als neue Protokollnachricht. Andernfalls wird diese Zeile an die aktuelle Protokollnachricht angehängt. Dieses Verhalten stellt sicher, dass die Log-Manager-Komponente keine Nachrichten aufteilt, die sich über mehrere Zeilen erstrecken, z. B. Stack-Ablaufverfolgungen.

`periodicUploadIntervalSec`

(Optional) Der Zeitraum in Sekunden, in dem die Protokollmanager-Komponente nach neuen Protokolldateien zum Hochladen sucht.

Standard: `300` (5 Minuten)

Minimum: `0.000001` (1 Mikrosekunde)

Example Beispiel: Aktualisierung der Konfigurationszusammenführung

Die folgende Beispielkonfiguration gibt an, Systemprotokolle und `com.example.HelloWorld` Komponentenprotokolle in CloudWatch Protokolle hochzuladen.

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfigurationMap": {
      "com.example.HelloWorld": {
        "minimumLogLevel": "INFO",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  },
  "periodicUploadIntervalSec": "300"
}
```

Example Beispiel: Konfiguration zum Hochladen mehrerer aktiver Protokolldateien mit Log Manager v2.3.1

Die folgende Beispielkonfiguration ist das empfohlene Beispiel, wenn Sie mehrere aktive Protokolldateien anvisieren möchten. Diese Beispielkonfiguration gibt an, welche aktiven Protokolldateien Sie in hochladen möchten CloudWatch. Bei Verwendung dieser Konfigurationsbeispielkonfiguration werden auch alle rotierten Dateien hochgeladen, die mit

dem übereinstimmen`logFileRegex`. Diese Beispielkonfiguration wird auf Log Manager v2.3.1 unterstützt.

```
{
  "logsUploaderConfiguration": {
    "componentLogsConfigurationMap": {
      "com.example.A": {
        "logFileRegex": "com.example.A\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
      "com.example.B": {
        "logFileRegex": "com.example.B\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  },
  "periodicUploadIntervalSec": "10"
}
```

v2.2.x

`logsUploaderConfiguration`

(Optional) Die Konfiguration für Protokolle, die die Log-Manager-Komponente hochlädt. Dieses Objekt enthält die folgenden Informationen:

`systemLogsConfiguration`

(Optional) Die Konfiguration für -AWS IoT GreengrassCore-Softwaresystemprotokolle, die Protokolle aus dem Greengrass-Kern und Plugin-Komponenten enthalten. Geben Sie diese Konfiguration an, damit die Log Manager-Komponente Systemprotokolle verwalten kann. Dieses Objekt enthält die folgenden Informationen:

`uploadToCloudWatch`

(Optional) Sie können Systemprotokolle in CloudWatch -Protokolle hochladen.

Standard: `false`

`minimumLogLevel`

(Optional) Die Mindestebene der hochzuladenden Protokollnachrichten. Diese Mindeststufe gilt nur, wenn Sie die Greengrass-Kernkomponente für die Ausgabe von Protokollen im JSON-Format konfigurieren. Um Protokolle im JSON-

Format zu aktivieren, geben Sie `JSON` für den Protokollierungsformatparameter (`logging.format`) an.

Wählen Sie aus den folgenden Protokollebenen aus, die hier in der Reihenfolge der Ebenen aufgeführt sind:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Standard: `INFO`

`diskSpaceLimit`

(Optional) Die maximale Gesamtgröße der Greengrass-Systemprotokolldateien in der Einheit, die Sie in angeben`diskSpaceLimitUnit`. Nachdem die Gesamtgröße der Greengrass-Systemprotokolldateien diese maximale Gesamtgröße überschreitet, löscht die AWS IoT Greengrass Core-Software die ältesten Greengrass-Systemprotokolldateien.

Dieser Parameter entspricht dem Parameter für die Protokollgrößenbeschränkung (`totalLogsSize`KB) der Greengrass-Kernkomponente . Die AWS IoT Greengrass Core-Software verwendet das Minimum der beiden Werte als maximale Gesamtgröße des Greengrass-Systemprotokolls.

`diskSpaceLimitUnit`

(Optional) Die Einheit für die `diskSpaceLimit`. Wählen Sie aus den folgenden Optionen aus:

- `KB` – Kilobyte
- `MB` – Megabyte
- `GB` – Gigabyte

Standard: `KB`

`deleteLogFileAfterCloudUpload`

(Optional) Sie können eine Protokolldatei löschen, nachdem die Protokollmanagerkomponente die Protokolle in CloudWatch Logs hochgeladen hat.

Standard: `false`

`componentLogsConfigurationMap`

(Optional) Eine Zuordnung von Protokollkonfigurationen für Komponenten auf dem Core-Gerät. Jedes `componentName` Objekt in dieser Zuordnung definiert die Protokollkonfiguration für die Komponente oder Anwendung. Die Log-Manager-Komponente lädt diese Komponentenprotokolle in CloudWatch Logs hoch.

> **Note**
>
> Wenn Sie von einer Version des Protokollmanagers vor v2.2.0 aktualisieren, können Sie weiterhin die `componentLogsConfiguration` Liste anstelle von verwenden`componentLogsConfigurationMap`. Es wird jedoch dringend empfohlen, das Kartenformat zu verwenden, damit Sie Zusammenführungs- und Reset-Aktualisierungen verwenden können, um Konfigurationen für bestimmte Komponenten zu ändern. Informationen zum `componentLogsConfiguration` Parameter finden Sie in den Konfigurationsparametern für v2.1.x dieser Komponente.

`componentName`

Die Protokollkonfiguration für die *`componentName`* Komponente oder Anwendung für diese Protokollkonfiguration. Sie können den Namen einer Greengrass-Komponente oder einen anderen Wert angeben, um diese Protokollgruppe zu identifizieren.

Jedes Objekt enthält die folgenden Informationen:

`minimumLogLevel`

(Optional) Die Mindestebene der hochzuladenden Protokollnachrichten. Diese Mindestebene gilt nur, wenn die Protokolle dieser Komponente ein bestimmtes JSON-Format verwenden, das Sie im AWS IoT Greengrass Protokollierungsmodul-Repository auf finden GitHub.

Wählen Sie aus den folgenden Protokollebenen aus, die hier in der Reihenfolge der Ebenen aufgeführt sind:

- `DEBUG`
- `INFO`
- `WARN`

- `ERROR`

Standard: `INFO`

`diskSpaceLimit`

(Optional) Die maximale Gesamtgröße aller Protokolldateien für diese Komponente in der Einheit, die Sie in angeben`diskSpaceLimitUnit`. Nachdem die Gesamtgröße der Protokolldateien dieser Komponente diese maximale Gesamtgröße überschreitet, löscht die AWS IoT Greengrass Core-Software die ältesten Protokolldateien dieser Komponente.

Dieser Parameter bezieht sich auf den Parameter zur Begrenzung der Protokollgröße (`totalLogsSizeKB`) der Greengrass-Kernkomponente . Die AWS IoT Greengrass Core-Software verwendet das Minimum der beiden Werte als maximale Gesamtprotokollgröße für diese Komponente.

`diskSpaceLimitUnit`

(Optional) Die Einheit für die `diskSpaceLimit`. Wählen Sie aus den folgenden Optionen aus:

- `KB` – Kilobyte
- `MB` – Megabyte
- `GB` – Gigabyte

Standard: `KB`

`logFileDirectoryPath`

(Optional) Der Pfad zu dem Ordner, der die Protokolldateien dieser Komponente enthält.

Sie müssen diesen Parameter nicht für Greengrass-Komponenten angeben, die in der Standardausgabe (stdout) und im Standardfehler (stderr) gedruckt werden.

Standard: */greengrass/v2*`/logs`.

`logFileRegex`

(Optional) Ein regulärer Ausdruck, der das Format des Protokolldateinamens angibt, das die Komponente oder Anwendung verwendet. Die Log Manager-Komponente verwendet diesen regulären Ausdruck, um Protokolldateien im Ordner unter zu identifizieren`logFileDirectoryPath`.

Sie müssen diesen Parameter nicht für Greengrass-Komponenten angeben, die in der Standardausgabe (stdout) und im Standardfehler (stderr) gedruckt werden.

Wenn Ihre Komponente oder Anwendung Protokolldateien rotiert, geben Sie einen Regex an, der den rotierten Protokolldateinamen entspricht. Sie können beispielsweise angeben, **`hello_world\\\\w*.log`** um Protokolle für eine Hello World-Anwendung hochzuladen. Das `\\\\w*` Muster entspricht null oder mehr Wortzeichen, die alphanumerische Zeichen und Unterstriche enthalten. Dieser Regex gleicht Protokolldateien mit und ohne Zeitstempel im Namen ab. In diesem Beispiel lädt der Protokollmanager die folgenden Protokolldateien hoch:

- `hello_world.log` – Die neueste Protokolldatei für die Hello World-Anwendung.
- `hello_world_2020_12_15_17_0.log` – Eine ältere Protokolldatei für die Hello World-Anwendung.

Standard: *`componentName`*`\\\\w*.log`, wobei *`componentName`* der Name der Komponente für diese Protokollkonfiguration ist.

`deleteLogFileAfterCloudUpload`

(Optional) Sie können eine Protokolldatei löschen, nachdem die Protokollmanagerkomponente die Protokolle in CloudWatch Logs hochgeladen hat.

Standard: `false`

`multiLineStartPattern`

(Optional) Ein regulärer Ausdruck, der angibt, wann eine Protokollnachricht in einer neuen Zeile eine neue Protokollnachricht ist. Wenn der reguläre Ausdruck nicht mit der neuen Zeile übereinstimmt, hängt die Protokollmanagerkomponente die neue Zeile an die Protokollnachricht für die vorherige Zeile an.

Standardmäßig prüft die Log-Manager-Komponente, ob die Zeile mit einem Leerzeichen beginnt, z. B. einer Registerkarte oder einem Leerzeichen. Andernfalls behandelt der Protokollmanager diese Zeile als neue Protokollnachricht. Andernfalls wird diese Zeile an die aktuelle Protokollnachricht angehängt. Dieses Verhalten stellt sicher, dass die Log-Manager-Komponente keine Nachrichten aufteilt, die sich über mehrere Zeilen erstrecken, z. B. Stack-Ablaufverfolgungen.

`periodicUploadIntervalSec`

(Optional) Der Zeitraum in Sekunden, in dem die Protokollmanager-Komponente nach neuen Protokolldateien zum Hochladen sucht.

Standard: `300` (5 Minuten)

Minimum: `0.000001` (1 Mikrosekunde)

Example Beispiel: Aktualisierung der Konfigurationszusammenführung

Die folgende Beispielkonfiguration gibt an, Systemprotokolle und `com.example.HelloWorld` Komponentenprotokolle in CloudWatch Protokolle hochzuladen.

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfigurationMap": {
      "com.example.HelloWorld": {
        "minimumLogLevel": "INFO",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  },
  "periodicUploadIntervalSec": "300"
}
```

v2.1.x

`logsUploaderConfiguration`

(Optional) Die Konfiguration für Protokolle, die die Log-Manager-Komponente hochlädt. Dieses Objekt enthält die folgenden Informationen:

`systemLogsConfiguration`

(Optional) Die Konfiguration für -AWS IoT GreengrassCore-Softwaresystemprotokolle, die Protokolle aus dem Greengrass-Kern und den Plugin-Komponenten enthalten. Geben Sie diese Konfiguration an, damit die Log Manager-Komponente Systemprotokolle verwalten kann. Dieses Objekt enthält die folgenden Informationen:

`uploadToCloudWatch`

(Optional) Sie können Systemprotokolle in CloudWatch -Protokolle hochladen.

Standard: `false`

`minimumLogLevel`

(Optional) Die Mindestebene der hochzuladenden Protokollnachrichten. Diese Mindeststufe gilt nur, wenn Sie die Greengrass-Kernkomponente für die Ausgabe von Protokollen im JSON-Format konfigurieren. Um Protokolle im JSON-Format zu aktivieren, geben Sie `JSON` für den Protokollierungsformatparameter (`logging.format`) an.

Wählen Sie aus den folgenden Protokollebenen aus, die hier in der Reihenfolge der Ebenen aufgeführt sind:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Standard: `INFO`

`diskSpaceLimit`

(Optional) Die maximale Gesamtgröße der Greengrass-Systemprotokolldateien in der Einheit, die Sie in angeben`diskSpaceLimitUnit`. Nachdem die Gesamtgröße der Greengrass-Systemprotokolldateien diese maximale Gesamtgröße überschreitet, löscht die AWS IoT Greengrass Core-Software die ältesten Greengrass-Systemprotokolldateien.

Dieser Parameter entspricht dem Parameter für die Protokollgrößenbeschränkung (`totalLogsSizeKB`) der Greengrass-Kernkomponente . Die AWS IoT Greengrass Core-Software verwendet das Minimum der beiden Werte als maximale Gesamtgröße des Greengrass-Systemprotokolls.

`diskSpaceLimitUnit`

(Optional) Die Einheit für die `diskSpaceLimit`. Wählen Sie aus den folgenden Optionen aus:

- `KB` – Kilobyte

- `MB` – Megabyte
- `GB` – Gigabyte

Standard: `KB`

`deleteLogFileAfterCloudUpload`

(Optional) Sie können eine Protokolldatei löschen, nachdem die Protokollmanagerkomponente die Protokolle in CloudWatch Logs hochgeladen hat.

Standard: `false`

`componentLogsConfiguration`

(Optional) Eine Liste von Protokollkonfigurationen für Komponenten auf dem Core-Gerät. Jede Konfiguration in dieser Liste definiert die Protokollkonfiguration für eine Komponente oder Anwendung. Die Log Manager-Komponente lädt diese Komponentenprotokolle in CloudWatch Logs hoch

Jedes Objekt enthält die folgenden Informationen:

`componentName`

Der Name der Komponente oder Anwendung für diese Protokollkonfiguration. Sie können den Namen einer Greengrass-Komponente oder einen anderen Wert angeben, um diese Protokollgruppe zu identifizieren.

`minimumLogLevel`

(Optional) Die Mindestebene der hochzuladenden Protokollnachrichten. Diese Mindestebene gilt nur, wenn die Protokolle dieser Komponente ein bestimmtes JSON-Format verwenden, das Sie im AWS IoT Greengrass Protokollierungsmodul-Repository auf finden GitHub.

Wählen Sie aus den folgenden Protokollebenen aus, die hier in der Reihenfolge der Ebenen aufgeführt sind:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Standard: `INFO`

`diskSpaceLimit`

(Optional) Die maximale Gesamtgröße aller Protokolldateien für diese Komponente in der Einheit, die Sie in angeben`diskSpaceLimitUnit`. Nachdem die Gesamtgröße der Protokolldateien dieser Komponente diese maximale Gesamtgröße überschreitet, löscht die AWS IoT Greengrass Core-Software die ältesten Protokolldateien dieser Komponente.

Dieser Parameter bezieht sich auf den Parameter zur Begrenzung der Protokollgröße (`totalLogsSizeKB`) der Greengrass-Kernkomponente . Die AWS IoT Greengrass Core-Software verwendet das Minimum der beiden Werte als maximale Gesamtprotokollgröße für diese Komponente.

`diskSpaceLimitUnit`

(Optional) Die Einheit für die `diskSpaceLimit`. Wählen Sie aus den folgenden Optionen aus:

- `KB` – Kilobyte
- `MB` – Megabyte
- `GB` – Gigabyte

Standard: `KB`

`logFileDirectoryPath`

(Optional) Der Pfad zu dem Ordner, der die Protokolldateien dieser Komponente enthält.

Sie müssen diesen Parameter nicht für Greengrass-Komponenten angeben, die in der Standardausgabe (stdout) und im Standardfehler (stderr) gedruckt werden.

Standard: *`/greengrass/v2`*`/logs`.

`logFileRegex`

(Optional) Ein regulärer Ausdruck, der das Format des Protokolldateinamens angibt, das die Komponente oder Anwendung verwendet. Die Log Manager-Komponente verwendet diesen regulären Ausdruck, um Protokolldateien im Ordner unter zu identifizieren`logFileDirectoryPath`.

Sie müssen diesen Parameter nicht für Greengrass-Komponenten angeben, die in der Standardausgabe (stdout) und im Standardfehler (stderr) gedruckt werden.

Wenn Ihre Komponente oder Anwendung Protokolldateien rotiert, geben Sie einen Regex an, der den rotierten Protokolldateinamen entspricht. Sie können beispielsweise angeben, **`hello_world\\\\w*.log`** um Protokolle für eine Hello World-Anwendung hochzuladen. Das `\\\\w*` Muster entspricht null oder mehr Wortzeichen, die alphanumerische Zeichen und Unterstriche enthalten. Dieser Regex gleicht Protokolldateien mit und ohne Zeitstempel im Namen ab. In diesem Beispiel lädt der Protokollmanager die folgenden Protokolldateien hoch:

- `hello_world.log` – Die neueste Protokolldatei für die Hello World-Anwendung.
- `hello_world_2020_12_15_17_0.log` – Eine ältere Protokolldatei für die Hello World-Anwendung.

Standard: *componentName*`\\\\w*.log`, wobei *componentName* der Name der Komponente für diese Protokollkonfiguration ist.

`deleteLogFileAfterCloudUpload`

(Optional) Sie können eine Protokolldatei löschen, nachdem die Protokollmanagerkomponente die Protokolle in CloudWatch Logs hochgeladen hat.

Standard: `false`

`multiLineStartPattern`

(Optional) Ein regulärer Ausdruck, der angibt, wann eine Protokollnachricht in einer neuen Zeile eine neue Protokollnachricht ist. Wenn der reguläre Ausdruck nicht mit der neuen Zeile übereinstimmt, hängt die Protokollmanagerkomponente die neue Zeile an die Protokollnachricht für die vorherige Zeile an.

Standardmäßig prüft die Log-Manager-Komponente, ob die Zeile mit einem Leerzeichen beginnt, z. B. einer Registerkarte oder einem Leerzeichen. Andernfalls behandelt der Protokollmanager diese Zeile als neue Protokollnachricht. Andernfalls wird diese Zeile an die aktuelle Protokollnachricht angehängt. Dieses Verhalten stellt sicher, dass die Log-Manager-Komponente keine Nachrichten aufteilt, die sich über mehrere Zeilen erstrecken, z. B. Stack-Ablaufverfolgungen.

`periodicUploadIntervalSec`

(Optional) Der Zeitraum in Sekunden, in dem die Protokollmanager-Komponente nach neuen Protokolldateien zum Hochladen sucht.

Standard: `300` (5 Minuten)

Minimum: `0.000001` (1 Mikrosekunde)

Example Beispiel: Aktualisierung der Konfigurationszusammenführung

Die folgende Beispielkonfiguration gibt an, Systemprotokolle und `com.example.HelloWorld` Komponentenprotokolle in CloudWatch Protokolle hochzuladen.

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfiguration": [
      {
        "componentName": "com.example.HelloWorld",
        "minimumLogLevel": "INFO",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    ]
  },
  "periodicUploadIntervalSec": "300"
}
```

v2.0.x

`logsUploaderConfiguration`

(Optional) Die Konfiguration für Protokolle, die die Log-Manager-Komponente hochlädt. Dieses Objekt enthält die folgenden Informationen:

`systemLogsConfiguration`

(Optional) Die Konfiguration für -AWS IoT GreengrassCore-Softwaresystemprotokolle. Geben Sie diese Konfiguration an, damit die Log Manager-Komponente Systemprotokolle verwalten kann. Dieses Objekt enthält die folgenden Informationen:

`uploadToCloudWatch`

(Optional) Sie können Systemprotokolle in CloudWatch -Protokolle hochladen.

Standard: `false`

`minimumLogLevel`

(Optional) Die Mindestebene der hochzuladenden Protokollnachrichten. Diese Mindeststufe gilt nur, wenn Sie die Greengrass-Kernkomponente für die Ausgabe von Protokollen im JSON-Format konfigurieren. Um Protokolle im JSON-Format zu aktivieren, geben Sie `JSON` für den Protokollierungsformatparameter (`logging.format`) an.

Wählen Sie aus den folgenden Protokollebenen aus, die hier in der Reihenfolge der Ebenen aufgeführt sind:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Standard: `INFO`

`diskSpaceLimit`

(Optional) Die maximale Gesamtgröße der Greengrass-Systemprotokolldateien in der Einheit, die Sie in angeben`diskSpaceLimitUnit`. Nachdem die Gesamtgröße der Greengrass-Systemprotokolldateien diese maximale Gesamtgröße überschreitet, löscht die AWS IoT Greengrass Core-Software die ältesten Greengrass-Systemprotokolldateien.

Dieser Parameter entspricht dem Parameter für die Protokollgrößenbeschränkung (`totalLogsSizeKB`) der Greengrass-Kernkomponente . Die AWS IoT Greengrass Core-Software verwendet das Minimum der beiden Werte als maximale Gesamtgröße des Greengrass-Systemprotokolls.

`diskSpaceLimitUnit`

(Optional) Die Einheit für die `diskSpaceLimit`. Wählen Sie aus den folgenden Optionen aus:

- `KB` – Kilobyte
- `MB` – Megabyte
- `GB` – Gigabyte

Standard: `KB`

`deleteLogFileAfterCloudUpload`

(Optional) Sie können eine Protokolldatei löschen, nachdem die Protokollmanagerkomponente die Protokolle in CloudWatch Logs hochgeladen hat.

Standard: `false`

`componentLogsConfiguration`

(Optional) Eine Liste von Protokollkonfigurationen für Komponenten auf dem Core-Gerät. Jede Konfiguration in dieser Liste definiert die Protokollkonfiguration für eine Komponente oder Anwendung. Die Log Manager-Komponente lädt diese Komponentenprotokolle in CloudWatch Logs hoch

Jedes Objekt enthält die folgenden Informationen:

`componentName`

Der Name der Komponente oder Anwendung für diese Protokollkonfiguration. Sie können den Namen einer Greengrass-Komponente oder einen anderen Wert angeben, um diese Protokollgruppe zu identifizieren.

`minimumLogLevel`

(Optional) Die Mindestebene der hochzuladenden Protokollnachrichten. Diese Mindestebene gilt nur, wenn die Protokolle dieser Komponente ein bestimmtes JSON-Format verwenden, das Sie im AWS IoT Greengrass Protokollierungsmodul-Repository auf finden GitHub.

Wählen Sie aus den folgenden Protokollebenen aus, die hier in der Reihenfolge der Ebenen aufgeführt sind:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Standard: `INFO`

`diskSpaceLimit`

(Optional) Die maximale Gesamtgröße aller Protokolldateien für diese Komponente in der Einheit, die Sie in angeben`diskSpaceLimitUnit`. Nachdem die Gesamtgröße der Protokolldateien dieser Komponente diese maximale Gesamtgröße überschreitet, löscht die AWS IoT Greengrass Core-Software die ältesten Protokolldateien dieser Komponente.

Dieser Parameter bezieht sich auf den Parameter zur Begrenzung der Protokollgröße (`totalLogsSizeKB`) der Greengrass-Kernkomponente . Die AWS IoT Greengrass Core-Software verwendet das Minimum der beiden Werte als maximale Gesamtprotokollgröße für diese Komponente.

`diskSpaceLimitUnit`

(Optional) Die Einheit für die `diskSpaceLimit`. Wählen Sie aus den folgenden Optionen aus:

- `KB` – Kilobyte
- `MB` – Megabyte
- `GB` – Gigabyte

Standard: `KB`

`logFileDirectoryPath`

Der Pfad zu dem Ordner, der die Protokolldateien dieser Komponente enthält.

Um die Protokolle einer Greengrass-Komponente hochzuladen, geben Sie an ***/greengrass/v2*/logs**und ersetzen Sie durch */greengrass/v2* Ihren Greengrass-Stammordner.

`logFileRegex`

Ein regulärer Ausdruck, der das Format des Protokolldateinamens angibt, das die Komponente oder Anwendung verwendet. Die Log Manager-Komponente verwendet diesen regulären Ausdruck, um Protokolldateien im Ordner unter zu identifizieren`logFileDirectoryPath`.

Um die Protokolle einer Greengrass-Komponente hochzuladen, geben Sie einen Regex an, der den rotierten Protokolldateinamen entspricht. Sie können beispielsweise

angeben, **`com.example.HelloWorld\\w*.log`** um Protokolle für eine Hello-World-Komponente hochzuladen. Das `\\w*` Muster entspricht null oder mehr Wortzeichen, die alphanumerische Zeichen und Unterstriche enthalten. Dieser Regex gleicht Protokolldateien mit und ohne Zeitstempel im Namen ab. In diesem Beispiel lädt der Protokollmanager die folgenden Protokolldateien hoch:

- `com.example.HelloWorld.log` – Die neueste Protokolldatei für die Hello World-Komponente.
- `com.example.HelloWorld_2020_12_15_17_0.log` – Eine ältere Protokolldatei für die Hello World-Komponente. Der Greengrass-Kernus fügt den Protokolldateien einen rotierenden Zeitstempel hinzu.

`deleteLogFileAfterCloudUpload`

(Optional) Sie können eine Protokolldatei löschen, nachdem die Protokollmanagerkomponente die Protokolle in CloudWatch Logs hochgeladen hat.

Standard: `false`

`multiLineStartPattern`

(Optional) Ein regulärer Ausdruck, der angibt, wann eine Protokollnachricht in einer neuen Zeile eine neue Protokollnachricht ist. Wenn der reguläre Ausdruck nicht mit der neuen Zeile übereinstimmt, hängt die Protokollmanagerkomponente die neue Zeile an die Protokollnachricht für die vorherige Zeile an.

Standardmäßig prüft die Log-Manager-Komponente, ob die Zeile mit einem Leerzeichen beginnt, z. B. einer Registerkarte oder einem Leerzeichen. Andernfalls behandelt der Protokollmanager diese Zeile als neue Protokollnachricht. Andernfalls wird diese Zeile an die aktuelle Protokollnachricht angehängt. Dieses Verhalten stellt sicher, dass die Log-Manager-Komponente keine Nachrichten aufteilt, die sich über mehrere Zeilen erstrecken, z. B. Stack-Ablaufverfolgungen.

`periodicUploadIntervalSec`

(Optional) Der Zeitraum in Sekunden, in dem die Protokollmanager-Komponente nach neuen Protokolldateien zum Hochladen sucht.

Standard: `300` (5 Minuten)

Minimum: `0.000001` (1 Mikrosekunde)

Example Beispiel: Aktualisierung der Konfigurationszusammenführung

Die folgende Beispielkonfiguration gibt an, Systemprotokolle und `com.example.HelloWorld` Komponentenprotokolle in CloudWatch Protokolle hochzuladen.

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfiguration": [
      {
        "componentName": "com.example.HelloWorld",
        "minimumLogLevel": "INFO",
        "logFileDirectoryPath": "/greengrass/v2/logs",
        "logFileRegex": "com.example.HelloWorld\\w*.log",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    ]
  },
  "periodicUploadIntervalSec": "300"
}
```

Verwendung

Die Log Manager-Komponente lädt in die folgenden Protokollgruppen und Protokollstreams hoch.

2.1.0 and later

Protokollgruppenname

```
/aws/greengrass/componentType/region/componentName
```

Der Name der Protokollgruppe verwendet die folgenden Variablen:

- `componentType` – Der Typ der Komponente, der einer der folgenden sein kann:

- `GreengrassSystemComponent` – Diese Protokollgruppe enthält Protokolle für die Kern- und Plugin-Komponenten, die in derselben JVM wie der Greengrass-Kern ausgeführt werden. Die Komponente ist Teil des Greengrass-Kerns .
- `UserComponent` – Diese Protokollgruppe enthält Protokolle für generische Komponenten, Lambda-Komponenten und andere Anwendungen auf dem Gerät. Die Komponente ist nicht Teil des Greengrass-Kerns.

Weitere Informationen finden Sie unter Komponententypen.

- `region` – Die AWS Region, die das Core-Gerät verwendet.
- `componentName` – Der Name der Komponente. Für Systemprotokolle ist dieser Wert `System`.

Name des Protokollstreams

```
/date/thing/thingName
```

Der Name des Protokollstreams verwendet die folgenden Variablen:

- `date` – Das Datum des Protokolls, z. B. `2020/12/15`. Die Log-Manager-Komponente verwendet das `yyyy/MM/dd` Format .
- `thingName` – Der Name des Core-Geräts.

> **Note**
>
> Wenn ein Objektname einen Doppelpunkt (`:`) enthält, ersetzt der Protokollmanager den Doppelpunkt durch ein Pluszeichen (`+`).

2.0.x

Protokollgruppenname

```
/aws/greengrass/componentType/region/componentName
```

Der Name der Protokollgruppe verwendet die folgenden Variablen:

- `componentType` – Der Typ der Komponente, der einer der folgenden sein kann:
  - `GreengrassSystemComponent` – Die Komponente ist Teil des Greengrass-Kerns .

  - `UserComponent` – Die Komponente ist nicht Teil des Greengrass-Kerns. Der Protokollmanager verwendet diesen Typ für Greengrass-Komponenten und andere Anwendungen auf dem Gerät.
- `region` – Die AWS Region, die das Core-Gerät verwendet.
- `componentName` – Der Name der Komponente. Für Systemprotokolle ist dieser Wert `System`.

Name des Protokollstreams

```
/date/deploymentTargets/thingName
```

Der Name des Protokollstreams verwendet die folgenden Variablen:

- `date` – Das Datum des Protokolls, z. B. `2020/12/15`. Die Log-Manager-Komponente verwendet das `yyyy/MM/dd` Format .
- `deploymentTargets` – Die Objekte, deren Bereitstellungen die Komponente enthalten. Die Protokollmanagerkomponente trennt jedes Ziel durch einen Schrägstrich. Wenn die Komponente aufgrund einer lokalen Bereitstellung auf dem Core-Gerät ausgeführt wird, ist dieser Wert `LOCAL_DEPLOYMENT`.

  Stellen Sie sich ein Beispiel vor`MyGreengrassCore`, in dem Sie ein Core-Gerät mit dem Namen haben und das Core-Gerät zwei Bereitstellungen hat:

  - Eine Bereitstellung, die auf das Core-Gerät abzielt, `MyGreengrassCore`.
  - Eine Bereitstellung, die auf eine Objektgruppe mit dem Namen abzielt`MyGreengrassCoreGroup`, die das Core-Gerät enthält.

  Die `deploymentTargets` für dieses Core-Gerät sind `thing/MyGreengrassCore/thinggroup/MyGreengrassCoreGroup`.
- `thingName` – Der Name des Core-Geräts.

Formate für Protokolleinträge.

Der Greengrass-Kern schreibt Protokolldateien entweder im Zeichenfolgen- oder JSON-Format. Für Systemprotokolle steuern Sie das Format, indem Sie das `format` Feld des `logging` Eintrags festlegen. Sie finden den `logging` Eintrag in der Konfigurationsdatei der Greengrass-Kernkomponente. Weitere Informationen finden Sie unter Greengrass-Kernkonfiguration.

Das Textformat ist frei und akzeptiert jede Zeichenfolge. Die folgende Servicemeldung für den Flottenstatus ist ein Beispiel für die Protokollierung im Zeichenfolgenformat:

```
2023-03-26T18:18:27.271Z [INFO] (pool-1-thread-2)
com.aws.greengrass.status.FleetStatusService: fss-status-update-published.
Status update published to FSS. {trigger=CADENCE, serviceName=FleetStatusService,
currentState=RUNNING}
```

Sie sollten das JSON-Format verwenden, wenn Sie Protokolle mit dem Greengrass-CLI-Protokollbefehl anzeigen oder programmgesteuert mit Protokollen interagieren möchten. Im folgenden Beispiel wird die JSON-Form beschrieben:

```
{
   "loggerName": <string>,
   "level": <"DEBUG" | "INFO" | "ERROR" | "TRACE" | "WARN">,
   "eventType": <string, optional>,
   "cause": <string, optional>,
   "contexts": {},
   "thread": <string>,
   "message": <string>,
   "timestamp": <epoch time> # Needs to be epoch time
}
```

Um die Ausgabe der Protokolle Ihrer Komponente zu steuern, können Sie die `minimumLogLevel` Konfigurationsoption verwenden. Um diese Option verwenden zu können, muss Ihre Komponente ihre Protokolleinträge im JSON-Format schreiben. Sie sollten dasselbe Format wie die Systemprotokolldatei verwenden.

Lokale Protokolldatei

Diese Komponente verwendet dieselbe Protokolldatei wie die Greengrass-Kernkomponente.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

So zeigen Sie die Protokolle dieser Komponente an

- Führen Sie den folgenden Befehl auf dem Core-Gerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

  Linux

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

  Windows (PowerShell)

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

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in jeder Version der Komponente beschrieben.

Version	Änderungen
2.3.7	Version für Greengrass-Kern Version 2.12.0 aktualisiert.
2.3.6	Fehlerbehebungen und Verbesserungen • Passt die Protokollstufen für bestimmte Fehler an.
2.3.5	Verbesserungen Verbessert die Geschwindigkeit des Protokoll-Uploads. Version für Greengrass-Kern Version 2.11.0 aktualisiert.
2.3.4	Fehlerbehebungen und Verbesserungen • Fügt Unterstützung für die Einstellung des `periodicUploadIntervalSec` Parameters auf Bruchwerte hinzu. Das Minimum beträgt 1 Mikrosekunde. • Behebt ein Problem, bei dem Log Manager die CloudWatch `putLogEvents` Limits nicht einhält.

Version	Änderungen
2.3.3	Version für Greengrass-Kern Version 2.10.0 aktualisiert.
2.3.2	Fehlerbehebungen und Verbesserungen • Verbessert die Speicherplatzverwaltung, sodass Protokolldateien nicht gelöscht werden, bevor sie hochgeladen werden. • Behebt Probleme mit der Cache-Verwaltung. • Zusätzliche kleinere Fehlerbehebungen und Verbesserungen.
2.3.1	Fehlerbehebungen und Verbesserungen • Behebt ein Problem, bei dem s, die Dateigruppen mit mehreren aktiven Protokolldateien anvisieren, doppelte Einträge in hochladen CloudWatch. • Zusätzliche kleinere Fehlerbehebungen und Verbesserungen.
2.3.0	**Note** Wir empfehlen, dass Sie ein Upgrade auf Greengrass-Kern 2.9.1 durchführen, wenn Sie ein Upgrade auf Log Manager 2.3.0 durchführen. Neue Features Reduziert Protokollverzögerungen, indem aktive Protokolldateien verarbeitet und direkt hochgeladen werden, anstatt darauf zu warten, dass neue Dateien rotiert werden. Fehlerbehebungen und Verbesserungen • Verbessert die Unterstützung der Protokollrotation beim Rotieren von Dateien mit einem eindeutigen Namen. • Zusätzliche kleinere Fehlerbehebungen und Verbesserungen.
2.2.8	Version für Greengrass-Kern Version 2.9.0 aktualisiert.
2.2.7	Version für Greengrass-Kern Version 2.8.0 aktualisiert.

Version	Änderungen
2.2.6	Version für Greengrass-Kern Version 2.7.0 aktualisiert.
2.2.5	Version für Greengrass-Kern Version 2.6.0 aktualisiert.
2.2.4	Fehlerbehebungen und Verbesserungen • Verbessert die Stabilität beim Umgang mit ungültigen Konfigurationen. • Zusätzliche kleinere Korrekturen und Verbesserungen.
2.2.3	Fehlerbehebungen und Verbesserungen • Verbessert die Stabilität in bestimmten Szenarien, in denen die Komponente neu gestartet wird oder Fehler auftreten. • Behebt Probleme, bei denen große Protokollmeldungen und große Protokolldateien in bestimmten Szenarien nicht hochgeladen werden konnten. • Behebt Probleme mit der Handhabung von Konfigurationsrücksetzungsaktualisierungen durch diese Komponente. • Behebt ein Problem, bei dem ein `null` `diskSpaceLimit` Konfigurationswert die Bereitstellung der Komponente verhindert hat.
2.2.2	Fehlerbehebungen und Verbesserungen • Fügt Unterstützung für Protokollnachrichten hinzu, die größer als 256 Kilobyte sind. Die Log-Manager-Komponente teilt diese großen Protokollmeldungen in mehrere Meldungen mit demselben Protokollereigniszeitstempel auf.
2.2.1	Version für Greengrass-Kern Version 2.5.0 aktualisiert.
2.2.0	Neues Feature • Fügt den `componentLogsConfigurationMap` Konfigurationsparameter hinzu, um ein Zuordnungsformat für Komponentenprotokollkonfigurationen zu unterstützen. Jedes `componentName` Objekt in der Zuordnung definiert die Protokollkonfiguration für eine Komponente oder Anwendung.
2.1.3	Version für Greengrass-Kern Version 2.4.0 aktualisiert.

Version	Änderungen
2.1.2	Version für Greengrass-Kern Version 2.3.0 aktualisiert.
2.1.1	Fehlerbehebungen und Verbesserungen • Behebt ein Problem, bei dem die Systemprotokollkonfiguration in bestimmten Fällen nicht aktualisiert wurde.
2.1.0	Fehlerbehebungen und Verbesserungen • Verwenden Sie Standardwerte für `logFileDirectoryPath` und `logFileRegex` , die für Greengrass-Komponenten funktionieren, die auf die Standardausgabe (stdout) und den Standardfehler (stderr) gedruckt werden. • Datenverkehr beim Hochladen von Protokollen in CloudWatch Protokolle korrekt über einen konfigurierten Netzwerk-Proxy weiterleiten. • Verarbeiten Sie Doppelpunktzeichen (`:`) in Protokollstreamnamen korrekt. CloudWatch Logs-Protokollstreamnamen unterstützen keine Doppelpunkte. • Vereinfachen Sie die Namen von Protokollstreams, indem Sie Objektgruppennamen aus dem Protokollstream entfernen. • Entfernen Sie eine Fehlermeldung, die während des normalen Verhaltens ausgegeben wird.
2.0.x	Erste Version

Machine-Learning-Komponenten

AWS IoT Greengrass bietet die folgenden Machine-Learning-Komponenten, die Sie auf unterstützten Geräten bereitstellen können, um Machine-Learning-Inferenzen mithilfe von Modellen durchzuführen, die in Amazon trainiert wurden, SageMaker oder mit Ihren eigenen vortrainierten Modellen, die in Amazon S3 gespeichert sind.

AWS bietet die folgenden Kategorien von Machine Learning-Komponenten:

- Modellkomponente – Enthält Machine-Learning-Modelle als Greengrass-Artefakte.

- Laufzeitkomponente – Enthält das Skript, das das Machine Learning-Framework installiert, und seine Abhängigkeiten auf dem Greengrass-Kerngerät.
- Inferenzkomponente – Enthält den Inferenzcode und enthält Komponentenabhängigkeiten, um das Machine-Learning-Framework zu installieren und vortrainierte Machine-Learning-Modelle herunterzuladen.

Sie können den Beispiel-Inferenzcode und vortrainierte Modelle in den von bereitgestellten Machine-LearningAWS-Komponenten verwenden, um die Bildklassifizierung und Objekterkennung mit DLR und TensorFlow Lite durchzuführen. Um eine benutzerdefinierte Machine-Learning-Inferenz mit Ihren eigenen Modellen durchzuführen, die in Amazon S3 gespeichert sind, oder um ein anderes Machine-Learning-Framework zu verwenden, können Sie die Rezepte dieser öffentlichen Komponenten als Vorlagen verwenden, um benutzerdefinierte Machine-Learning-Komponenten zu erstellen. Weitere Informationen finden Sie unter Anpassen Ihrer Machine-Learning-Komponenten.

AWS IoT Greengrass enthält auch eine von bereitgestellte Komponente zur Verwaltung der Installation und des Lebenszyklus des SageMaker Edge-ManagerAWS-Agenten auf Greengrass-Core-Geräten. Mit SageMaker Edge Manager können Sie von Amazon SageMaker Neo kompilierte Modelle direkt auf Ihrem Core-Gerät verwenden. Weitere Informationen finden Sie unter Verwenden von Amazon SageMaker Edge Manager auf Greengrass-Core-Geräten.

In der folgenden Tabelle sind die Machine-Learning-Komponenten aufgeführt, die in verfügbar sindAWS IoT Greengrass.

> **Note**
>
> Mehrere AWSvon bereitgestellte Komponenten hängen von bestimmten Nebenversionen des Greengrass-Kerns ab. Aufgrund dieser Abhängigkeit müssen Sie diese Komponenten aktualisieren, wenn Sie den Greengrass-Kern auf eine neue Nebenversion aktualisieren. Informationen zu den spezifischen Versionen des Kerns, von denen jede Komponente abhängt, finden Sie im entsprechenden Komponententhema. Weitere Informationen zum Aktualisieren des Kerns finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Wenn eine Komponente sowohl den Komponententyp generisch als auch Lambda hat, ist die aktuelle Version der Komponente der generische Typ und eine frühere Version der Komponente der Lambda-Typ.

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
Lookout für Vision Edge Agent	Stellt die Laufzeit von Amazon Lookout for Vision auf dem Greengrass-Kerngerät bereit, sodass Sie Computer Vision verwenden können, um Fehler in Industrieprodukten zu finden.	Generisch	Linux	Nein
SageMaker Edge-Manager	Stellt den Amazon SageMaker Edge Manager-Agenten auf dem Greengrass-Kerngerät bereit.	Generisch	Linux, Windows	Nein
DLR-Bildklassifizierung	Inferenzkomponente, die den DLR-Bildklassifizi	Generisch	Linux, Windows	Nein

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
	erungsmodellspeicher und die DLR-Laufzeitkomponente als Abhängigkeiten verwendet, um DLR zu installieren, Bildklassifizierungsmodelle herunterzuladen und Bildklassifizierungsinferenzen auf unterstützten Geräten durchzuführen.			

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
DLR-Objekterkennung	Inferenzkomponente, die den DLR-Objekterkennungsmodellspeicher und die DLR-Laufzeitkomponente als Abhängigkeiten verwendet, um DLR zu installieren, Beispiel-Objekterkennungsmodelle herunterzuladen und Objekterkennungs-Inferenzen auf unterstützten Geräten durchzuführen.	Generisch	Linux, Windows	Nein

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
Modellspeicher für die DLR-Bildklassifizierung	Modellkomponente, die Beispielmodelle zur Bildklassifizierung von ResNet-50 als Greengrass-Artefakte enthält.	Generisch	Linux, Windows	Nein
Modellspeicher für DLR-Objekterkennung	Modellkomponente, die YOLOv3-Beispielobjekterkennungsmodelle als Greengrass-Artefakte enthält.	Generisch	Linux, Windows	Nein

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
DLR-Laufzeit	Laufzeitkomponente, die ein Installationsskript enthält, das zur Installation von DLR und seinen Abhängigkeiten auf dem Greengrass-Kerngerät verwendet wird.	Generisch	Linux, Windows	Nein

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
TensorFlow Lite-Bildklassifizierung	Inferenzkomponente, die den TensorFlow Lite-Bildklassifizierungsmodellspeicher und die TensorFlow Lite-Laufzeitkomponente als Abhängigkeiten verwendet, um TensorFlow Lite zu installieren, Beispiel-Bildklassifizierungsmodelle herunterzuladen und Bildklassifizierungsinferenzen auf unterstützten Geräten durchzuführen.	Generisch	Linux, Windows	Nein

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
TensorFlow Lite-Objekterkennung	Inferenzkomponente, die den TensorFlow Lite-Objekterkennungsmodellspeicher und die TensorFlow Lite-Laufzeitkomponente als Abhängigkeiten verwendet, um TensorFlow Lite zu installieren, Beispiel-Objekterkennungsmodelle herunterzuladen und Objekterkennungs-Inferenzen auf unterstützten Geräten durchzuführen.	Generisch	Linux, Windows	Nein

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
TensorFlow Modellspeicher für Lite-Bildklassifizierung	Modellkomponente, die ein Beispiel MobileNet - v1-Modell als Greengrass-Artefakt enthält.	Generisch	Linux, Windows	Nein
TensorFlow Modellspeicher für Lite-Objekterkennung	Modellkomponente, die ein Single Shot Detection (SSD)-MobileNet Beispielmodell als Greengrass-Artefakt enthält.	Generisch	Linux, Windows	Nein

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
TensorFlow Lite-Laufzeit	Laufzeitkomponente, die ein Installationsskript enthält, das zur Installation von TensorFlow Lite und seinen Abhängigkeiten auf dem Greengrass-Kerngerät verwendet wird.	Generisch	Linux, Windows	Nein

Lookout für Vision Edge Agent

Die Komponente Lookout for Vision Edge Agent (`aws.iot.lookoutvision.EdgeAgent`) installiert einen lokalen Laufzeitserver von Amazon Lookout for Vision, der Computer Vision verwendet, um visuelle Fehler in Industrieprodukten zu finden.

Um diese Komponente zu verwenden, erstellen Sie Machine-Learning-Modellkomponenten für Lookout for Vision und stellen Sie sie bereit. Diese Machine-Learning-Modelle prognostizieren das Vorhandensein von Anomalien in Bildern, indem sie Muster in Bildern finden, mit denen Sie das Modell trainieren. Anschließend können Sie benutzerdefinierte Greengrass-Komponenten, sogenannte Client-Anwendungskomponenten, entwickeln und bereitstellen, die Images und Videostreams für diese Laufzeitkomponente bereitstellen, um Anomalien mithilfe der Machine-Learning-Modelle zu erkennen.

Sie können die Lookout for Vision Edge Agent-API verwenden, um mit dieser Komponente von anderen Greengrass-Komponenten aus zu interagieren. Diese API wird mit gRPC implementiert,

einem Protokoll für Aufrufe von Remote-Prozeduren. Weitere Informationen finden Sie unter Schreiben einer Clientanwendungskomponente und Referenz zur Lookout for Vision Edge Agent API im Entwicklerhandbuch für Amazon Lookout for Vision.

Weitere Informationen zur Verwendung dieser Komponente finden Sie unter:

- Amazon Lookout for Vision
- Was ist Amazon Lookout for Vision? im Entwicklerhandbuch für Amazon Lookout for Vision
- Erstellen eines Lookout-for-Vision-Modells im Entwicklerhandbuch für Amazon Lookout for Vision.
- Verwenden eines Lookout for Vision-Modells auf einem Edge-Gerät im Entwicklerhandbuch für Amazon Lookout for Vision.

> **Note**
>
> Die Komponente Lookout for Vision Edge Agent ist nur in den folgenden verfügbar AWS-Regionen:
>
> - US East (Ohio)
> - USA Ost (Nord-Virginia)
> - USA West (Oregon)
> - Europe (Frankfurt)
> - Europa (Irland)
> - Asien-Pazifik (Tokio)
> - Asien-Pazifik (Seoul)

Themen

- Änderungsprotokoll

Versionen

Diese Komponente hat die folgenden Versionen:

- 1.2.x
- 1.1.x
- 1.0.x
- 0.1.x

Typ

Diese Komponente ist eine generische Komponente (`aws.greengrass.generic`). Der Greengrass-Kern führt die Lebenszyklusskripte der Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann nur auf Linux-Core-Geräten installiert werden.

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Das Greengrass-Core-Gerät muss eine Armv8 (AArch64)- oder x86_64-Architektur verwenden.
- Wenn Sie Version 1.0.0 oder höher dieser Komponente verwenden, ist Python 3.8 oder Python 3.9, einschließlich `pip`, auf dem Greengrass-Core-Gerät installiert.

  Wenn Sie Version 0.1.x dieser Komponente verwenden, ist Python 3.7, einschließlich `pip`, auf dem Greengrass-Kerngerät installiert.

  > **Important**
  >
  > Das Gerät muss eine dieser genauen Versionen von Python haben. Diese Komponente unterstützt keine späteren Versionen von Python.

- Um die Inferenz von Grafikverarbeitungseinheiten (GPU) verwenden zu können, muss das Core-Gerät die folgenden Anforderungen erfüllen. GPU-Inferenz ist in Version 1.1.0 und höher dieser Komponente optional.
  - Eine Grafikverarbeitungseinheit (GPU), die CUDA unterstützt. Weitere Informationen finden Sie unter Überprüfen, ob Sie über eine CUDA-fähige GPU verfügen in der Dokumentation zum CUDA Toolkit.
  - cuDNN , CUDA und TensorRT auf dem Greengrass-Core-Gerät installiert.
    - Auf NVIDIA Jetson-Geräten wie Jetson Nano oder Jetson Xavier, cuDNN , CUDA und TensorRT sind mit NVIDIA installiert JetPack. Sie müssen keine Änderungen vornehmen. Diese Komponente unterstützt JetPack 4.4, JetPack 4.5, JetPack 4.5.1 und JetPack 4.6.1.

      > **Important**
      >
      > Sie müssen eine dieser Versionen von JetPack und keine andere Version installieren. Der Lookout for Vision-Service kompiliert Computer Vision-Modelle für diese JetPack Plattformen.

    - Gehen Sie auf x86-Geräten mit einer GPU mit der NVIDIA Ampere-Mikroarchitektur (oder der Rechenkapazität der GPU ist 8,0) wie folgt vor:
      - Installieren Sie cuDNN, indem Sie den Anweisungen im NVIDIA cuDNN-Installationshandbuch folgen.
      - Installieren Sie CUDA Version 11.2, indem Sie den Anweisungen im NVIDIA CUDA-Installationshandbuch für Linux folgen.
      - Installieren Sie TensorRT Version 8.2.0, indem Sie den Anweisungen in der NVIDIA TensorRT-Dokumentation folgen.
    - Gehen Sie auf x86-Geräten mit einer GPU, die vor Ampere über eine NVIDIA-Architektur verfügt (oder die Rechenkapazität der GPU unter 8,0 liegt), wie folgt vor:
      - Installieren Sie cuDNN, indem Sie den Anweisungen im NVIDIA cuDNN-Installationshandbuch folgen.
      - Installieren Sie CUDA Version 10.2, indem Sie den Anweisungen im NVIDIA CUDA-Installationshandbuch für Linux folgen.
      - Installieren Sie TensorRT Version 7.1.3 oder höher, aber früher als Version 8.0.0, indem Sie den Anweisungen in der NVIDIA TensorRT-Dokumentation folgen.
- Der Systembenutzer, der diese Komponente ausführt, muss Mitglied der Systemgruppe sein, die Zugriff auf die GPU auf dem Gerät hat. Der Name dieser Gruppe unterscheidet sich je nach

Betriebssystem. Konsultieren Sie die Dokumentation für Ihr Betriebssystem und Ihre GPU, um den Namen dieser Systemgruppe zu ermitteln.

Auf NVIDIA Jetson-Geräten lautet der Name dieser Gruppe beispielsweise , und Sie können den folgenden Befehl ausführen`video`, um dieser Gruppe einen Systembenutzer hinzuzufügen. Ersetzen Sie *`ggc_user`* durch den Namen des hinzuzufügenden Benutzers.

```
sudo usermod -aG video ggc_user
```

Abhängigkeiten

Diese Komponente hat keine Abhängigkeiten.

Konfiguration

Diese Komponente bietet die folgenden Konfigurationsparameter, die Sie anpassen können, wenn Sie die Komponente bereitstellen.

`Socket`

(Optional) Der Datei-Socket, in dem der Edge-Agent ausgeführt wird. Die Modellkomponenten von Lookout for Vision verwenden diesen Datei-Socket, um mit dem Edge-Agenten zu kommunizieren. Wenn Sie diesen Parameter ändern, müssen Sie denselben Wert angeben, wenn Sie Lookout for Vision-Modellkomponenten bereitstellen.

Standard: `unix:///tmp/aws.iot.lookoutvision.EdgeAgent.sock`

Lokale Protokolldatei

Diese Komponente verwendet die folgende Protokolldatei.

```
/greengrass/v2/logs/aws.iot.lookoutvision.EdgeAgent.log
```

So zeigen Sie die Protokolle dieser Komponente an

- Führen Sie den folgenden Befehl auf dem Core-Gerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie durch */greengrass/v2* den Pfad zum AWS IoT Greengrass Stammordner.

```
sudo tail -f /greengrass/v2/logs/aws.iot.lookoutvision.EdgeAgent.log
```

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in jeder Version der Komponente beschrieben.

Version	Änderungen
1.2.0	Allgemeine Fehlerbehebungen und Verbesserungen.
1.1.9	Allgemeine Fehlerbehebungen und Verbesserungen.
1.1.8	Allgemeine Fehlerbehebungen und Verbesserungen.
1.1.7	Neue Features • Installiert das `opencv-python-headless` Paket in der virtuellen Umgebung von Lookout for Vision Edge Agent. Fehlerbehebungen und Verbesserungen • Verbessert die Berechnung des Konfidenzwerts. • Ändert die Größe der Heatmap-Modellmaske auf die ursprüngliche Dateigröße. • Allgemeine Fehlerbehebungen und Verbesserungen.
1.1.6	Neue Features Dem `DetectAnomalies` Ergebnis wurden neue Werte hinzugefügt. • `anomaly_score` – Die Zahl zwischen 0,0 und 1,0, die angibt, wie ungewöhnlich ein Bild ist. • `anomaly_threshold` – Während des Modelltrainings festgelegter Schwellenwert, der die Grenze zwischen einem anomalen Bild und einem normalen Bild bestimmt. Allgemeine Fehlerbehebungen und Verbesserungen.

Version	Änderungen
1.1.4	Neue Features Unterstützung für OpenCV zur Größenanpassung von Images hinzugefügt, falls verfügbar. Edge-Agent verwendet Pillow, wenn OpenCV nicht verfügbar ist. Fehlerbehebungen und Verbesserungen Allgemeine Fehlerbehebungen und Verbesserungen.
1.1.3	Allgemeine Fehlerbehebungen und Verbesserungen.
1.1.1	Allgemeine Fehlerbehebungen und Verbesserungen.
1.1.0	Neue Features • Fügt Unterstützung für Bildsegmentierungsmodelle hinzu, die Anomalien in Bildern identifizieren. • Fügt Unterstützung für CPU-Inferenz hinzu, sodass Sie Lookout for Vision-Modelle auf Core-Geräten ohne GPU verwenden können. Fehlerbehebungen und Verbesserungen • Allgemeine Fehlerbehebungen und Verbesserungen.
1.0.0	Diese Version der Komponente von Lookout for Vision Edge Agent erfordert eine andere Version von Python als Version 0.1.x. Wenn Sie ein Upgrade von v0.1.x auf v1.x durchführen möchten, müssen Sie die Python-Installation auf dem Core-Gerät aktualisieren. Fehlerbehebungen und Verbesserungen • Allgemeine Fehlerbehebungen und Verbesserungen.
0.1.37	Allgemeine Fehlerbehebungen und Verbesserungen.
0.1.36	Erste Version

SageMaker Edge-Manager

> **Important**
>
> SageMaker Edge Manager wird am 26. April 2024 eingestellt. Weitere Informationen zur weiteren Bereitstellung Ihrer Modelle auf Edge-Geräten finden Sie unter Ende der Nutzungsdauer von SageMaker Edge Manager.

Die Amazon SageMaker Edge Manager-Komponente (`aws.greengrass.SageMakerEdgeManager`) installiert die SageMaker Edge Manager-Agent-Binärdatei.

SageMaker Edge Manager bietet Modellmanagement für Edge-Geräte, sodass Sie Modelle für maschinelles Lernen auf Flotten von Edge-Geräten optimieren, sichern, überwachen und verwalten können. Die SageMaker Edge Manager-Komponente installiert und verwaltet den Lebenszyklus des SageMaker Edge Manager-Agenten auf Ihrem Kerngerät. Sie können SageMaker Edge Manager auch verwenden, um SageMaker NEO-kompilierte Modelle als Modellkomponenten auf Greengrass-Kerngeräten zu verpacken und zu verwenden. Weitere Informationen zur Verwendung SageMaker des Edge Manager-Agenten auf Ihrem Kerngerät finden Sie unter. Verwenden von Amazon SageMaker Edge Manager auf Greengrass-Core-Geräten

SageMaker Die Edge Manager-Komponente v1.3.x installiert die Edge Manager-Agent-Binärdatei v1.20220822.836f3023. Weitere Informationen zu den Binärversionen des Edge Manager-Agents finden Sie unter Edge Manager Agent.

> **Note**
>
> Die SageMaker Edge Manager-Komponente ist nur in den folgenden Versionen verfügbar AWS-Regionen:
>
> - US East (Ohio)
> - USA Ost (Nord-Virginia)
> - USA West (Oregon)
> - EU (Frankfurt)
> - EU (Irland)
> - Asien-Pazifik (Tokio)

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 1.3.x
- 1.2.x
- 1.1.x
- 1.0.x

Typ

Diese Komponente ist eine generische Komponente (`aws.greengrass.generic`). Der Greengrass-Kern führt die Lebenszyklusskripte der Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf Kerngeräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Ein Greengrass-Core-Gerät, das auf Amazon Linux 2, einer Debian-basierten Linux-Plattform (x86_64 oder Armv8) oder Windows (x86_64) läuft. Falls Sie noch keines haben, beachten Sie die Informationen unter Tutorial: Erste Schritte mit AWS IoT Greengrass V2.
- Python 3.6 oder höher, auch `pip` für Ihre Version von Python, die auf Ihrem Kerngerät installiert ist.
- Die Greengrass-Geräterolle wurde wie folgt konfiguriert:
  - Eine Vertrauensbeziehung, die es ermöglicht `credentials.iot.amazonaws.com` und `sagemaker.amazonaws.com` die Übernahme der Rolle ermöglicht, wie im folgenden Beispiel für eine IAM-Richtlinie dargestellt.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
       },
      "Action": "sts:AssumeRole"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sagemaker.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

  - Die von AmazonSageMakerEdgeDeviceFleetPolicyIAM verwaltete Richtlinie.
  - Die `s3:PutObject` Aktion, wie im folgenden Beispiel für eine IAM-Richtlinie dargestellt.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "*"
```

```
      ],
      "Effect": "Allow"
    }
  ]
}
```

- Ein Amazon S3 S3-Bucket, der im selben AWS-Konto und AWS-Region wie Ihr Greengrass-Core-Gerät erstellt wurde. SageMaker Edge Manager benötigt einen S3-Bucket, um eine Edge-Geräteflotte zu erstellen und Beispieldaten aus laufenden Inferenzen auf Ihrem Gerät zu speichern. Informationen zum Erstellen von S3-Buckets finden Sie unter Erste Schritte mit Amazon S3.
- Eine SageMaker Edge-Geräteflotte, die denselben AWS IoT Rollenalias wie Ihr Greengrass-Core-Gerät verwendet. Weitere Informationen finden Sie unter Erstellen Sie eine Flotte von Edge-Geräten.
- Ihr Greengrass Core-Gerät ist als Edge-Gerät in Ihrer SageMaker Edge-Geräteflotte registriert. Der Name des Edge-Geräts muss mit dem AWS IoT Dingnamen für Ihr Core-Gerät übereinstimmen. Weitere Informationen finden Sie unter Registrieren Sie Ihr Greengrass Core-Gerät.

Endpunkte und Anschlüsse

Diese Komponente muss in der Lage sein, ausgehende Anfragen an die folgenden Endpunkte und Ports auszuführen, zusätzlich zu den Endpunkten und Ports, die für den Basisbetrieb erforderlich sind. Weitere Informationen finden Sie unter Zulassen von Gerätedatenverkehr über einen Proxy oder eine Firewall.

Endpunkt	Port	Erforderlich	Beschreibung
`edge.sagemaker.`*`region`*`.amazonaws.com`	443	Ja	Überprüfen Sie den Status der Geräteregistrierung und senden Sie Messdaten an. SageMaker

Endpunkt	Port	Erforderlich	Beschreibung
`*.s3.amazonaws.com`	443	Ja	Laden Sie die Erfassungsdaten in den von Ihnen angegebenen S3-Bucket hoch. Sie können es * durch den Namen jedes Buckets ersetzen, in den Sie Daten hochladen.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und all ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitstellen zu können. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente sowie die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste der Abhängigkeiten.

1.3.5 and 1.3.6

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 1.3.5 und 1.3.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.13.0	Weich
Token-Austauschdienst	>=0.0.0	Hart

1.3.4

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.3.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.12.0	Weich
Token-Austauschdienst	>=0.0.0	Hart

1.3.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.3.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.11.0	Weich
Token-Austauschdienst	>=0.0.0	Hart

1.3.2

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.3.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.10.0	Weich
Token-Austauschdienst	>=0.0.0	Hart

1.3.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.3.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.9.0	Weich
Token-Austauschdienst	>=0.0.0	Hart

1.1.1 - 1.3.0

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 1.1.1 bis 1.3.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.8.0	Weich
Token-Austauschdienst	>=0.0.0	Hart

1.1.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.1.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.6.0	Weich
Token-Austauschdienst	>=0.0.0	Hart

1.0.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.0.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.5.0	Weich
Token-Austauschdienst	>=0.0.0	Hart

1.0.1 and 1.0.2

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 1.0.1 und 1.0.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.4.0	Weich
Token-Austauschdienst	>=0.0.0	Hart

1.0.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.0.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.3.0	Weich
Token-Austauschdienst	>=0.0.0	Hart

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zu den Komponentenrezepten.

Konfiguration

Diese Komponente stellt die folgenden Konfigurationsparameter bereit, die Sie bei der Bereitstellung der Komponente anpassen können.

> **Note**
>
> In diesem Abschnitt werden die Konfigurationsparameter beschrieben, die Sie in der Komponente festlegen. Weitere Informationen zur entsprechenden SageMaker Edge Manager-Konfiguration finden Sie unter Edge Manager Agent im Amazon SageMaker Developer Guide.

`DeviceFleetName`

Der Name der SageMaker Edge Manager-Geräteflotte, die Ihr Greengrass-Core-Gerät enthält.

Sie müssen im Konfigurationsupdate einen Wert für diesen Parameter angeben, wenn Sie diese Komponente bereitstellen.

`BucketName`

Der Name des S3-Buckets, in den Sie erfasste Inferenzdaten hochladen. Der Bucket-Name muss die Zeichenfolge `sagemaker` enthalten.

Wenn Sie `CaptureDataDestination` auf `Cloud` oder wenn Sie `CaptureDataPeriodicUpload` auf setzen`true`, müssen Sie bei der Bereitstellung dieser Komponente im Konfigurationsupdate einen Wert für diesen Parameter angeben.

> **Note**
>
> Daten erfassen ist eine SageMaker Funktion, mit der Sie Inferenzeingaben, Inferenzergebnisse und zusätzliche Inferenzdaten für future Analysen in einen S3-Bucket oder ein lokales Verzeichnis hochladen. Weitere Informationen zur Verwendung von Erfassungsdaten mit SageMaker Edge Manager finden Sie unter Modell verwalten im Amazon SageMaker Developer Guide.

`CaptureDataBatchSize`

(Optional) Die Größe eines Stapels von Anfragen zur Erfassung von Daten, die der Agent bearbeitet. Dieser Wert muss kleiner als die Puffergröße sein, die Sie in angeben`CaptureDataBufferSize`. Wir empfehlen, die Hälfte der Puffergröße nicht zu überschreiten.

Der Agent verarbeitet einen Anforderungsstapel, wenn die Anzahl der Anfragen im Puffer der `CaptureDataBatchSize` Anzahl entspricht, oder wenn das `CaptureDataPushPeriodSeconds` Intervall abgelaufen ist, je nachdem, was zuerst eintritt.

Standard: `10`

`CaptureDataBufferSize`

(Optional) Die maximale Anzahl von Erfassungsdatenanforderungen, die im Puffer gespeichert sind.

Standard: `30`

`CaptureDataDestination`

(Optional) Das Ziel, an dem Sie die erfassten Daten speichern. Dieser Parameter kann die folgenden Werte haben:

- `Cloud`— Lädt erfasste Daten in den von Ihnen angegebenen S3-Bucket hoch. `BucketName`
- `Disk`— Schreibt die erfassten Daten in das Arbeitsverzeichnis der Komponente.

Wenn Sie dies angeben`Disk`, können Sie die erfassten Daten auch regelmäßig in Ihren S3-Bucket hochladen, indem Sie `CaptureDataPeriodicUpload` auf `true` einstellen.

Standard: `Cloud`

`CaptureDataPeriodicUpload`

(Optional) Zeichenkettenwert, der angibt, ob die erfassten Daten regelmäßig hochgeladen werden sollen. Unterstützte Werte sind `true` und `false`.

Setzen Sie diesen Parameter auf, `true` wenn Sie `CaptureDataDestination` auf festlegen und Sie möchten`Disk`, dass der Agent die erfassten Daten regelmäßig in Ihren S3-Bucket hochlädt.

Standard: `false`

`CaptureDataPeriodicUploadPeriodSeconds`

(Optional) Das Intervall in Sekunden, in dem der SageMaker Edge Manager-Agent die erfassten Daten in den S3-Bucket hochlädt. Verwenden Sie diesen Parameter, wenn Sie `CaptureDataPeriodicUpload` auf `true` einstellen.

Standard: 8

`CaptureDataPushPeriodSeconds`

(Optional) Das Intervall in Sekunden, in dem der SageMaker Edge Manager-Agent einen Stapel von Erfassungsdatenanforderungen aus dem Puffer verarbeitet.

Der Agent verarbeitet einen Anforderungsstapel, wenn die Anzahl der Anfragen im Puffer der `CaptureDataBatchSize` Anzahl entspricht, oder wenn das `CaptureDataPushPeriodSeconds` Intervall abgelaufen ist, je nachdem, was zuerst eintritt.

Standard: `4`

`CaptureDataBase64EmbedLimit`

(Optional) Die maximale Größe der erfassten Daten in Byte, die der SageMaker Edge Manager-Agent hochlädt.

Standard: `3072`

`FolderPrefix`

(Optional) Der Name des Ordners, in den der Agent die erfassten Daten schreibt. Wenn Sie `CaptureDataDestination` auf festlegen`Disk`, erstellt der Agent den Ordner in dem Verzeichnis, das von angegeben ist`CaptureDataDiskPath`. Wenn Sie `CaptureDataDestination` auf `Cloud` oder wenn Sie `CaptureDataPeriodicUpload` auf festlegen`true`, erstellt der Agent den Ordner in Ihrem S3-Bucket.

Standard: `sme-capture`

`CaptureDataDiskPath`

Diese Funktion ist in Version 1.1.0 und späteren Versionen der SageMaker Edge Manager-Komponente verfügbar.

(Optional) Der Pfad zu dem Ordner, in dem der Agent den Ordner mit den erfassten Daten erstellt. Wenn Sie `CaptureDataDestination` auf festlegen`Disk`, erstellt der Agent den Ordner mit den erfassten Daten in diesem Verzeichnis. Wenn Sie diesen Wert nicht angeben, erstellt der Agent den Ordner mit den erfassten Daten im Arbeitsverzeichnis der Komponente. Verwenden Sie den `FolderPrefix` Parameter, um den Namen des Ordners für erfasste Daten anzugeben.

Standard: *`/greengrass/v2`*`/work/aws.greengrass.SageMakerEdgeManager/capture`

`LocalDataRootPath`

Diese Funktion ist in Version 1.2.0 und späteren Versionen der SageMaker Edge Manager-Komponente verfügbar.

(Optional) Der Pfad, in dem diese Komponente die folgenden Daten auf dem Kerngerät speichert:

- Die lokale Datenbank für Laufzeitdaten, wenn Sie `DbEnable` auf einstellen`true`.
- SageMaker Neo-kompilierte Modelle, die diese Komponente automatisch herunterlädt, wenn Sie `DeploymentEnable` auf `true` einstellen.

Standard: */greengrass/v2*`/work/aws.greengrass.SageMakerEdgeManager`

`DbEnable`

(Optional) Sie können diese Komponente aktivieren, um Laufzeitdaten in einer lokalen Datenbank zu speichern, um die Daten für den Fall zu schützen, dass die Komponente ausfällt oder das Gerät ausfällt.

Diese Datenbank benötigt 5 MB Speicherplatz im Dateisystem des Kerngeräts.

Standard: `false`

`DeploymentEnable`

Diese Funktion ist in Version 1.2.0 und späteren Versionen der SageMaker Edge Manager-Komponente verfügbar.

(Optional) Sie können diese Komponente aktivieren, um automatisch SageMaker NEO-kompilierte Modelle abzurufen, von denen Sie auf Amazon S3 hochladen. Nachdem Sie ein neues Modell auf Amazon S3 hochgeladen haben, verwenden Sie SageMaker Studio oder die SageMaker API, um das neue Modell auf diesem Kerngerät bereitzustellen. Wenn Sie diese Funktion aktivieren, können Sie neue Modelle auf Kerngeräten bereitstellen, ohne eine AWS IoT Greengrass Bereitstellung erstellen zu müssen.

> **Important**
>
> Um diese Funktion verwenden zu können, müssen Sie `DbEnable` auf `true` einstellen. Diese Funktion verwendet die lokale Datenbank, um Modelle zu verfolgen, die sie aus der AWS Cloud abruft.

Standard: `false`

`DeploymentPollInterval`

Diese Funktion ist in Version 1.2.0 und späteren Versionen der SageMaker Edge Manager-Komponente verfügbar.

(Optional) Der Zeitraum (in Minuten), zwischen dem diese Komponente nach neuen Modellen sucht, die heruntergeladen werden können. Diese Option gilt, wenn Sie `DeploymentEnable` auf einstellen`true`.

Standard: `1440` (1 Tag)

`DLRBackendOptions`

Diese Funktion ist in Version 1.2.0 und späteren Versionen der SageMaker Edge Manager-Komponente verfügbar.

(Optional) Die DLR-Laufzeit-Flags, die in der von dieser Komponente verwendeten DLR-Laufzeit gesetzt werden sollen. Sie können das folgende Flag setzen:

- `TVM_TENSORRT_CACHE_DIR`— Aktiviert das Zwischenspeichern von TensorRT-Modellen. Geben Sie einen absoluten Pfad zu einem vorhandenen Ordner mit Lese-/Schreibberechtigungen an.
- `TVM_TENSORRT_CACHE_DISK_SIZE_MB`— Weist die Obergrenze des TensorRT-Modell-Cache-Ordners zu. Wenn die Verzeichnisgröße diese Grenze überschreitet, werden die zwischengespeicherten Engines, die am wenigsten verwendet werden, gelöscht. Der Standardwert ist 512 MB.

Sie können diesen Parameter beispielsweise auf den folgenden Wert setzen, um das Zwischenspeichern von TensorRT-Modellen zu aktivieren und die Cachegröße auf 800 MB zu begrenzen.

```
TVM_TENSORRT_CACHE_DIR=/data/secured_folder/trt/cache;
 TVM_TENSORRT_CACHE_DISK_SIZE_MB=800
```

`SagemakerEdgeLogVerbose`

(Optional) Zeichenkettenwert, der angibt, ob die Debug-Protokollierung aktiviert werden soll. Unterstützte Werte sind `true` und `false`.

Standard: `false`

`UnixSocketName`

(Optional) Der Speicherort des SageMaker Edge Manager-Socket-Dateideskriptors auf dem Kerngerät.

Standard: `/tmp/aws.greengrass.SageMakerEdgeManager.sock`

Example Beispiel: Update zur Zusammenführung von Konfigurationen

Die folgende Beispielkonfiguration gibt an, dass das Kerngerät Teil von ist *MyEdgeDeviceFleet*und dass der Agent die Erfassungsdaten sowohl auf das Gerät als auch in einen S3-Bucket schreibt. Diese Konfiguration ermöglicht auch die Debug-Protokollierung.

```
{
    "DeviceFleetName": "MyEdgeDeviceFleet",
    "BucketName": "DOC-EXAMPLE-BUCKET",
    "CaptureDataDestination": "Disk",
    "CaptureDataPeriodicUpload": "true",
    "SagemakerEdgeLogVerbose": "true"
}
```

Lokale Protokolldatei

Diese Komponente verwendet die folgende Protokolldatei.

Linux

```
/greengrass/v2/logs/aws.greengrass.SageMakerEdgeManager.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.SageMakerEdgeManager.log
```

Um die Protokolle dieser Komponente einzusehen

- Führen Sie den folgenden Befehl auf dem Kerngerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

  Linux

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

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\aws.greengrass.SageMakerEdgeManager.log -Tail
 10 -Wait
```

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
1.3.6	Die Version wurde für die Version Greengrass Nucleus 2.12.5 aktualisiert.
1.3.5	Die Version wurde für die Version 2.12.0 von Greengrass Nucleus aktualisiert.
1.3.4	Die Version wurde für die Version 2.11.0 von Greengrass Nucleus aktualisiert.
1.3.3	Die Version wurde für die Version 2.10.0 von Greengrass Nucleus aktualisiert.
1.3.2	Die Version wurde für die Version 2.9.0 von Greengrass Nucleus aktualisiert.
1.3.1	Die Version wurde für die Version 2.8.0 von Greengrass Nucleus aktualisiert.
1.3.0	Neue Features • Fügt Unterstützung für die Verwaltung der TensorRT-Cache-Fes tplattengröße hinzu. • Fügt dem BackendOptions DLR-Parameter das optionale `TVM_TENSO RRT_CACHE_DISK_SIZE_MB` Flag hinzu, um die Größenbes chränkung für zwischengespeicherte Modelle auf der Festplatte festzulegen. Verbesserungen • Bietet eine verbesserte Parallelität bei der Vorhersage. Dies trägt zu einer besseren Nutzung von Gerätebeschleuniger-Engines wie GPUs bei.

Version	Änderungen
1.2.0	Neue Features • Fügt Unterstützung für diese Komponente hinzu, um SageMaker NEO-kompilierte Modelle, die Sie auf Amazon S3 hochladen, automatisch abzurufen. Wenn Sie diese Funktion aktivieren, können Sie neue Modelle auf Kerngeräten bereitstellen, ohne eine AWS IoT Greengrass Bereitstellung erstellen zu müssen. • Integriert die Unterstützung für eine Backup-Datenbank, die diese Komponente zur Aufbewahrung von Laufzeitdaten verwendet, falls die Komponente ausfällt oder das Gerät ausfällt. • Integriert die Unterstützung für die Konfiguration von DLR-Runtime-Flags bei der Konfiguration dieser Komponente.
1.1.1	Die Version wurde für die Version 2.7.0 von Greengrass Nucleus aktualisiert.
1.1.0	Neue Features • Fügt Unterstützung für Greengrass-Core-Geräte hinzu, auf denen Amazon Linux 2 ausgeführt wird. • Fügt den neuen `CaptureDataDiskPath` Konfigurationsparameter hinzu. Sie können diesen Parameter verwenden, um den Pfad des Ordners mit erfassten Daten auf Ihrem Gerät anzugeben. Fehlerkorrekturen und Verbesserungen • Die Version wurde für die Version 2.5.0 von Greengrass Nucleus aktualisiert.
1.0.3	Die Version wurde für die Version 2.4.0 von Greengrass Nucleus aktualisiert.
1.0.2	Fehlerkorrekturen und Verbesserungen Aktualisiert das Installationsskript im Komponentenlebenszyklus. Auf Ihren Kerngeräten muss jetzt Python 3.6 oder höher, auch `pip` für Ihre Version von Python, auf dem Gerät installiert sein, bevor Sie diese Komponente bereitstellen.
1.0.1	Die Version wurde für die Version 2.3.0 von Greengrass Nucleus aktualisiert.

Version	Änderungen
1.0.0	Erste Version

DLR-Bildklassifizierung

Die DLR-Komponente zur Bildklassifizierung (`aws.greengrass.DLRImageClassification`) enthält Beispiel-Inferenzcode für die Durchführung von Inferenzen zur Bildklassifizierung mithilfe von Deep Learning Runtime - und Resnet-50-Modellen. Diese Komponente verwendet die Variante Modellspeicher für die DLR-Bildklassifizierung und die DLR-Laufzeit Komponenten als Abhängigkeiten, um DLR und die Beispielmodelle herunterzuladen.

Um diese Inferenzkomponente mit einem speziell trainierten DLR-Modell zu verwenden, erstellen Sie eine benutzerdefinierte Version der abhängigen Modellspeicherkomponente. Um Ihren eigenen benutzerdefinierten Inferenzcode zu verwenden, können Sie das Rezept dieser Komponente als Vorlage verwenden, um eine benutzerdefinierte Inferenzkomponente zu erstellen.

Themen

- Versionen
- Typ
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Konfiguration
- Lokale Protokolldatei
- Änderungsprotokoll

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.1.x
- 2.0.x

Typ

Diese Komponente ist eine generische Komponente (`aws.greengrass.generic`). Der Greengrass-Kern führt die Lebenszyklusskripte der Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf Kerngeräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Auf Greengrass-Core-Geräten, auf denen Amazon Linux 2 oder Ubuntu 18.04 ausgeführt wird, ist die GNU C Library (Glibc) Version 2.27 oder höher auf dem Gerät installiert.
- Auf ARMv7L-Geräten wie Raspberry Pi sind Abhängigkeiten für OpenCV-Python auf dem Gerät installiert. Führen Sie den folgenden Befehl aus, um die Abhängigkeiten zu installieren.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- Raspberry Pi-Geräte, auf denen Raspberry Pi OS Bullseye ausgeführt wird, müssen die folgenden Anforderungen erfüllen:
  - NumPy 1.22.4 oder höher auf dem Gerät installiert. Raspberry Pi OS Bullseye enthält eine frühere Version von NumPy, sodass Sie den folgenden Befehl ausführen können, um das Gerät zu aktualisieren NumPy .

    ```
    pip3 install --upgrade numpy
    ```

  - Der ältere Kamerastack ist auf dem Gerät aktiviert. Raspberry Pi OS Bullseye enthält einen neuen Kamerastack, der standardmäßig aktiviert und nicht kompatibel ist. Sie müssen also den älteren Kamerastack aktivieren.

Um den Legacy-Kamerastack zu aktivieren

1. Führen Sie den folgenden Befehl aus, um das Raspberry Pi-Konfigurationstool zu öffnen.

   ```
   sudo raspi-config
   ```

2. Wählen Sie Schnittstellenoptionen.
3. Wählen Sie Legacy-Kamera aus, um den Legacy-Kamerastack zu aktivieren.
4. Starten Sie den Raspberry Pi neu.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt sie AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und all ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitstellen zu können. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente sowie die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste der Abhängigkeiten.

2.1.13 and 2.1.14

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.1.13 und 2.1.14 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.13.0	Weich
Modellspeicher für die DLR-Bildklassifizierung	~2.1.0	Hart
DLR	~1.6.0	Hart

2.1.12

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.12 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.12.0	Weich
Modellspeicher für die DLR-Bildklassifizierung	~2.1.0	Hart
DLR	~1.6.0	Hart

2.1.11

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.11 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.11.0	Weich
Modellspeicher für die DLR-Bildklassifizierung	~2.1.0	Hart
DLR	~1.6.0	Hart

2.1.10

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.10 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.10.0	Weich

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Modellspeicher für die DLR-Bildklassifizierung	~2.1.0	Hart
DLR	~1.6.0	Hart

2.1.9

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.9 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.9.0	Weich
Modellspeicher für die DLR-Bildklassifizierung	~2.1.0	Hart
DLR	~1.6.0	Hart

2.1.8

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.8 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.8.0	Weich
Modellspeicher für die DLR-Bildklassifizierung	~2.1.0	Hart
DLR	~1.6.0	Hart

2.1.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.7.0	Weich
Modellspeicher für die DLR-Bildklassifizierung	~2.1.0	Hart
DLR	~1.6.0	Hart

2.1.6

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.6.0	Weich
Modellspeicher für die DLR-Bildklassifizierung	~2.1.0	Hart
DLR	~1.6.0	Hart

2.1.4 - 2.1.5

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.1.4 bis 2.1.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.5.0	Weich
Modellspeicher für die DLR-Bildklassifizierung	~2.1.0	Hart
DLR	~1.6.0	Hart

2.1.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.4.0	Weich
Modellspeicher für die DLR-Bildklassifizierung	~2.1.0	Hart
DLR	~1.6.0	Hart

2.1.2

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.3.0	Weich
Modellspeicher für die DLR-Bildklassifizierung	~2.1.0	Hart
DLR	~1.6.0	Hart

2.1.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.2.0	Weich
Modellspeicher für die DLR-Bildklassifizierung	~2.1.0	Hart
DLR	~1.6.0	Hart

2.0.x

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.x dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	~2.0.0	Weich
Modellspeicher für die DLR-Bildklassifizierung	~2.0.0	Hart
DLR	~1.3.0	Weich

Konfiguration

Diese Komponente bietet die folgenden Konfigurationsparameter, die Sie bei der Bereitstellung der Komponente anpassen können.

2.1.x

`accessControl`

(Optional) Das Objekt, das die Autorisierungsrichtlinie enthält, die es der Komponente ermöglicht, Nachrichten unter dem Standardthema für Benachrichtigungen zu veröffentlichen.

Standard:

```
{
   "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.DLRImageClassification:mqttproxy:1": {
         "policyDescription": "Allows access to publish via topic ml/dlr/image-
classification.",
         "operations": [
            "aws.greengrass#PublishToIoTCore"
         ],
         "resources": [
            "ml/dlr/image-classification"
         ]
      }
   }
}
```

`PublishResultsOnTopic`

(Optional) Das Thema, zu dem Sie die Inferenzergebnisse veröffentlichen möchten. Wenn Sie diesen Wert ändern, müssen Sie auch den Wert von `resources` im `accessControl` Parameter so ändern, dass er Ihrem benutzerdefinierten Themennamen entspricht.

Standard: `ml/dlr/image-classification`

`Accelerator`

Der Beschleuniger, den Sie verwenden möchten. Unterstützte Werte sind `cpu` und `gpu`.

Die Beispielmodelle in der abhängigen Modellkomponente unterstützen nur CPU-Beschleunigung. Um die GPU-Beschleunigung mit einem anderen benutzerdefinierten Modell zu verwenden, erstellen Sie eine benutzerdefinierte Modellkomponente, um die Komponente des öffentlichen Modells zu überschreiben.

Standard: `cpu`

`ImageDirectory`

(Optional) Der Pfad des Ordners auf dem Gerät, in dem Inferenzkomponenten Bilder lesen. Sie können diesen Wert an einen beliebigen Ort auf Ihrem Gerät ändern, auf den Sie Lese-/Schreibzugriff haben.

Standard: *`/greengrass/v2`*`/packages/artifacts-unarchived/`*`component-name`*`/image_classification/sample_images/`

> **Note**
>
> Wenn Sie den Wert auf setzen `UseCamera`true, wird dieser Konfigurationsparameter ignoriert.

`ImageName`

(Optional) Der Name des Bilds, das die Inferenzkomponente als Eingabe für eine Make-Vorhersage verwendet. Die Komponente sucht in dem unter angegebenen Ordner nach dem Bild. `ImageDirectory` Standardmäßig verwendet die Komponente das Beispielbild im Standard-Bildverzeichnis. AWS IoT Greengrass unterstützt die folgenden Bildformate: `jpeg`jpg,png, und`npy`.

Standard: `cat.jpeg`

> **Note**
>
> Wenn Sie den Wert `UseCamera` auf setzen`true`, wird dieser Konfigurationsparameter ignoriert.

`InferenceInterval`

(Optional) Die Zeit in Sekunden zwischen den einzelnen Vorhersagen des Inferenzcodes. Der Beispiel-Inferenzcode wird unbegrenzt ausgeführt und wiederholt seine Vorhersagen im angegebenen Zeitintervall. Sie können dieses Intervall beispielsweise auf ein kürzeres Intervall ändern, wenn Sie mit einer Kamera aufgenommene Bilder für Vorhersagen in Echtzeit verwenden möchten.

Standard: `3600`

`ModelResourceKey`

(Optional) Die Modelle, die in der abhängigen öffentlichen Modellkomponente verwendet werden. Ändern Sie diesen Parameter nur, wenn Sie die Komponente des öffentlichen Modells durch eine benutzerdefinierte Komponente überschreiben.

Standard:

```
{
    "armv7l": "DLR-resnet50-armv7l-cpu-ImageClassification",
    "aarch64": "DLR-resnet50-aarch64-cpu-ImageClassification",
    "x86_64": "DLR-resnet50-x86_64-cpu-ImageClassification",
    "windows": "DLR-resnet50-win-cpu-ImageClassification"
}
```

`UseCamera`

(Optional) Zeichenkettenwert, der definiert, ob Bilder von einer Kamera verwendet werden sollen, die mit dem Greengrass-Core-Gerät verbunden ist. Unterstützte Werte sind `true` und `false`.

Wenn Sie diesen Wert auf setzen`true`, greift der Beispiel-Inferenzcode auf die Kamera auf Ihrem Gerät zu und führt die Inferenz lokal für das aufgenommene Bild aus. Die Werte der `ImageDirectory` Parameter `ImageName` und werden ignoriert. Stellen Sie sicher, dass der

Benutzer, der diese Komponente ausführt, Lese-/Schreibzugriff auf den Speicherort hat, an dem die Kamera die aufgenommenen Bilder speichert.

Standard: `false`

> **Note**
>
> Wenn Sie sich das Rezept für diese Komponente ansehen, erscheint der `UseCamera` Konfigurationsparameter nicht in der Standardkonfiguration. Sie können den Wert dieses Parameters jedoch in einem Update zur Zusammenführung der Konfiguration ändern, wenn Sie die Komponente bereitstellen.
> Wenn Sie `UseCamera` auf setzen`true`, müssen Sie auch einen Symlink erstellen, damit die Inferenzkomponente über die virtuelle Umgebung, die von der Runtime-Komponente erstellt wurde, auf Ihre Kamera zugreifen kann. Weitere Hinweise zur Verwendung einer Kamera mit den Beispiel-Inferenzkomponenten finden Sie unter. Komponentenkonfigurationen aktualisieren

2.0.x

`MLRootPath`

(Optional) Der Pfad des Ordners auf Linux-Kerngeräten, in dem Inferenzkomponenten Bilder lesen und Inferenzergebnisse schreiben. Sie können diesen Wert in einen beliebigen Speicherort auf Ihrem Gerät ändern, auf den der Benutzer, der diese Komponente ausführt, Lese-/Schreibzugriff hat.

Standard: `/greengrass/v2/work/variant.DLR/greengrass_ml`

Standard: `/greengrass/v2/work/variant.TensorFlowLite/greengrass_ml`

`Accelerator`

Der Beschleuniger, den Sie verwenden möchten. Unterstützte Werte sind `cpu` und `gpu`.

Die Beispielmodelle in der abhängigen Modellkomponente unterstützen nur CPU-Beschleunigung. Um die GPU-Beschleunigung mit einem anderen benutzerdefinierten Modell zu verwenden, erstellen Sie eine benutzerdefinierte Modellkomponente, um die Komponente des öffentlichen Modells zu überschreiben.

Standard: `cpu`

`ImageName`

(Optional) Der Name des Bilds, das die Inferenzkomponente als Eingabe für eine Make-Vorhersage verwendet. Die Komponente sucht in dem unter angegebenen Ordner nach dem Bild. `ImageDirectory` Der Standardspeicherort ist*`MLRootPath`*`/images`. AWS IoT Greengrass unterstützt die folgenden Bildformate: `jpeg`jpg,png, undnpy.

Standard: `cat.jpeg`

`InferenceInterval`

(Optional) Die Zeit in Sekunden zwischen den einzelnen Vorhersagen des Inferenzcodes. Der Beispiel-Inferenzcode wird unbegrenzt ausgeführt und wiederholt seine Vorhersagen im angegebenen Zeitintervall. Sie können dieses Intervall beispielsweise auf ein kürzeres Intervall ändern, wenn Sie mit einer Kamera aufgenommene Bilder für Vorhersagen in Echtzeit verwenden möchten.

Standard: `3600`

`ModelResourceKey`

(Optional) Die Modelle, die in der abhängigen öffentlichen Modellkomponente verwendet werden. Ändern Sie diesen Parameter nur, wenn Sie die Komponente des öffentlichen Modells durch eine benutzerdefinierte Komponente überschreiben.

Standard:

```
armv7l: "DLR-resnet50-armv7l-cpu-ImageClassification"
x86_64: "DLR-resnet50-x86_64-cpu-ImageClassification"
```

Lokale Protokolldatei

Diese Komponente verwendet die folgende Protokolldatei.

Linux

```
/greengrass/v2/logs/aws.greengrass.DLRImageClassification.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.DLRImageClassification.log
```

Um die Protokolle dieser Komponente einzusehen

- Führen Sie den folgenden Befehl auf dem Kerngerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

  Linux

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

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.DLRImageClassification.log -
  Tail 10 -Wait
  ```

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
2.1.14	Die Version wurde für die Version Greengrass Nucleus 2.12.5 aktualisiert.
2.1.13	Die Version wurde für die Version 2.12.0 von Greengrass Nucleus aktualisiert.
2.1.12	Die Version wurde für die Version 2.11.0 von Greengrass Nucleus aktualisiert.
2.1.11	Die Version wurde für die Version 2.10.0 von Greengrass Nucleus aktualisiert.
2.1.10	Die Version wurde für die Version 2.9.0 von Greengrass Nucleus aktualisiert.
2.1.9	Die Version wurde für die Version 2.8.0 von Greengrass Nucleus aktualisiert.
2.1.8	Die Version wurde für die Version 2.7.0 von Greengrass Nucleus aktualisiert.
2.1.7	Die Version wurde für die Version 2.6.0 von Greengrass Nucleus aktualisiert.
2.1.6	Die Version wurde für die Version 2.5.0 von Greengrass Nucleus aktualisiert.

Version	Änderungen
2.1.5	Die Komponente wurde insgesamt veröffentlicht. AWS-Regionen
2.1.4	Die Version wurde für die Version 2.4.0 von Greengrass Nucleus aktualisiert. Diese Version ist in Europa (London) nicht verfügbar (`eu-west-2`).
2.1.3	Die Version wurde für die Version 2.3.0 von Greengrass Nucleus aktualisiert.
2.1.2	Die Version wurde für die Version 2.2.0 von Greengrass Nucleus aktualisiert.

Version	Änderungen
2.1.1	Neue Features • Verwenden Sie Deep Learning Runtime v1.6.0. • Unterstützung für die Klassifizierung von Beispielbildern auf Armv8-Plattformen (AArch64) hinzugefügt. Dies erweitert die Unterstützung für maschinelles Lernen für Greengrass-Kerngeräte, auf denen NVIDIA Jetson ausgeführt wird, wie z. B. den Jetson Nano. • Aktivieren Sie die Kameraintegration für Probeninferenz. Verwenden Sie den neuen `UseCamera` Konfigurationsparameter, um den Beispiel-Inferenzcode für den Zugriff auf die Kamera auf Ihrem Greengrass-Core-Gerät zu aktivieren und die Inferenz lokal auf dem aufgenommenen Bild auszuführen. • Fügen Sie Unterstützung für die Veröffentlichung von Inferenzergebnissen zur hinzu. AWS Cloud Verwenden Sie den neuen `PublishResultsOnTopic` Konfigurationsparameter, um das Thema anzugeben, zu dem Sie Ergebnisse veröffentlichen möchten. • Fügen Sie den neuen `ImageDirectory` Konfigurationsparameter hinzu, mit dem Sie ein benutzerdefiniertes Verzeichnis für das Bild angeben können, für das Sie Inferenzen durchführen möchten. Fehlerkorrekturen und Verbesserungen • Schreiben Sie die Inferenzergebnisse in die Protokolldatei der Komponente statt in eine separate Inferenzdatei. • Verwenden Sie das AWS IoT Greengrass Core-Software-Logging-Modul, um die Komponentenausgabe zu protokollieren. • Verwenden Sie das AWS IoT Device SDK , um die Komponentenkonfiguration zu lesen und Konfigurationsänderungen anzuwenden.
2.0.4	Erste Version

DLR-Objekterkennung

Die DLR-Objekterkennungskomponente (`aws.greengrass.DLRObjectDetection`) enthält Beispiel-Inferenzcode für die Durchführung von Inferenzen zur Objekterkennung mithilfe von Deep Learning Runtime sowie vortrainierte Beispielmodelle. Diese Komponente verwendet die Variante

Modellspeicher für DLR-Objekterkennung und die DLR-Laufzeit Komponenten als Abhängigkeiten, um DLR und die Beispielmodelle herunterzuladen.

Um diese Inferenzkomponente mit einem speziell trainierten DLR-Modell zu verwenden, erstellen Sie eine benutzerdefinierte Version der abhängigen Modellspeicherkomponente. Um Ihren eigenen benutzerdefinierten Inferenzcode zu verwenden, können Sie das Rezept dieser Komponente als Vorlage verwenden, um eine benutzerdefinierte Inferenzkomponente zu erstellen.

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.1.x
- 2.0.x

Typ

Diese Komponente ist eine generische Komponente (`aws.greengrass.generic`). Der Greengrass-Kern führt die Lebenszyklusskripte der Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf Kerngeräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Auf Greengrass-Core-Geräten, auf denen Amazon Linux 2 oder Ubuntu 18.04 ausgeführt wird, ist die GNU C Library (Glibc) Version 2.27 oder höher auf dem Gerät installiert.
- Auf ARMv7L-Geräten wie Raspberry Pi sind Abhängigkeiten für OpenCV-Python auf dem Gerät installiert. Führen Sie den folgenden Befehl aus, um die Abhängigkeiten zu installieren.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- Raspberry Pi-Geräte, auf denen Raspberry Pi OS Bullseye ausgeführt wird, müssen die folgenden Anforderungen erfüllen:
  - NumPy 1.22.4 oder höher auf dem Gerät installiert. Raspberry Pi OS Bullseye enthält eine frühere Version von NumPy, sodass Sie den folgenden Befehl ausführen können, um das Gerät zu aktualisieren NumPy .

    ```
    pip3 install --upgrade numpy
    ```

  - Der ältere Kamerastack ist auf dem Gerät aktiviert. Raspberry Pi OS Bullseye enthält einen neuen Kamerastack, der standardmäßig aktiviert und nicht kompatibel ist. Sie müssen also den älteren Kamerastack aktivieren.

    Um den Legacy-Kamerastack zu aktivieren

    1. Führen Sie den folgenden Befehl aus, um das Raspberry Pi-Konfigurationstool zu öffnen.

       ```
       sudo raspi-config
       ```

    2. Wählen Sie Schnittstellenoptionen.
    3. Wählen Sie Legacy-Kamera aus, um den Legacy-Kamerastack zu aktivieren.
    4. Starten Sie den Raspberry Pi neu.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt sie AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und all ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitstellen zu können. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente sowie die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste der Abhängigkeiten.

2.1.13 and 2.1.14

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.1.13 und 2.1.14 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.13.0	Weich
Modellspeicher zur DLR-Objekterkennung	~2.1.0	Hart
DLR	~1.6.0	Hart

2.1.12

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.12 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.12.0	Weich
Modellspeicher zur DLR-Objekterkennung	~2.1.0	Hart
DLR	~1.6.0	Hart

2.1.11

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.11 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.11.0	Weich
Modellspeicher zur DLR-Objekterkennung	~2.1.0	Hart
DLR	~1.6.0	Hart

2.1.10

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.10 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.10.0	Weich
Modellspeicher zur DLR-Objekterkennung	~2.1.0	Hart
DLR	~1.6.0	Hart

2.1.9

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.9 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.9.0	Weich
Modellspeicher zur DLR-Objekterkennung	~2.1.0	Hart

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
DLR	~1.6.0	Hart

2.1.8

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.8 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.8.0	Weich
Modellspeicher zur DLR-Objekterkennung	~2.1.0	Hart
DLR	~1.6.0	Hart

2.1.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.7.0	Weich
Modellspeicher zur DLR-Objekterkennung	~2.1.0	Hart
DLR	~1.6.0	Hart

2.1.6

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.6.0	Weich

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Modellspeicher zur DLR-Objekterkennung	~2.1.0	Hart
DLR	~1.6.0	Hart

2.1.4 - 2.1.5

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.1.4 bis 2.1.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.5.0	Weich
Modellspeicher zur DLR-Objekterkennung	~2.1.0	Hart
DLR	~1.6.0	Hart

2.1.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.4.0	Weich
Modellspeicher zur DLR-Objekterkennung	~2.1.0	Hart
DLR	~1.6.0	Hart

2.1.2

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.3.0	Weich
Modellspeicher zur DLR-Objekterkennung	~2.1.0	Hart
DLR	~1.6.0	Hart

2.1.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.2.0	Weich
Modellspeicher zur DLR-Objekterkennung	~2.1.0	Hart
DLR	~1.6.0	Hart

2.0.x

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.x dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	~2.0.0	Weich
Modellspeicher für DLR-Objekterkennung	~2.0.0	Hart
DLR	~1.3.0	Weich

Konfiguration

Diese Komponente bietet die folgenden Konfigurationsparameter, die Sie bei der Bereitstellung der Komponente anpassen können.

2.1.x

`accessControl`

(Optional) Das Objekt, das die Autorisierungsrichtlinie enthält, die es der Komponente ermöglicht, Nachrichten unter dem Standardthema für Benachrichtigungen zu veröffentlichen.

Standard:

```
{
   "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.DLRObjectDetection:mqttproxy:1": {
         "policyDescription": "Allows access to publish via topic ml/dlr/object-
detection.",
         "operations": [
            "aws.greengrass#PublishToIoTCore"
         ],
         "resources": [
            "ml/dlr/object-detection"
         ]
      }
   }
}
```

`PublishResultsOnTopic`

(Optional) Das Thema, zu dem Sie die Inferenzergebnisse veröffentlichen möchten. Wenn Sie diesen Wert ändern, müssen Sie auch den Wert von `resources` im `accessControl` Parameter so ändern, dass er Ihrem benutzerdefinierten Themennamen entspricht.

Standard: `ml/dlr/object-detection`

`Accelerator`

Der Beschleuniger, den Sie verwenden möchten. Unterstützte Werte sind `cpu` und `gpu`.

Die Beispielmodelle in der abhängigen Modellkomponente unterstützen nur CPU-Beschleunigung. Um die GPU-Beschleunigung mit einem anderen benutzerdefinierten Modell

zu verwenden, erstellen Sie eine benutzerdefinierte Modellkomponente, um die Komponente des öffentlichen Modells zu überschreiben.

Standard: `cpu`

`ImageDirectory`

(Optional) Der Pfad des Ordners auf dem Gerät, in dem Inferenzkomponenten Bilder lesen. Sie können diesen Wert an einen beliebigen Ort auf Ihrem Gerät ändern, auf den Sie Lese-/Schreibzugriff haben.

Standard: *`/greengrass/v2`*`/packages/artifacts-unarchived/`*`component-name`*`/object_detection/sample_images/`

> **Note**
>
> Wenn Sie den Wert auf setzen `UseCamera`true, wird dieser Konfigurationsparameter ignoriert.

`ImageName`

(Optional) Der Name des Bilds, das die Inferenzkomponente als Eingabe für eine Make-Vorhersage verwendet. Die Komponente sucht in dem unter angegebenen Ordner nach dem Bild. `ImageDirectory` Standardmäßig verwendet die Komponente das Beispielbild im Standard-Bildverzeichnis. AWS IoT Greengrass unterstützt die folgenden Bildformate: `jpeg`jpg,png, und`npy`.

Standard: `objects.jpg`

> **Note**
>
> Wenn Sie den Wert `UseCamera` auf setzen`true`, wird dieser Konfigurationsparameter ignoriert.

`InferenceInterval`

(Optional) Die Zeit in Sekunden zwischen den einzelnen Vorhersagen des Inferenzcodes. Der Beispiel-Inferenzcode wird unbegrenzt ausgeführt und wiederholt seine Vorhersagen im angegebenen Zeitintervall. Sie können dieses Intervall beispielsweise auf ein kürzeres

Intervall ändern, wenn Sie mit einer Kamera aufgenommene Bilder für Vorhersagen in Echtzeit verwenden möchten.

Standard: `3600`

`ModelResourceKey`

(Optional) Die Modelle, die in der abhängigen öffentlichen Modellkomponente verwendet werden. Ändern Sie diesen Parameter nur, wenn Sie die Komponente des öffentlichen Modells durch eine benutzerdefinierte Komponente überschreiben.

Standard:

```
{
    "armv7l": "DLR-yolo3-armv7l-cpu-ObjectDetection",
    "aarch64": "DLR-yolo3-aarch64-gpu-ObjectDetection",
    "x86_64": "DLR-yolo3-x86_64-cpu-ObjectDetection",
    "windows": "DLR-resnet50-win-cpu-ObjectDetection"
}
```

`UseCamera`

(Optional) Zeichenkettenwert, der definiert, ob Bilder von einer Kamera verwendet werden sollen, die mit dem Greengrass-Core-Gerät verbunden ist. Unterstützte Werte sind `true` und `false`.

Wenn Sie diesen Wert auf setzen`true`, greift der Beispiel-Inferenzcode auf die Kamera auf Ihrem Gerät zu und führt die Inferenz lokal für das aufgenommene Bild aus. Die Werte der `ImageDirectory` Parameter `ImageName` und werden ignoriert. Stellen Sie sicher, dass der Benutzer, der diese Komponente ausführt, Lese-/Schreibzugriff auf den Speicherort hat, an dem die Kamera die aufgenommenen Bilder speichert.

Standard: `false`

> **Note**
>
> Wenn Sie sich das Rezept für diese Komponente ansehen, erscheint der `UseCamera` Konfigurationsparameter nicht in der Standardkonfiguration. Sie können den Wert dieses Parameters jedoch in einem Update zur Zusammenführung der Konfiguration ändern, wenn Sie die Komponente bereitstellen.
> Wenn Sie `UseCamera` auf setzen`true`, müssen Sie auch einen Symlink erstellen, damit die Inferenzkomponente über die virtuelle Umgebung, die von der Runtime-

Komponente erstellt wurde, auf Ihre Kamera zugreifen kann. Weitere Hinweise zur Verwendung einer Kamera mit den Beispiel-Inferenzkomponenten finden Sie unter. Komponentenkonfigurationen aktualisieren

2.0.x

`MLRootPath`

(Optional) Der Pfad des Ordners auf Linux-Kerngeräten, in dem Inferenzkomponenten Bilder lesen und Inferenzergebnisse schreiben. Sie können diesen Wert in einen beliebigen Speicherort auf Ihrem Gerät ändern, auf den der Benutzer, der diese Komponente ausführt, Lese-/Schreibzugriff hat.

Standard: */greengrass/v2*`/work/variant.DLR/greengrass_ml`

Standard: */greengrass/v2*`/work/variant.TensorFlowLite/greengrass_ml`

`Accelerator`

Ändern Sie nicht. Derzeit ist der einzige unterstützte Wert für den Beschleuniger`cpu`, weil die Modelle in den abhängigen Modellkomponenten nur für den CPU-Beschleuniger kompiliert werden.

`ImageName`

(Optional) Der Name des Bilds, das die Inferenzkomponente als Eingabe für eine Make-Vorhersage verwendet. Die Komponente sucht in dem unter angegebenen Ordner nach dem Bild. `ImageDirectory` Der Standardspeicherort ist*MLRootPath*`/images`. AWS IoT Greengrass unterstützt die folgenden Bildformate: `jpeg`jpg,png, und`npy`.

Standard: `objects.jpg`

`InferenceInterval`

(Optional) Die Zeit in Sekunden zwischen den einzelnen Vorhersagen des Inferenzcodes. Der Beispiel-Inferenzcode wird unbegrenzt ausgeführt und wiederholt seine Vorhersagen im angegebenen Zeitintervall. Sie können dieses Intervall beispielsweise auf ein kürzeres Intervall ändern, wenn Sie mit einer Kamera aufgenommene Bilder für Vorhersagen in Echtzeit verwenden möchten.

Standard: `3600`

`ModelResourceKey`

(Optional) Die Modelle, die in der abhängigen öffentlichen Modellkomponente verwendet werden. Ändern Sie diesen Parameter nur, wenn Sie die Komponente des öffentlichen Modells durch eine benutzerdefinierte Komponente überschreiben.

Standard:

```
{
    armv7l: "DLR-yolo3-armv7l-cpu-ObjectDetection",
    x86_64: "DLR-yolo3-x86_64-cpu-ObjectDetection"
}
```

Lokale Protokolldatei

Diese Komponente verwendet die folgende Protokolldatei.

Linux

```
/greengrass/v2/logs/aws.greengrass.DLRObjectDetection.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.DLRObjectDetection.log
```

Um die Protokolle dieser Komponente einzusehen

- Führen Sie den folgenden Befehl auf dem Kerngerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

  Linux

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

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.DLRObjectDetection.log -Tail 10
   -Wait
  ```

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
2.1.14	Die Version wurde für die Version Greengrass Nucleus 2.12.5 aktualisiert.
2.1.13	Die Version wurde für die Version 2.12.0 von Greengrass Nucleus aktualisiert.
2.1.12	Die Version wurde für die Version 2.11.0 von Greengrass Nucleus aktualisiert.
2.1.11	Die Version wurde für die Version 2.10.0 von Greengrass Nucleus aktualisiert.
2.1.10	Die Version wurde für die Version 2.9.0 von Greengrass Nucleus aktualisiert.
2.1.9	Die Version wurde für die Version 2.8.0 von Greengrass Nucleus aktualisiert.
2.1.8	Die Version wurde für die Version 2.7.0 von Greengrass Nucleus aktualisiert.
2.1.7	Die Version wurde für die Version 2.6.0 von Greengrass Nucleus aktualisiert.
2.1.6	Die Version wurde für die Version 2.5.0 von Greengrass Nucleus aktualisiert.
2.1.5	Die Komponente wurde insgesamt veröffentlicht. AWS-Regionen
2.1.4	Die Version wurde für die Version 2.4.0 von Greengrass Nucleus aktualisiert. Diese Version ist in Europa (London) nicht verfügbar (`eu-west-2`).
2.1.3	Die Version wurde für die Version 2.3.0 von Greengrass Nucleus aktualisiert.
2.1.2	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem mit der Bildskalierung, das zu ungenauen Begrenzungsrahmen in den Beispielinferenzergebnissen der DLR-Objek terkennung führte.
2.1.1	Neue Features • Verwenden Sie Deep Learning Runtime v1.6.0.

Version	Änderungen
	• Unterstützung für die Erkennung von Beispielobjekten auf Armv8-Plattformen (AArch64) hinzugefügt. Dies erweitert die Unterstützung für maschinelles Lernen für Greengrass-Kerngeräte, auf denen NVIDIA Jetson ausgeführt wird, wie z. B. den Jetson Nano. • Aktivieren Sie die Kameraintegration für Probeninferenz. Verwenden Sie den neuen `UseCamera` Konfigurationsparameter, um den Beispiel-Inferenzcode für den Zugriff auf die Kamera auf Ihrem Greengrass-Core-Gerät zu aktivieren und die Inferenz lokal auf dem aufgenommenen Bild auszuführen. • Fügen Sie Unterstützung für die Veröffentlichung von Inferenzergebnissen zur hinzu. AWS Cloud Verwenden Sie den neuen `PublishResultsOnTopic` Konfigurationsparameter, um das Thema anzugeben, zu dem Sie Ergebnisse veröffentlichen möchten. • Fügen Sie den neuen `ImageDirectory` Konfigurationsparameter hinzu, mit dem Sie ein benutzerdefiniertes Verzeichnis für das Bild angeben können, für das Sie Inferenzen durchführen möchten. Fehlerkorrekturen und Verbesserungen • Schreiben Sie die Inferenzergebnisse in die Protokolldatei der Komponente statt in eine separate Inferenzdatei. • Verwenden Sie das AWS IoT Greengrass Core-Software-Logging-Modul, um die Komponentenausgabe zu protokollieren. • Verwenden Sie das AWS IoT Device SDK , um die Komponentenkonfiguration zu lesen und Konfigurationsänderungen anzuwenden.
2.0.4	Erste Version

Modellspeicher für die DLR-Bildklassifizierung

Der DLR-Bildklassifizierungsmodellspeicher ist eine Modellkomponente für maschinelles Lernen, die vortrainierte ResNet -50 Modelle als Greengrass-Artefakte enthält. Die in dieser Komponente verwendeten vortrainierten Modelle werden aus dem GluonCV Model Zoo abgerufen und mit Neo Deep Learning Runtime kompiliert. SageMaker

Die DLR-Inferenzkomponente zur Bildklassifizierung verwendet diese Komponente als Abhängigkeit für die Modellquelle. Um ein benutzerdefiniertes DLR-Modell zu verwenden, erstellen Sie eine benutzerdefinierte Version dieser Modellkomponente und fügen Sie Ihr benutzerdefiniertes Modell als Komponentenartefakt hinzu. Sie können das Rezept dieser Komponente als Vorlage verwenden, um benutzerdefinierte Modellkomponenten zu erstellen.

> **Note**
>
> Der Name der DLR-Speicherkomponente für das Bildklassifizierungsmodell variiert je nach Version. Der Komponentenname für Version 2.1.x und spätere Versionen lautet. `variant.DLR.ImageClassification.ModelStore` Der Komponentenname für Version 2.0.x lautet. `variant.ImageClassification.ModelStore`

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.1.x () `variant.DLR.ImageClassification.ModelStore`
- 2.0.x () `variant.ImageClassification.ModelStore`

Typ

Diese Komponente ist eine generische Komponente (`aws.greengrass.generic`). Der Greengrass-Kern führt die Lebenszyklusskripte der Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf Kerngeräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Auf Greengrass-Core-Geräten, auf denen Amazon Linux 2 oder Ubuntu 18.04 ausgeführt wird, ist die GNU C Library (Glibc) Version 2.27 oder höher auf dem Gerät installiert.
- Auf ARMv7L-Geräten wie Raspberry Pi sind Abhängigkeiten für OpenCV-Python auf dem Gerät installiert. Führen Sie den folgenden Befehl aus, um die Abhängigkeiten zu installieren.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- Raspberry Pi-Geräte, auf denen Raspberry Pi OS Bullseye ausgeführt wird, müssen die folgenden Anforderungen erfüllen:
  - NumPy 1.22.4 oder höher auf dem Gerät installiert. Raspberry Pi OS Bullseye enthält eine frühere Version von NumPy, sodass Sie den folgenden Befehl ausführen können, um das Gerät zu aktualisieren NumPy .

    ```
    pip3 install --upgrade numpy
    ```

  - Der ältere Kamerastack ist auf dem Gerät aktiviert. Raspberry Pi OS Bullseye enthält einen neuen Kamerastack, der standardmäßig aktiviert und nicht kompatibel ist. Sie müssen also den älteren Kamerastack aktivieren.

    Um den Legacy-Kamerastack zu aktivieren

    1. Führen Sie den folgenden Befehl aus, um das Raspberry Pi-Konfigurationstool zu öffnen.

       ```
       sudo raspi-config
       ```

2. Wählen Sie Schnittstellenoptionen.
3. Wählen Sie Legacy-Kamera aus, um den Legacy-Kamerastack zu aktivieren.
4. Starten Sie den Raspberry Pi neu.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt sie AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und all ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitstellen zu können. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente sowie die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste der Abhängigkeiten.

2.1.12 - 2.1.14

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.1.12 und 2.1.13 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.13.0	Weich

2.1.11

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.11 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.12.0	Weich

2.1.10

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.10 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.11.0	Weich

2.1.9

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.9 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.10.0	Weich

2.1.8

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.8 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.9.0	Weich

2.1.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.8.0	Weich

2.1.6

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.7.0	Weich

2.1.5

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.6.0	Weich

2.1.4

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.5.0	Weich

2.1.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.4.0	Weich

2.1.2

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.3.0	Weich

2.1.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.2.0	Weich

2.0.x

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.x dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	~2.0.0	Weich

Konfiguration

Diese Komponente hat keine Konfigurationsparameter.

Lokale Protokolldatei

Diese Komponente gibt keine Protokolle aus.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
2.1.13	Die Version wurde für die Version Greengrass Nucleus 2.12.5 aktualisiert.
2.1.12	Die Version wurde für die Version 2.12.0 von Greengrass Nucleus aktualisiert.
2.1.11	Die Version wurde für die Version 2.11.0 von Greengrass Nucleus aktualisiert.
2.1.10	Die Version wurde für die Version 2.10.0 von Greengrass Nucleus aktualisiert.
2.1.9	Die Version wurde für die Version 2.9.0 von Greengrass Nucleus aktualisiert.
2.1.8	Die Version wurde für die Version 2.8.0 von Greengrass Nucleus aktualisiert.

Version	Änderungen
2.1.7	Die Version wurde für die Version 2.7.0 von Greengrass Nucleus aktualisiert.
2.1.6	Die Version wurde für die Version 2.6.0 von Greengrass Nucleus aktualisiert.
2.1.5	Neue Features • Fügt Beispielmodelle zur Bildklassifizierung für Windows Core-Geräte hinzu. • Die Version wurde für die Version 2.5.0 von Greengrass Nucleus aktualisiert.
2.1.4	Die Version wurde für die Version 2.4.0 von Greengrass Nucleus aktualisiert.
2.1.3	Die Version wurde für die Version 2.3.0 von Greengrass Nucleus aktualisiert.
2.1.2	Die Version wurde für die Version 2.2.0 von Greengrass Nucleus aktualisiert.
2.1.1	Neue Features • Fügen Sie ein Beispiel für ein ResNet -50-Bildklassifizierungsmodell für Armv8-Plattformen (AArch64) hinzu. Dies erweitert die Unterstützung für maschinelles Lernen für Greengrass-Kerngeräte, auf denen NVIDIA Jetson ausgeführt wird, wie z. B. den Jetson Nano.
2.0.4	Erste Version

Modellspeicher für DLR-Objekterkennung

Der DLR-Objekterkennungsmodellspeicher ist eine Modellkomponente für maschinelles Lernen, die vortrainierte YoloV3-Modelle als Greengrass-Artefakte enthält. Die in dieser Komponente verwendeten Beispielmodelle werden aus dem GluonCV Model Zoo abgerufen und mit Neo Deep Learning Runtime kompiliert. SageMaker

Die DLR-Inferenzkomponente zur Objekterkennung verwendet diese Komponente als Abhängigkeit für die Modellquelle. Um ein benutzerdefiniertes DLR-Modell zu verwenden, erstellen Sie eine benutzerdefinierte Version dieser Modellkomponente und fügen Sie Ihr benutzerdefiniertes Modell als Komponentenartefakt hinzu. Sie können das Rezept dieser Komponente als Vorlage verwenden, um benutzerdefinierte Modellkomponenten zu erstellen.

> **Note**
>
> Der Name der Speicherkomponente für das DLR-Objekterkennungsmodell variiert je nach Version. Der Komponentenname für Version 2.1.x und spätere Versionen lautet. `variant.DLR.ObjectDetection.ModelStore` Der Komponentenname für Version 2.0.x lautet. `variant.ObjectDetection.ModelStore`

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.1.x
- 2.0.x

Typ

Diese Komponente ist eine generische Komponente (`aws.greengrass.generic`). Der Greengrass-Kern führt die Lebenszyklusskripte der Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf Kerngeräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Auf Greengrass-Core-Geräten, auf denen Amazon Linux 2 oder Ubuntu 18.04 ausgeführt wird, ist die GNU C Library (Glibc) Version 2.27 oder höher auf dem Gerät installiert.
- Auf ARMv7L-Geräten wie Raspberry Pi sind Abhängigkeiten für OpenCV-Python auf dem Gerät installiert. Führen Sie den folgenden Befehl aus, um die Abhängigkeiten zu installieren.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- Raspberry Pi-Geräte, auf denen Raspberry Pi OS Bullseye ausgeführt wird, müssen die folgenden Anforderungen erfüllen:
  - NumPy 1.22.4 oder höher auf dem Gerät installiert. Raspberry Pi OS Bullseye enthält eine frühere Version von NumPy, sodass Sie den folgenden Befehl ausführen können, um das Gerät zu aktualisieren NumPy .

    ```
    pip3 install --upgrade numpy
    ```

  - Der ältere Kamerastack ist auf dem Gerät aktiviert. Raspberry Pi OS Bullseye enthält einen neuen Kamerastack, der standardmäßig aktiviert und nicht kompatibel ist. Sie müssen also den älteren Kamerastack aktivieren.

    Um den Legacy-Kamerastack zu aktivieren

    1. Führen Sie den folgenden Befehl aus, um das Raspberry Pi-Konfigurationstool zu öffnen.

       ```
       sudo raspi-config
       ```

    2. Wählen Sie Schnittstellenoptionen.
    3. Wählen Sie Legacy-Kamera aus, um den Legacy-Kamerastack zu aktivieren.
    4. Starten Sie den Raspberry Pi neu.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt sie AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und all ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitstellen zu können. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente sowie die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste der Abhängigkeiten.

2.1.13 and 2.1.14

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.1.13 und 2.1.14 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.13.0	Weich

2.1.12

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.12 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.12.0	Weich

2.1.11

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.11 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.11.0	Weich

2.1.10

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.10 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.10.0	Weich

2.1.9

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.9 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.9.0	Weich

2.1.8

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.8 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.8.0	Weich

2.1.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.7.0	Weich

2.1.5 and 2.1.6

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.1.5 und 2.1.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.6.0	Weich

2.1.4

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.5.0	Weich

2.1.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.4.0	Weich

2.1.2

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.3.0	Weich

2.1.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.2.0	Weich

2.0.x

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.x dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	~2.0.0	Weich

Konfiguration

Diese Komponente hat keine Konfigurationsparameter.

Lokale Protokolldatei

Diese Komponente gibt keine Protokolle aus.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
2.1.14	Die Version wurde für die Version Greengrass Nucleus 2.12.5 aktualisiert.
2.1.13	Die Version wurde für die Version 2.12.0 von Greengrass Nucleus aktualisiert.
2.1.12	Die Version wurde für die Version 2.11.0 von Greengrass Nucleus aktualisiert.
2.1.11	Die Version wurde für die Version 2.10.0 von Greengrass Nucleus aktualisiert.
2.1.10	Die Version wurde für die Version 2.9.0 von Greengrass Nucleus aktualisiert.
2.1.9	Die Version wurde für die Version 2.8.0 von Greengrass Nucleus aktualisiert.
2.1.8	Die Version wurde für die Version 2.7.0 von Greengrass Nucleus aktualisiert.
2.1.7	Die Version wurde für die Version 2.6.0 von Greengrass Nucleus aktualisiert.
2.1.6	Fügt ein CPU-Modell hinzu, um ein Problem auf Armv8-Geräten (AArch64) zu beheben.

Version	Änderungen
2.1.5	Neue Features • Fügt Beispielmodelle zur Objekterkennung für Windows-Kerngeräte hinzu. Fehlerkorrekturen und Verbesserungen • Die Version wurde für die Version 2.5.0 von Greengrass Nucleus aktualisiert.
2.1.4	Die Version wurde für die Version 2.4.0 von Greengrass Nucleus aktualisiert.
2.1.3	Die Version wurde für die Version 2.3.0 von Greengrass Nucleus aktualisiert.
2.1.2	Die Version wurde für die Version 2.2.0 von Greengrass Nucleus aktualisiert.
2.1.1	Neue Features • Fügen Sie ein Beispiel für ein YOLOv3-Objekterkennungsmodell für Armv8-Plattformen (AArch64) hinzu. Dies erweitert die Unterstützung für maschinelles Lernen für Greengrass-Kerngeräte, auf denen NVIDIA Jetson ausgeführt wird, wie z. B. den Jetson Nano.
2.0.4	Erste Version

DLR-Laufzeit

Die DLR-Laufzeitkomponente (`variant.DLR`) enthält ein Skript, das Deep Learning Runtime (DLR) und seine Abhängigkeiten in einer virtuellen Umgebung auf Ihrem Gerät installiert. Die DLR-Objekterkennung Komponenten DLR-Bildklassifizierung und verwenden diese Komponente als Abhängigkeit für die Installation von DLR. Mit der Komponentenversion 1.6.x wird DLR v1.6.0 installiert und mit der Komponentenversion 1.3.x wird DLR v1.3.0 installiert.

Um eine andere Laufzeit zu verwenden, können Sie das Rezept dieser Komponente als Vorlage verwenden, um eine benutzerdefinierte Komponente für maschinelles Lernen zu erstellen.

Themen

- Versionen
- Typ

Versionen

Diese Komponente hat die folgenden Versionen:

- 1.6.x
- 1.3.x

Typ

Diese Komponente ist eine generische Komponente () `aws.greengrass.generic`. Der Greengrass-Kern führt die Lebenszyklusskripte der Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf Kerngeräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Auf Greengrass-Core-Geräten, auf denen Amazon Linux 2 oder Ubuntu 18.04 ausgeführt wird, ist die GNU C Library (Glibc) Version 2.27 oder höher auf dem Gerät installiert.

- Auf ARMv7L-Geräten wie Raspberry Pi sind Abhängigkeiten für OpenCV-Python auf dem Gerät installiert. Führen Sie den folgenden Befehl aus, um die Abhängigkeiten zu installieren.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- Raspberry Pi-Geräte, auf denen Raspberry Pi OS Bullseye ausgeführt wird, müssen die folgenden Anforderungen erfüllen:
  - NumPy 1.22.4 oder höher auf dem Gerät installiert. Raspberry Pi OS Bullseye enthält eine frühere Version von NumPy, sodass Sie den folgenden Befehl ausführen können, um das Gerät zu aktualisieren NumPy .

    ```
    pip3 install --upgrade numpy
    ```

  - Der ältere Kamerastack ist auf dem Gerät aktiviert. Raspberry Pi OS Bullseye enthält einen neuen Kamerastack, der standardmäßig aktiviert und nicht kompatibel ist. Sie müssen also den älteren Kamerastack aktivieren.

    Um den Legacy-Kamerastack zu aktivieren

    1. Führen Sie den folgenden Befehl aus, um das Raspberry Pi-Konfigurationstool zu öffnen.

       ```
       sudo raspi-config
       ```

    2. Wählen Sie Schnittstellenoptionen.
    3. Wählen Sie Legacy-Kamera aus, um den Legacy-Kamerastack zu aktivieren.
    4. Starten Sie den Raspberry Pi neu.

Endpunkte und Anschlüsse

Standardmäßig verwendet diese Komponente ein Installationsskript, um Pakete mit den `pip` Befehlen`apt`, `yum`, `brew`, und zu installieren, je nachdem, welche Plattform das Kerngerät verwendet. Diese Komponente muss in der Lage sein, ausgehende Anfragen an verschiedene Paketindizes und Repositorys auszuführen, um das Installationsskript auszuführen. Damit der ausgehende Datenverkehr dieser Komponente über einen Proxy oder eine Firewall übertragen werden kann, müssen Sie die Endpunkte für die Paketindizes und Repositorys identifizieren, zu denen Ihr Kerngerät eine Verbindung zur Installation herstellt.

Beachten Sie bei der Identifizierung von Endpunkten, die für das Installationsskript dieser Komponente erforderlich sind, Folgendes:

- Die Endpunkte hängen von der Plattform des Kerngeräts ab. Beispielsweise verwendet ein Kerngerät, auf dem Ubuntu ausgeführt wird, `apt` eher als `yum` oder`brew`. Darüber hinaus haben Geräte, die denselben Paketindex verwenden, möglicherweise unterschiedliche Quelllisten, sodass sie Pakete aus verschiedenen Repositorys abrufen können.
- Die Endpunkte können sich bei mehreren Geräten, die denselben Paketindex verwenden, unterscheiden, da jedes Gerät über eigene Quelllisten verfügt, die definieren, wo Pakete abgerufen werden sollen.
- Die Endpunkte können sich im Laufe der Zeit ändern. Jeder Paketindex enthält die URLs der Repositorys, in die Sie Pakete herunterladen, und der Besitzer eines Pakets kann ändern, welche URLs der Paketindex bereitstellt.

Weitere Informationen zu den Abhängigkeiten, die diese Komponente installiert, und zur Deaktivierung des Installationsskripts finden Sie im UseInstallerKonfigurationsparameter.

Weitere Informationen zu Endpunkten und Ports, die für den Basisbetrieb erforderlich sind, finden Sie unterZulassen von Gerätedatenverkehr über einen Proxy oder eine Firewall.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und all ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitstellen zu können. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente sowie die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste der Abhängigkeiten.

1.6.11 - 1.6.16

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 1.6.11 bis 1.6.16 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <3.0.0	Weich

1.6.10

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.6.10 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.9.0	Weich

1.6.9

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.6.9 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.8.0	Weich

1.6.8

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.6.8 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.7.0	Weich

1.6.6 and 1.6.7

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 1.6.6 und 1.6.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.6.0	Weich

1.6.4 and 1.6.5

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 1.6.4 und 1.6.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.5.0	Weich

1.6.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.6.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.4.0	Weich

1.6.2

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.6.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.3.0	Weich

1.6.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.6.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.2.0	Weich

1.3.x

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.3.x dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	~2.0.0	Weich

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zu den Komponentenrezepten.

Konfiguration

Diese Komponente stellt die folgenden Konfigurationsparameter bereit, die Sie bei der Bereitstellung der Komponente anpassen können.

`MLRootPath`

(Optional) Der Pfad des Ordners auf Linux-Kerngeräten, in dem Inferenzkomponenten Bilder lesen und Inferenzergebnisse schreiben. Sie können diesen Wert in einen beliebigen Speicherort auf Ihrem Gerät ändern, auf den der Benutzer, der diese Komponente ausführt, Lese-/Schreibzugriff hat.

Standard: */greengrass/v2*`/work/variant.DLR/greengrass_ml`

`WindowsMLRootPath`

Diese Funktion ist in Version 1.6.6 und höher dieser Komponente verfügbar.

(Optional) Der Pfad des Ordners auf dem Windows Core-Gerät, in dem Inferenzkomponenten Bilder lesen und Inferenzergebnisse schreiben. Sie können diesen Wert in einen beliebigen Speicherort auf Ihrem Gerät ändern, auf den der Benutzer, der diese Komponente ausführt, Lese-/Schreibzugriff hat.

Standard: *C:\greengrass\v2*`\\work\\variant.DLR\\greengrass_ml`

`UseInstaller`

(Optional) Zeichenkettenwert, der definiert, ob das Installationsskript in dieser Komponente zur Installation von DLR und seinen Abhängigkeiten verwendet werden soll. Unterstützte Werte sind `true` und `false`.

Legen Sie diesen Wert auf fest, `false` wenn Sie ein benutzerdefiniertes Skript für die DLR-Installation verwenden möchten oder wenn Sie Laufzeitabhängigkeiten in ein vorgefertigtes Linux-Image aufnehmen möchten. Um diese Komponente mit den von AWS-bereitgestellten DLR-Inferenzkomponenten zu verwenden, installieren Sie die folgenden Bibliotheken, einschließlich aller Abhängigkeiten, und stellen Sie sie dem Systembenutzer zur Verfügung, z. B. dem Systembenutzer, der die ML-Komponenten `ggc_user` ausführt.

- Python 3.7 oder höher, auch `pip` für Ihre Version von Python.
- Deep Learning Runtime v1.6.0
- NumPy.
- OpenCV-Python.
- AWS IoT Device SDK v2 für Python.
- AWS Common Runtime (CRT) Python.
- Picamera (nur für Raspberry Pi-Geräte).
- `awscamModul` (für AWS DeepLens Geräte).
- LibGL (für Linux-Geräte)

Standard: `true`

Verwendung

Verwenden Sie diese Komponente mit dem `UseInstaller` Konfigurationsparameter auf, `true` um DLR und seine Abhängigkeiten auf Ihrem Gerät zu installieren. Die Komponente richtet eine virtuelle Umgebung auf Ihrem Gerät ein, die OpenCV und die NumPy Bibliotheken enthält, die für DLR erforderlich sind.

> **Note**
>
> Das Installationsskript in dieser Komponente installiert auch die neuesten Versionen zusätzlicher Systembibliotheken, die für die Konfiguration der virtuellen Umgebung auf Ihrem Gerät und die Verwendung des installierten Frameworks für maschinelles Lernen erforderlich sind. Dadurch könnten die vorhandenen Systembibliotheken auf Ihrem Gerät aktualisiert werden. In der folgenden Tabelle finden Sie eine Liste der Bibliotheken, die diese Komponente für jedes unterstützte Betriebssystem installiert. Wenn Sie diesen Installationsvorgang anpassen möchten, setzen Sie den `UseInstaller` Konfigurationsparameter auf `false` und entwickeln Sie Ihr eigenes Installationsskript.

Plattform	Auf dem Gerätesystem installierte Bibliotheken	In der virtuellen Umgebung installierte Bibliotheken
Armv7l	`build-essential` , `cmake`, `ca-certificates` , `git`	`setuptools` , `wheel`
Amazon Linux 2	`mesa-libGL`	None
Ubuntu	`wget`	None

Wenn Sie Ihre Inferenzkomponente bereitstellen, überprüft diese Runtime-Komponente zunächst, ob DLR und seine Abhängigkeiten bereits auf Ihrem Gerät installiert sind. Falls nicht, werden sie dann für Sie installiert.

Lokale Protokolldatei

Diese Komponente verwendet die folgende Protokolldatei.

Linux

```
/greengrass/v2/logs/variant.DLR.log
```

Windows

```
C:\greengrass\v2\logs\variant.DLR.log
```

Um die Protokolle dieser Komponente einzusehen

- Führen Sie den folgenden Befehl auf dem Kerngerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

  Linux

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

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\variant.DLR.log -Tail 10 -Wait
```

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
1.6.16	Version für Greengrass Nucleus Version 2.12.5 aktualisiert.
1.6.12	Fehlerkorrekturen und Verbesserungen • Behebt das Installationsskript für Benutzer von Windows-Betriebssystemen.
1.6.11	Die Version wurde für die Version 2.9.0 von Greengrass Nucleus aktualisiert.
1.6.10	Die Version wurde für die Version 2.8.0 von Greengrass Nucleus aktualisiert.
1.6.9	Die Version wurde für die Version 2.7.0 von Greengrass Nucleus aktualisiert.
1.6.8	Die Version wurde für die Version 2.6.0 von Greengrass Nucleus aktualisiert.
1.6.7	Fehlerkorrekturen und Verbesserungen • Aktualisiert das `UseInstaller` Installationsskript zur Installation von LibGL, das auf bestimmten Linux-Plattformen standardmäßig nicht verfügbar ist. • Aktualisiert das `UseInstaller` Installationsskript so, dass es in der virtuellen Umgebung dieser Komponente immer Python 3.9 verwendet. Diese Änderung trägt dazu bei, die Kompatibilität mit anderen Bibliotheken sicherzustellen.
1.6.6	Neue Features • Fügt Unterstützung für Kerngeräte hinzu, auf denen Windows ausgeführt wird.

Version	Änderungen
	• Fügt den neuen `WindowsMLRootPath` Konfigurationsparameter hinzu, mit dem Sie den Ordner mit den Inferenzergebnissen auf Windows-Core-Geräten konfigurieren können.
1.6.5	Neue Features • Fügt den neuen `UseInstaller` Konfigurationsparameter hinzu, mit dem Sie das Installationsskript in dieser Komponente deaktivieren können.
1.6.4	Die Version wurde für die Version 2.4.0 von Greengrass Nucleus aktualisiert.
1.6.3	Die Version wurde für die Version 2.3.0 von Greengrass Nucleus aktualisiert.
1.6.2	Die Version wurde für die Version 2.2.0 von Greengrass Nucleus aktualisiert.
1.6.1	Neue Features • Installieren Sie Deep Learning Runtime v1.6.0 und seine Abhängigk eiten. • Unterstützung für die Installation von DLR auf Armv8-Plattformen (AArch64) hinzugefügt. Dies erweitert die Unterstützung für maschinel les Lernen für Greengrass-Kerngeräte, auf denen NVIDIA Jetson ausgeführt wird, wie z. B. den Jetson Nano. Fehlerkorrekturen und Verbesserungen • Installieren Sie das AWS IoT Device SDK in der virtuellen Umgebung, um die Komponentenkonfiguration zu lesen und die Konfigurationsände rungen anzuwenden. • Zusätzliche kleinere Fehlerkorrekturen und Verbesserungen.
1.3.2	Erste Version Installiert DLR v1.3.0.

TensorFlow Lite-Bildklassifizierung

Die TensorFlow Lite-Komponente zur Bildklassifizierung (`aws.greengrass.TensorFlowLiteImageClassification`) enthält einen Beispiel-Inferenzcode für die Durchführung von Inferenzen zur Bildklassifizierung mithilfe der TensorFlow

Lite-Laufzeit und ein vortrainiertes quantisiertes MobileNet 1.0-Beispielmodell. Diese Komponente verwendet die Variante TensorFlow Modellspeicher für Lite-Bildklassifizierung und die TensorFlow Lite-Laufzeit Komponenten als Abhängigkeiten, um die TensorFlow Lite-Laufzeit und das Beispielmodell herunterzuladen.

Um diese Inferenzkomponente mit einem individuell trainierten TensorFlow Lite-Modell zu verwenden, erstellen Sie eine benutzerdefinierte Version der abhängigen Modellspeicherkomponente. Um Ihren eigenen benutzerdefinierten Inferenzcode zu verwenden, können Sie das Rezept dieser Komponente als Vorlage verwenden, um eine benutzerdefinierte Inferenzkomponente zu erstellen.

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.1.x

Typ

Diese Komponente ist eine generische Komponente () `aws.greengrass.generic`. Der Greengrass-Kern führt die Lebenszyklusskripte der Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf Kerngeräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Auf Greengrass-Core-Geräten, auf denen Amazon Linux 2 oder Ubuntu 18.04 ausgeführt wird, ist die GNU C Library (Glibc) Version 2.27 oder höher auf dem Gerät installiert.
- Auf ARMv7L-Geräten wie Raspberry Pi sind Abhängigkeiten für OpenCV-Python auf dem Gerät installiert. Führen Sie den folgenden Befehl aus, um die Abhängigkeiten zu installieren.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- Raspberry Pi-Geräte, auf denen Raspberry Pi OS Bullseye ausgeführt wird, müssen die folgenden Anforderungen erfüllen:
  - NumPy 1.22.4 oder höher auf dem Gerät installiert. Raspberry Pi OS Bullseye enthält eine frühere Version von NumPy, sodass Sie den folgenden Befehl ausführen können, um das Gerät zu aktualisieren NumPy .

    ```
    pip3 install --upgrade numpy
    ```

  - Der ältere Kamerastack ist auf dem Gerät aktiviert. Raspberry Pi OS Bullseye enthält einen neuen Kamerastack, der standardmäßig aktiviert und nicht kompatibel ist. Sie müssen also den älteren Kamerastack aktivieren.

    Um den alten Kamerastack zu aktivieren

    1. Führen Sie den folgenden Befehl aus, um das Raspberry Pi-Konfigurationstool zu öffnen.

       ```
       sudo raspi-config
       ```

    2. Wählen Sie Schnittstellenoptionen.
    3. Wählen Sie Legacy-Kamera aus, um den Legacy-Kamerastack zu aktivieren.
    4. Starten Sie den Raspberry Pi neu.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt sie AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und all ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitstellen zu können. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente sowie die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können die Abhängigkeiten für jede Version der Komponente auch in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste der Abhängigkeiten.

2.1.11 and 2.1.12

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.1.11 und 2.1.12 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.13.0	Weich
TensorFlow Lite-Modellspeicher zur Bildklassifizierung	>=2.1.0 <2.2.0	Hart
TensorFlow Leicht	>=2,5,0 <2,6,0	Hart

2.1.10

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.10 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.12.0	Weich
TensorFlow Lite-Modellspeicher zur Bildklassifizierung	>=2.1.0 <2.2.0	Hart

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
TensorFlow Leicht	>=2,5,0 <2,6,0	Hart

2.1.9

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.9 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.11.0	Weich
TensorFlow Lite-Mode llspeicher zur Bildklass ifizierung	>=2.1.0 <2.2.0	Hart
TensorFlow Leicht	>=2,5,0 <2,6,0	Hart

2.1.8

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.8 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.10.0	Weich
TensorFlow Lite-Mode llspeicher zur Bildklass ifizierung	>=2.1.0 <2.2.0	Hart
TensorFlow Leicht	>=2,5,0 <2,6,0	Hart

2.1.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.9.0	Weich
TensorFlow Lite-Modellspeicher zur Bildklassifizierung	>=2.1.0 <2.2.0	Hart
TensorFlow Leicht	>=2,5,0 <2,6,0	Hart

2.1.6

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.8.0	Weich
TensorFlow Lite-Modellspeicher zur Bildklassifizierung	>=2.1.0 <2.2.0	Hart
TensorFlow Leicht	>=2,5,0 <2,6,0	Hart

2.1.5

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.7.0	Weich
TensorFlow Lite-Modellspeicher zur Bildklassifizierung	>=2.1.0 <2.2.0	Hart
TensorFlow Leicht	>=2,5,0 <2,6,0	Hart

2.1.4

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.6.0	Weich
TensorFlow Lite-Modellspeicher zur Bildklassifizierung	>=2.1.0 <2.2.0	Hart
TensorFlow Leicht	>=2,5,0 <2,6,0	Hart

2.1.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.5.0	Weich
TensorFlow Lite-Modellspeicher zur Bildklassifizierung	>=2.1.0 <2.2.0	Hart
TensorFlow Leicht	>=2,5,0 <2,6,0	Hart

2.1.2

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.4.0	Weich

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
TensorFlow Lite-Modellspeicher zur Bildklassifizierung	>=2.1.0 <2.2.0	Hart
TensorFlow Leicht	>=2,5,0 <2,6,0	Hart

2.1.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.3.0	Weich
TensorFlow Lite-Modellspeicher zur Bildklassifizierung	>=2.1.0 <2.2.0	Hart
TensorFlow Leicht	>=2,5,0 <2,6,0	Hart

2.1.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.2.0	Weich
TensorFlow Lite-Modellspeicher zur Bildklassifizierung	>=2.1.0 <2.2.0	Hart
TensorFlow Leicht	>=2,5,0 <2,6,0	Hart

Konfiguration

Diese Komponente stellt die folgenden Konfigurationsparameter bereit, die Sie bei der Bereitstellung der Komponente anpassen können.

`accessControl`

(Optional) Das Objekt, das die Autorisierungsrichtlinie enthält, die es der Komponente ermöglicht, Nachrichten unter dem Standardthema für Benachrichtigungen zu veröffentlichen.

Standard:

```
{
   "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.TensorFlowLiteImageClassification:mqttproxy:1": {
         "policyDescription": "Allows access to publish via topic ml/tflite/image-
classification.",
         "operations": [
            "aws.greengrass#PublishToIoTCore"
         ],
         "resources": [
            "ml/tflite/image-classification"
         ]
      }
   }
}
```

`PublishResultsOnTopic`

(Optional) Das Thema, zu dem Sie die Inferenzergebnisse veröffentlichen möchten. Wenn Sie diesen Wert ändern, müssen Sie auch den Wert von `resources` im `accessControl` Parameter so ändern, dass er Ihrem benutzerdefinierten Themennamen entspricht.

Standard: `ml/tflite/image-classification`

`Accelerator`

Der Beschleuniger, den Sie verwenden möchten. Unterstützte Werte sind `cpu` und `gpu`.

Die Beispielmodelle in der abhängigen Modellkomponente unterstützen nur CPU-Beschleunigung. Um die GPU-Beschleunigung mit einem anderen benutzerdefinierten Modell zu verwenden, erstellen Sie eine benutzerdefinierte Modellkomponente, um die öffentliche Modellkomponente zu überschreiben.

Standard: `cpu`

`ImageDirectory`

(Optional) Der Pfad des Ordners auf dem Gerät, in dem Inferenzkomponenten Bilder lesen. Sie können diesen Wert an einen beliebigen Ort auf Ihrem Gerät ändern, auf den Sie Lese-/Schreibzugriff haben.

Standard: */greengrass/v2*`/packages/artifacts-unarchived/`*component-name*`/image_classification/sample_images/`

> **Note**
>
> Wenn Sie den Wert auf setzen `UseCameratrue`, wird dieser Konfigurationsparameter ignoriert.

`ImageName`

(Optional) Der Name des Bilds, das die Inferenzkomponente als Eingabe für eine Make-Vorhersage verwendet. Die Komponente sucht in dem unter angegebenen Ordner nach dem Bild. `ImageDirectory` Standardmäßig verwendet die Komponente das Beispielbild im Standard-Bildverzeichnis. AWS IoT Greengrass unterstützt die folgenden Bildformate: `jpegjpg`,`png`, und`npy`.

Standard: `cat.jpeg`

> **Note**
>
> Wenn Sie den Wert `UseCamera` auf setzen`true`, wird dieser Konfigurationsparameter ignoriert.

`InferenceInterval`

(Optional) Die Zeit in Sekunden zwischen den einzelnen Vorhersagen des Inferenzcodes. Der Beispiel-Inferenzcode wird unbegrenzt ausgeführt und wiederholt seine Vorhersagen im angegebenen Zeitintervall. Sie können dieses Intervall beispielsweise auf ein kürzeres Intervall ändern, wenn Sie mit einer Kamera aufgenommene Bilder für Vorhersagen in Echtzeit verwenden möchten.

Standard: `3600`

`ModelResourceKey`

(Optional) Die Modelle, die in der abhängigen öffentlichen Modellkomponente verwendet werden. Ändern Sie diesen Parameter nur, wenn Sie die Komponente des öffentlichen Modells durch eine benutzerdefinierte Komponente überschreiben.

Standard:

```
{
    "model": "TensorFlowLite-Mobilenet"
}
```

`UseCamera`

(Optional) Zeichenkettenwert, der definiert, ob Bilder von einer Kamera verwendet werden sollen, die mit dem Greengrass-Core-Gerät verbunden ist. Unterstützte Werte sind `true` und `false`.

Wenn Sie diesen Wert auf setzen`true`, greift der Beispiel-Inferenzcode auf die Kamera auf Ihrem Gerät zu und führt die Inferenz lokal für das aufgenommene Bild aus. Die Werte der `ImageDirectory` Parameter `ImageName` und werden ignoriert. Stellen Sie sicher, dass der Benutzer, der diese Komponente ausführt, Lese-/Schreibzugriff auf den Speicherort hat, an dem die Kamera aufgenommene Bilder speichert.

Standard: `false`

> **Note**
>
> Wenn Sie sich das Rezept für diese Komponente ansehen, erscheint der `UseCamera` Konfigurationsparameter nicht in der Standardkonfiguration. Sie können den Wert dieses Parameters jedoch in einem Update zur Zusammenführung der Konfiguration ändern, wenn Sie die Komponente bereitstellen.
> Wenn Sie `UseCamera` auf setzen`true`, müssen Sie auch einen Symlink erstellen, damit die Inferenzkomponente über die virtuelle Umgebung, die von der Runtime-Komponente erstellt wurde, auf Ihre Kamera zugreifen kann. Weitere Hinweise zur Verwendung einer Kamera mit den Beispiel-Inferenzkomponenten finden Sie unter. Komponentenkonfigurationen aktualisieren

Lokale Protokolldatei

Diese Komponente verwendet die folgende Protokolldatei.

Linux

```
/greengrass/v2/logs/aws.greengrass.TensorFlowLiteImageClassification.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.TensorFlowLiteImageClassification.log
```

Um die Protokolle dieser Komponente einzusehen

- Führen Sie den folgenden Befehl auf dem Kerngerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

  Linux

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

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs
  \aws.greengrass.TensorFlowLiteImageClassification.log -Tail 10 -Wait
  ```

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
2.1.12	Die Version wurde für die Version Greengrass Nucleus 2.12.5 aktualisiert.
2.1.11	Die Version wurde für die Version 2.12.0 von Greengrass Nucleus aktualisiert.

Version	Änderungen
2.1.10	Die Version wurde für die Version 2.11.0 von Greengrass Nucleus aktualisiert.
2.1.9	Die Version wurde für die Version 2.10.0 von Greengrass Nucleus aktualisiert.
2.1.8	Die Version wurde für die Version 2.9.0 von Greengrass Nucleus aktualisiert.
2.1.7	Die Version wurde für die Version 2.8.0 von Greengrass Nucleus aktualisiert.
2.1.6	Die Version wurde für die Version 2.7.0 von Greengrass Nucleus aktualisiert.
2.1.5	Die Version wurde für die Version 2.6.0 von Greengrass Nucleus aktualisiert.
2.1.4	Die Version wurde für die Version 2.5.0 von Greengrass Nucleus aktualisiert.
2.1.3	Die Version wurde für die Version 2.4.0 von Greengrass Nucleus aktualisiert.
2.1.2	Die Version wurde für die Version 2.3.0 von Greengrass Nucleus aktualisiert.
2.1.1	Die Version wurde für die Version 2.2.0 von Greengrass Nucleus aktualisiert.
2.1.0	Erste Version

TensorFlow Lite-Objekterkennung

Die TensorFlow Lite-Objekterkennungskomponente (`aws.greengrass.TensorFlowLiteObjectDetection`) enthält einen Beispiel-Inferenzcode für die Durchführung von Inferenzen zur Objekterkennung mit TensorFlow Lite und ein vortrainiertes Single Shot Detection (SSD) MobileNet 1.0-Modell als Beispiel. Diese Komponente verwendet die Variante TensorFlow Modellspeicher für Lite-Objekterkennung und die TensorFlow Lite-Laufzeit Komponenten als Abhängigkeiten, um TensorFlow Lite und das Beispielmodell herunterzuladen.

Um diese Inferenzkomponente mit einem speziell trainierten TensorFlow Lite-Modell zu verwenden, können Sie eine benutzerdefinierte Version der abhängigen Modellspeicherkomponente erstellen. Um Ihren eigenen benutzerdefinierten Inferenzcode zu verwenden, verwenden Sie das Rezept dieser Komponente als Vorlage, um eine benutzerdefinierte Inferenzkomponente zu erstellen.

Themen

- Versionen

- Typ
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Konfiguration
- Lokale Protokolldatei
- Änderungsprotokoll

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.1.x

Typ

Diese Komponente ist eine generische Komponente () `aws.greengrass.generic`. Der Greengrass-Kern führt die Lebenszyklusskripte der Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf Kerngeräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Auf Greengrass-Core-Geräten, auf denen Amazon Linux 2 oder Ubuntu 18.04 ausgeführt wird, ist die GNU C Library (Glibc) Version 2.27 oder höher auf dem Gerät installiert.
- Auf ARMv7L-Geräten wie Raspberry Pi sind Abhängigkeiten für OpenCV-Python auf dem Gerät installiert. Führen Sie den folgenden Befehl aus, um die Abhängigkeiten zu installieren.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Raspberry Pi-Geräte, auf denen Raspberry Pi OS Bullseye ausgeführt wird, müssen die folgenden Anforderungen erfüllen:
  - NumPy 1.22.4 oder höher auf dem Gerät installiert. Raspberry Pi OS Bullseye enthält eine frühere Version von NumPy, sodass Sie den folgenden Befehl ausführen können, um das Gerät zu aktualisieren NumPy .

    ```
    pip3 install --upgrade numpy
    ```

  - Der ältere Kamerastack ist auf dem Gerät aktiviert. Raspberry Pi OS Bullseye enthält einen neuen Kamerastack, der standardmäßig aktiviert und nicht kompatibel ist. Sie müssen also den älteren Kamerastack aktivieren.

    Um den alten Kamerastack zu aktivieren

    1. Führen Sie den folgenden Befehl aus, um das Raspberry Pi-Konfigurationstool zu öffnen.

       ```
       sudo raspi-config
       ```

    2. Wählen Sie Schnittstellenoptionen.
    3. Wählen Sie Legacy-Kamera aus, um den Legacy-Kamerastack zu aktivieren.
    4. Starten Sie den Raspberry Pi neu.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt sie AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und all ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitstellen zu können. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente sowie die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste der Abhängigkeiten.

2.1.11 and 2.1.12

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.1.11 und 2.1.12 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.13.0	Weich
TensorFlow Lite-Modellspeicher zur Bildklassifizierung	>=2.1.0 <2.2.0	Hart
TensorFlow Leicht	>=2,5,0 <2,6,0	Hart

2.1.10

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.10 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.12.0	Weich
TensorFlow Lite-Modellspeicher zur Bildklassifizierung	>=2.1.0 <2.2.0	Hart
TensorFlow Leicht	>=2,5,0 <2,6,0	Hart

2.1.9

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.9 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.11.0	Weich

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
TensorFlow Lite-Modellspeicher zur Bildklassifizierung	>=2.1.0 <2.2.0	Hart
TensorFlow Leicht	>=2,5,0 <2,6,0	Hart

2.1.8

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.8 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.10.0	Weich
TensorFlow Lite-Modellspeicher zur Bildklassifizierung	>=2.1.0 <2.2.0	Hart
TensorFlow Leicht	>=2,5,0 <2,6,0	Hart

2.1.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.9.0	Weich
TensorFlow Lite-Modellspeicher zur Bildklassifizierung	>=2.1.0 <2.2.0	Hart
TensorFlow Leicht	>=2,5,0 <2,6,0	Hart

2.1.6

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.8.0	Weich
TensorFlow Lite-Modellspeicher zur Bildklassifizierung	>=2.1.0 <2.2.0	Hart
TensorFlow Leicht	>=2,5,0 <2,6,0	Hart

2.1.5

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.7.0	Weich
TensorFlow Lite-Modellspeicher zur Bildklassifizierung	>=2.1.0 <2.2.0	Hart
TensorFlow Leicht	>=2,5,0 <2,6,0	Hart

2.1.4

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.6.0	Weich

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
TensorFlow Lite-Modellspeicher zur Bildklassifizierung	>=2.1.0 <2.2.0	Hart
TensorFlow Leicht	>=2,5,0 <2,6,0	Hart

2.1.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.5.0	Weich
TensorFlow Lite-Modellspeicher zur Bildklassifizierung	>=2.1.0 <2.2.0	Hart
TensorFlow Leicht	>=2,5,0 <2,6,0	Hart

2.1.2

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.4.0	Weich
TensorFlow Lite-Modellspeicher zur Bildklassifizierung	>=2.1.0 <2.2.0	Hart
TensorFlow Leicht	>=2,5,0 <2,6,0	Hart

2.1.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.3.0	Weich
TensorFlow Lite-Modellspeicher zur Bildklassifizierung	>=2.1.0 <2.2.0	Hart
TensorFlow Leicht	>=2,5,0 <2,6,0	Hart

2.1.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.2.0	Weich
TensorFlow Lite-Modellspeicher zur Bildklassifizierung	>=2.1.0 <2.2.0	Hart
TensorFlow Leicht	>=2,5,0 <2,6,0	Hart

Konfiguration

Diese Komponente stellt die folgenden Konfigurationsparameter bereit, die Sie bei der Bereitstellung der Komponente anpassen können.

`accessControl`

(Optional) Das Objekt, das die Autorisierungsrichtlinie enthält, die es der Komponente ermöglicht, Nachrichten unter dem Standardthema für Benachrichtigungen zu veröffentlichen.

Standard:

```
{
   "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.TensorFlowLiteObjectDetection:mqttproxy:1": {
         "policyDescription": "Allows access to publish via topic ml/tflite/object-
detection.",
         "operations": [
            "aws.greengrass#PublishToIoTCore"
         ],
         "resources": [
            "ml/tflite/object-detection"
         ]
      }
   }
}
```

`PublishResultsOnTopic`

(Optional) Das Thema, zu dem Sie die Inferenzergebnisse veröffentlichen möchten. Wenn Sie diesen Wert ändern, müssen Sie auch den Wert von `resources` im `accessControl` Parameter so ändern, dass er Ihrem benutzerdefinierten Themennamen entspricht.

Standard: `ml/tflite/object-detection`

`Accelerator`

Der Beschleuniger, den Sie verwenden möchten. Unterstützte Werte sind `cpu` und `gpu`.

Die Beispielmodelle in der abhängigen Modellkomponente unterstützen nur CPU-Beschleunigung. Um die GPU-Beschleunigung mit einem anderen benutzerdefinierten Modell zu verwenden, erstellen Sie eine benutzerdefinierte Modellkomponente, um die öffentliche Modellkomponente zu überschreiben.

Standard: `cpu`

`ImageDirectory`

(Optional) Der Pfad des Ordners auf dem Gerät, in dem Inferenzkomponenten Bilder lesen. Sie können diesen Wert an einen beliebigen Ort auf Ihrem Gerät ändern, auf den Sie Lese-/Schreibzugriff haben.

Standard: */greengrass/v2*`/packages/artifacts-unarchived/`*component-name*`/object_detection/sample_images/`

> **Note**
>
> Wenn Sie den Wert auf setzen `UseCamera`true, wird dieser Konfigurationsparameter ignoriert.

`ImageName`

(Optional) Der Name des Bilds, das die Inferenzkomponente als Eingabe für eine Make-Vorhersage verwendet. Die Komponente sucht in dem unter angegebenen Ordner nach dem Bild. `ImageDirectory` Standardmäßig verwendet die Komponente das Beispielbild im Standard-Bildverzeichnis. AWS IoT Greengrass unterstützt die folgenden Bildformate: `jpeg`jpg,png, undnpy.

Standard: `objects.jpg`

> **Note**
>
> Wenn Sie den Wert `UseCamera` auf setzen`true`, wird dieser Konfigurationsparameter ignoriert.

`InferenceInterval`

(Optional) Die Zeit in Sekunden zwischen den einzelnen Vorhersagen des Inferenzcodes. Der Beispiel-Inferenzcode wird unbegrenzt ausgeführt und wiederholt seine Vorhersagen im angegebenen Zeitintervall. Sie können dieses Intervall beispielsweise auf ein kürzeres Intervall ändern, wenn Sie mit einer Kamera aufgenommene Bilder für Vorhersagen in Echtzeit verwenden möchten.

Standard: `3600`

`ModelResourceKey`

(Optional) Die Modelle, die in der abhängigen öffentlichen Modellkomponente verwendet werden. Ändern Sie diesen Parameter nur, wenn Sie die Komponente des öffentlichen Modells durch eine benutzerdefinierte Komponente überschreiben.

Standard:

```
{
```

```
    "model": "TensorFlowLite-SSD"
}
```

`UseCamera`

(Optional) Zeichenkettenwert, der definiert, ob Bilder von einer Kamera verwendet werden sollen, die mit dem Greengrass-Core-Gerät verbunden ist. Unterstützte Werte sind `true` und `false`.

Wenn Sie diesen Wert auf setzen`true`, greift der Beispiel-Inferenzcode auf die Kamera auf Ihrem Gerät zu und führt die Inferenz lokal für das aufgenommene Bild aus. Die Werte der `ImageDirectory` Parameter `ImageName` und werden ignoriert. Stellen Sie sicher, dass der Benutzer, der diese Komponente ausführt, Lese-/Schreibzugriff auf den Speicherort hat, an dem die Kamera aufgenommene Bilder speichert.

Standard: `false`

> **Note**
>
> Wenn Sie sich das Rezept für diese Komponente ansehen, erscheint der `UseCamera` Konfigurationsparameter nicht in der Standardkonfiguration. Sie können den Wert dieses Parameters jedoch in einem Update zur Zusammenführung der Konfiguration ändern, wenn Sie die Komponente bereitstellen.
> Wenn Sie `UseCamera` auf setzen`true`, müssen Sie auch einen Symlink erstellen, damit die Inferenzkomponente über die virtuelle Umgebung, die von der Runtime-Komponente erstellt wurde, auf Ihre Kamera zugreifen kann. Weitere Hinweise zur Verwendung einer Kamera mit den Beispiel-Inferenzkomponenten finden Sie unter. Komponentenkonfigurationen aktualisieren

> **Note**
>
> Wenn Sie sich das Rezept für diese Komponente ansehen, wird der `UseCamera` Konfigurationsparameter in der Standardkonfiguration nicht angezeigt. Sie können den Wert dieses Parameters jedoch in einem Update zur Zusammenführung der Konfiguration ändern, wenn Sie die Komponente bereitstellen.
> Wenn Sie `UseCamera` auf setzen`true`, müssen Sie auch einen Symlink erstellen, damit die Inferenzkomponente über die virtuelle Umgebung, die von der Runtime-Komponente erstellt wurde, auf Ihre Kamera zugreifen kann. Weitere Hinweise zur Verwendung einer Kamera

mit den Beispiel-Inferenzkomponenten finden Sie unter. Komponentenkonfigurationen aktualisieren

Lokale Protokolldatei

Diese Komponente verwendet die folgende Protokolldatei.

Linux

```
/greengrass/v2/logs/aws.greengrass.TensorFlowLiteObjectDetection.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.TensorFlowLiteObjectDetection.log
```

Um die Protokolle dieser Komponente einzusehen

- Führen Sie den folgenden Befehl auf dem Kerngerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

  Linux

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

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs
  \aws.greengrass.TensorFlowLiteObjectDetection.log -Tail 10 -Wait
  ```

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
2.1.12	Die Version wurde für die Version Greengrass Nucleus 2.12.5 aktualisiert.
2.1.11	Die Version wurde für die Version 2.12.0 von Greengrass Nucleus aktualisiert.
2.1.10	Die Version wurde für die Version 2.11.0 von Greengrass Nucleus aktualisiert.
2.1.9	Die Version wurde für die Version 2.10.0 von Greengrass Nucleus aktualisiert.
2.1.8	Die Version wurde für die Version 2.9.0 von Greengrass Nucleus aktualisiert.
2.1.7	Die Version wurde für die Version 2.8.0 von Greengrass Nucleus aktualisiert.
2.1.6	Die Version wurde für die Version 2.7.0 von Greengrass Nucleus aktualisiert.
2.1.5	Die Version wurde für die Version 2.6.0 von Greengrass Nucleus aktualisiert.
2.1.4	Die Version wurde für die Version 2.5.0 von Greengrass Nucleus aktualisiert.
2.1.3	Die Version wurde für die Version 2.4.0 von Greengrass Nucleus aktualisiert.
2.1.2	Die Version wurde für die Version 2.3.0 von Greengrass Nucleus aktualisiert.
2.1.1	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem mit der Bildskalierung, das zu ungenauen Begrenzungsrahmen in den Inferenzergebnissen der TensorFlow Lite-Objekterkennung führte.
2.1.0	Erste Version

TensorFlow Modellspeicher für Lite-Bildklassifizierung

Der TensorFlow Lite-Bildklassifizierungsmodellspeicher (`variant.TensorFlowLite.ImageClassification.ModelStore`) ist eine Modellkomponente für maschinelles Lernen, die ein vortrainiertes MobileNet v1-Modell als Greengrass-Artefakt enthält. Das in dieser Komponente verwendete Beispielmodell wird vom TensorFlowHub abgerufen und mit Lite implementiert. TensorFlow

Die TensorFlow Lite-Bildklassifizierung Inferenzkomponente verwendet diese Komponente als Abhängigkeit für die Modellquelle. Um ein benutzerdefiniertes TensorFlow Lite-Modell zu verwenden, erstellen Sie eine benutzerdefinierte Version dieser Modellkomponente und fügen Sie Ihr benutzerdefiniertes Modell als Komponentenartefakt hinzu. Sie können das Rezept dieser Komponente als Vorlage verwenden, um benutzerdefinierte Modellkomponenten zu erstellen.

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.1.x

Typ

Diese Komponente ist eine generische Komponente () `aws.greengrass.generic`. Der Greengrass-Kern führt die Lebenszyklusskripte der Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf Kerngeräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Auf Greengrass-Core-Geräten, auf denen Amazon Linux 2 oder Ubuntu 18.04 ausgeführt wird, ist die GNU C Library (Glibc) Version 2.27 oder höher auf dem Gerät installiert.
- Auf ARMv7L-Geräten wie Raspberry Pi sind Abhängigkeiten für OpenCV-Python auf dem Gerät installiert. Führen Sie den folgenden Befehl aus, um die Abhängigkeiten zu installieren.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- Raspberry Pi-Geräte, auf denen Raspberry Pi OS Bullseye ausgeführt wird, müssen die folgenden Anforderungen erfüllen:
  - NumPy 1.22.4 oder höher auf dem Gerät installiert. Raspberry Pi OS Bullseye enthält eine frühere Version von NumPy, sodass Sie den folgenden Befehl ausführen können, um das Gerät zu aktualisieren NumPy .

    ```
    pip3 install --upgrade numpy
    ```

  - Der ältere Kamerastack ist auf dem Gerät aktiviert. Raspberry Pi OS Bullseye enthält einen neuen Kamerastack, der standardmäßig aktiviert und nicht kompatibel ist. Sie müssen also den älteren Kamerastack aktivieren.

    Um den Legacy-Kamerastack zu aktivieren

    1. Führen Sie den folgenden Befehl aus, um das Raspberry Pi-Konfigurationstool zu öffnen.

       ```
       sudo raspi-config
       ```

    2. Wählen Sie Schnittstellenoptionen.
    3. Wählen Sie Legacy-Kamera aus, um den Legacy-Kamerastack zu aktivieren.
    4. Starten Sie den Raspberry Pi neu.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt sie AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und all ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitstellen zu

können. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente sowie die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können die Abhängigkeiten für jede Version der Komponente auch in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste der Abhängigkeiten.

2.1.11 and 2.1.12

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.1.11 und 2.1.12 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.13.0	Weich

2.1.10

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.10 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.12.0	Weich

2.1.9

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.9 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.11.0	Weich

2.1.8

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.8 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.10.0	Weich

2.1.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.9.0	Weich

2.1.6

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.8.0	Weich

2.1.5

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.7.0	Weich

2.1.4

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.6.0	Weich

2.1.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.5.0	Weich

2.1.2

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.4.0	Weich

2.1.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.3.0	Weich

2.1.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.2.0	Weich

Konfiguration

Diese Komponente hat keine Konfigurationsparameter.

Lokale Protokolldatei

Diese Komponente gibt keine Protokolle aus.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
2.1.12	Die Version wurde für die Version Greengrass Nucleus 2.12.5 aktualisiert.
2.1.11	Die Version wurde für die Version 2.12.0 von Greengrass Nucleus aktualisiert.
2.1.10	Die Version wurde für die Version 2.11.0 von Greengrass Nucleus aktualisiert.
2.1.9	Die Version wurde für die Version 2.10.0 von Greengrass Nucleus aktualisiert.
2.1.8	Die Version wurde für die Version 2.9.0 von Greengrass Nucleus aktualisiert.
2.1.7	Die Version wurde für die Version 2.8.0 von Greengrass Nucleus aktualisiert.
2.1.6	Die Version wurde für die Version 2.7.0 von Greengrass Nucleus aktualisiert.
2.1.5	Die Version wurde für die Version 2.6.0 von Greengrass Nucleus aktualisiert.
2.1.4	Die Version wurde für die Version 2.5.0 von Greengrass Nucleus aktualisiert.
2.1.3	Die Version wurde für die Version 2.4.0 von Greengrass Nucleus aktualisiert.
2.1.2	Die Version wurde für die Version 2.3.0 von Greengrass Nucleus aktualisiert.
2.1.1	Die Version wurde für die Version 2.2.0 von Greengrass Nucleus aktualisiert.
2.1.0	Erste Version

TensorFlow Modellspeicher für Lite-Objekterkennung

Der TensorFlow Lite-Objekterkennungsmodellspeicher (`variant.TensorFlowLite.ObjectDetection.ModelStore`) ist eine Modellkomponente

für maschinelles Lernen, die ein vortrainiertes Single Shot Detection (SSD) MobileNet -Modell als Greengrass-Artefakt enthält. Das in dieser Komponente verwendete Beispielmodell wird vom TensorFlow Hub abgerufen und mit Lite implementiert. TensorFlow

Die TensorFlow Lite-Inferenzkomponente zur Objekterkennung verwendet diese Komponente als Abhängigkeit für die Modellquelle. Um ein benutzerdefiniertes TensorFlow Lite-Modell zu verwenden, erstellen Sie eine benutzerdefinierte Version dieser Modellkomponente und fügen Sie Ihr benutzerdefiniertes Modell als Komponentenartefakt hinzu. Sie können das Rezept dieser Komponente als Vorlage verwenden, um benutzerdefinierte Modellkomponenten zu erstellen.

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.1.x

Typ

Diese Komponente ist eine generische Komponente () `aws.greengrass.generic`. Der Greengrass-Kern führt die Lebenszyklusskripte der Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf Kerngeräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Auf Greengrass-Core-Geräten, auf denen Amazon Linux 2 oder Ubuntu 18.04 ausgeführt wird, ist die GNU C Library (Glibc) Version 2.27 oder höher auf dem Gerät installiert.
- Auf ARMv7L-Geräten wie Raspberry Pi sind Abhängigkeiten für OpenCV-Python auf dem Gerät installiert. Führen Sie den folgenden Befehl aus, um die Abhängigkeiten zu installieren.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- Raspberry Pi-Geräte, auf denen Raspberry Pi OS Bullseye ausgeführt wird, müssen die folgenden Anforderungen erfüllen:
  - NumPy 1.22.4 oder höher auf dem Gerät installiert. Raspberry Pi OS Bullseye enthält eine frühere Version von NumPy, sodass Sie den folgenden Befehl ausführen können, um das Gerät zu aktualisieren NumPy .

    ```
    pip3 install --upgrade numpy
    ```

  - Der ältere Kamerastack ist auf dem Gerät aktiviert. Raspberry Pi OS Bullseye enthält einen neuen Kamerastack, der standardmäßig aktiviert und nicht kompatibel ist. Sie müssen also den älteren Kamerastack aktivieren.

    Um den alten Kamerastack zu aktivieren

    1. Führen Sie den folgenden Befehl aus, um das Raspberry Pi-Konfigurationstool zu öffnen.

       ```
       sudo raspi-config
       ```

    2. Wählen Sie Schnittstellenoptionen.
    3. Wählen Sie Legacy-Kamera aus, um den Legacy-Kamerastack zu aktivieren.
    4. Starten Sie den Raspberry Pi neu.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt sie AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und all ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitstellen zu können. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente sowie die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste der Abhängigkeiten.

2.1.11 and 2.1.12

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.1.11 und 2.1.12 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.13.0	Weich

2.1.10

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.10 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.12.0	Weich

2.1.9

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.9 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.11.0	Weich

2.1.8

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.8 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.10.0	Weich

2.1.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.9.0	Weich

2.1.6

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.8.0	Weich

2.1.5

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.7.0	Weich

2.1.4

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.6.0	Weich

2.1.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.5.0	Weich

2.1.2

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.4.0	Weich

2.1.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.3.0	Weich

2.1.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.2.0	Weich

Konfiguration

Diese Komponente hat keine Konfigurationsparameter.

Lokale Protokolldatei

Diese Komponente gibt keine Protokolle aus.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
2.1.12	Die Version wurde für die Version Greengrass Nucleus 2.12.5 aktualisiert.
2.1.11	Die Version wurde für die Version 2.12.0 von Greengrass Nucleus aktualisiert.
2.1.10	Die Version wurde für die Version 2.11.0 von Greengrass Nucleus aktualisiert.
2.1.9	Die Version wurde für die Version 2.10.0 von Greengrass Nucleus aktualisiert.
2.1.8	Die Version wurde für die Version 2.9.0 von Greengrass Nucleus aktualisiert.
2.1.7	Die Version wurde für die Version 2.8.0 von Greengrass Nucleus aktualisiert.
2.1.6	Die Version wurde für die Version 2.7.0 von Greengrass Nucleus aktualisiert.
2.1.5	Die Version wurde für die Version 2.6.0 von Greengrass Nucleus aktualisiert.
2.1.4	Die Version wurde für die Version 2.5.0 von Greengrass Nucleus aktualisiert.
2.1.3	Die Version wurde für die Version 2.4.0 von Greengrass Nucleus aktualisiert.
2.1.2	Die Version wurde für die Version 2.3.0 von Greengrass Nucleus aktualisiert.
2.1.1	Die Version wurde für die Version 2.2.0 von Greengrass Nucleus aktualisiert.
2.1.0	Erste Version

TensorFlow Lite-Laufzeit

Die TensorFlow Lite-Runtime-Komponente (`variant.TensorFlowLite`) enthält ein Skript, das die TensorFlow Lite-Version 2.5.0 und ihre Abhängigkeiten in einer virtuellen Umgebung auf Ihrem Gerät installiert. Die TensorFlow Lite-Komponente zur Bildklassifizierung und TensorFlow Lite-Objekterkennung verwendet diese Laufzeitkomponente als Abhängigkeit für die Installation von TensorFlow Lite.

> **Note**
>
> TensorFlow Die Lite-Runtime-Komponente v2.5.6 und höher installiert bestehende Installationen der TensorFlow Lite-Runtime und ihrer Abhängigkeiten neu. Durch diese Neuinstallation wird sichergestellt, dass auf dem Kerngerät kompatible Versionen von TensorFlow Lite und seinen Abhängigkeiten ausgeführt werden.

Um eine andere Laufzeit zu verwenden, können Sie das Rezept dieser Komponente als Vorlage verwenden, um eine benutzerdefinierte Komponente für maschinelles Lernen zu erstellen.

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.5.x

Typ

Diese Komponente ist eine generische Komponente () `aws.greengrass.generic`. Der Greengrass-Kern führt die Lebenszyklusskripte der Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf Kerngeräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Auf Greengrass-Core-Geräten, auf denen Amazon Linux 2 oder Ubuntu 18.04 ausgeführt wird, ist die GNU C Library (Glibc) Version 2.27 oder höher auf dem Gerät installiert.
- Auf ARMv7L-Geräten wie Raspberry Pi sind Abhängigkeiten für OpenCV-Python auf dem Gerät installiert. Führen Sie den folgenden Befehl aus, um die Abhängigkeiten zu installieren.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- Raspberry Pi-Geräte, auf denen Raspberry Pi OS Bullseye ausgeführt wird, müssen die folgenden Anforderungen erfüllen:
  - NumPy 1.22.4 oder höher auf dem Gerät installiert. Raspberry Pi OS Bullseye enthält eine frühere Version von NumPy, sodass Sie den folgenden Befehl ausführen können, um das Gerät zu aktualisieren NumPy .

    ```
    pip3 install --upgrade numpy
    ```

  - Der ältere Kamerastack ist auf dem Gerät aktiviert. Raspberry Pi OS Bullseye enthält einen neuen Kamerastack, der standardmäßig aktiviert und nicht kompatibel ist. Sie müssen also den älteren Kamerastack aktivieren.

Um den alten Kamerastack zu aktivieren

1. Führen Sie den folgenden Befehl aus, um das Raspberry Pi-Konfigurationstool zu öffnen.

   ```
   sudo raspi-config
   ```

2. Wählen Sie Schnittstellenoptionen.
3. Wählen Sie Legacy-Kamera aus, um den Legacy-Kamerastack zu aktivieren.
4. Starten Sie den Raspberry Pi neu.

Endpunkte und Anschlüsse

Standardmäßig verwendet diese Komponente ein Installationsskript, um Pakete mit den `pip` Befehlen`apt`, `yum``brew`, und zu installieren, je nachdem, welche Plattform das Kerngerät verwendet. Diese Komponente muss in der Lage sein, ausgehende Anfragen an verschiedene Paketindizes und Repositorys auszuführen, um das Installationsskript auszuführen. Damit der ausgehende Datenverkehr dieser Komponente über einen Proxy oder eine Firewall übertragen werden kann, müssen Sie die Endpunkte für die Paketindizes und Repositorys identifizieren, zu denen Ihr Kerngerät eine Verbindung zur Installation herstellt.

Beachten Sie bei der Identifizierung von Endpunkten, die für das Installationsskript dieser Komponente erforderlich sind, Folgendes:

- Die Endpunkte hängen von der Plattform des Kerngeräts ab. Beispielsweise verwendet ein Kerngerät, auf dem Ubuntu ausgeführt wird, `apt` eher als `yum` oder`brew`. Darüber hinaus haben Geräte, die denselben Paketindex verwenden, möglicherweise unterschiedliche Quelllisten, sodass sie Pakete aus verschiedenen Repositorys abrufen können.
- Die Endpunkte können sich bei mehreren Geräten, die denselben Paketindex verwenden, unterscheiden, da jedes Gerät über eigene Quelllisten verfügt, die definieren, wo Pakete abgerufen werden sollen.
- Die Endpunkte können sich im Laufe der Zeit ändern. Jeder Paketindex enthält die URLs der Repositorys, in die Sie Pakete herunterladen, und der Besitzer eines Pakets kann ändern, welche URLs der Paketindex bereitstellt.

Weitere Informationen zu den Abhängigkeiten, die diese Komponente installiert, und zur Deaktivierung des Installationsskripts finden Sie im UseInstallerKonfigurationsparameter.

Weitere Informationen zu Endpunkten und Ports, die für den Basisbetrieb erforderlich sind, finden Sie unterZulassen von Gerätedatenverkehr über einen Proxy oder eine Firewall.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und all ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitstellen zu können. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente sowie die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste der Abhängigkeiten.

2.5.14 and 2.5.15

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.5.14 und 2.5.15 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.13.0	Weich

2.5.13

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.5.13 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.12.0	Weich

2.5.12

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.5.12 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.11.0	Weich

2.5.11

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.5.11 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.10.0	Weich

2.5.10

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.5.10 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.9.0	Weich

2.5.9

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.5.9 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.8.0	Weich

2.5.8

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.5.8 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.7.0	Weich

2.5.5 - 2.5.7

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.5.5 bis 2.5.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.6.0	Weich

2.5.3 and 2.5.4

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.5.3 und 2.5.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.5.0	Weich

2.5.2

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.5.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.4.0	Weich

2.5.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.5.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.3.0	Weich

2.5.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.5.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.2.0	Weich

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zu den Komponentenrezepten.

Konfiguration

Diese Komponente stellt die folgenden Konfigurationsparameter bereit, die Sie bei der Bereitstellung der Komponente anpassen können.

`MLRootPath`

(Optional) Der Pfad des Ordners auf Linux-Kerngeräten, in dem Inferenzkomponenten Bilder lesen und Inferenzergebnisse schreiben. Sie können diesen Wert in einen beliebigen Speicherort auf Ihrem Gerät ändern, auf den der Benutzer, der diese Komponente ausführt, Lese-/Schreibzugriff hat.

Standard: `/greengrass/v2/work/variant.TensorFlowLite/greengrass_ml`

`WindowsMLRootPath`

Diese Funktion ist in Version 1.6.6 und höher dieser Komponente verfügbar.

(Optional) Der Pfad des Ordners auf dem Windows Core-Gerät, in dem Inferenzkomponenten Bilder lesen und Inferenzergebnisse schreiben. Sie können diesen Wert in einen beliebigen Speicherort auf Ihrem Gerät ändern, auf den der Benutzer, der diese Komponente ausführt, Lese-/Schreibzugriff hat.

Standard: `C:\greengrass\v2\\work\\variant.DLR\\greengrass_ml`

`UseInstaller`

(Optional) Zeichenkettenwert, der definiert, ob das Installationsskript in dieser Komponente verwendet werden soll, um TensorFlow Lite und seine Abhängigkeiten zu installieren. Unterstützte Werte sind `true` und `false`.

Legen Sie diesen Wert auf fest, `false` wenn Sie ein benutzerdefiniertes Skript für die TensorFlow Lite-Installation verwenden möchten oder wenn Sie Laufzeitabhängigkeiten in ein vorgefertigtes Linux-Image aufnehmen möchten. Um diese Komponente mit den von AWS-bereitgestellten TensorFlow Lite-Inferenzkomponenten zu verwenden, installieren Sie die folgenden Bibliotheken, einschließlich aller Abhängigkeiten, und stellen Sie sie dem Systembenutzer zur Verfügung, z. B. `ggc_user` dem Systembenutzer, der die ML-Komponenten ausführt.

- Python 3.8 oder höher, auch `pip` für Ihre Version von Python
- TensorFlow Lite v2.5.0
- NumPy
- Öffnen Sie CV-Python
- AWS IoT Device SDK v2 für Python
- AWS Common Runtime (CRT) Python
- Picamera (für Raspberry Pi-Geräte)
- `awscamModul` (für AWS DeepLens Geräte)
- LibGL (für Linux-Geräte)

Standard: `true`

Verwendung

Verwenden Sie diese Komponente mit dem `UseInstaller` Konfigurationsparameter auf, `true` um TensorFlow Lite und seine Abhängigkeiten auf Ihrem Gerät zu installieren. Die Komponente richtet eine virtuelle Umgebung auf Ihrem Gerät ein, die OpenCV und die NumPy Bibliotheken enthält, die für TensorFlow Lite erforderlich sind.

> **Note**
>
> Das Installationsskript in dieser Komponente installiert auch die neuesten Versionen zusätzlicher Systembibliotheken, die für die Konfiguration der virtuellen Umgebung auf Ihrem Gerät und die Verwendung des installierten Frameworks für maschinelles Lernen erforderlich sind. Dadurch könnten die vorhandenen Systembibliotheken auf

Ihrem Gerät aktualisiert werden. In der folgenden Tabelle finden Sie eine Liste der Bibliotheken, die diese Komponente für jedes unterstützte Betriebssystem installiert. Wenn Sie diesen Installationsvorgang anpassen möchten, setzen Sie den `UseInstaller` Konfigurationsparameter auf `false` und entwickeln Sie Ihr eigenes Installationsskript.

Plattform	Auf dem Gerätesystem installierte Bibliotheken	In der virtuellen Umgebung installierte Bibliotheken
Armv7l	`build-essential` , `cmake`, `ca-certificates` , `git`	`setuptools` , `wheel`
Amazon Linux 2	`mesa-libGL`	None
Ubuntu	`wget`	None

Wenn Sie Ihre Inferenzkomponente bereitstellen, überprüft diese Runtime-Komponente zunächst, ob TensorFlow Lite und die zugehörigen Abhängigkeiten auf Ihrem Gerät bereits installiert sind. Wenn nicht, installiert die Runtime-Komponente sie für Sie.

Lokale Protokolldatei

Diese Komponente verwendet die folgende Protokolldatei.

Linux

```
/greengrass/v2/logs/variant.TensorFlowLite.log
```

Windows

```
C:\greengrass\v2\logs\variant.TensorFlowLite.log
```

Um die Protokolle dieser Komponente einzusehen

- Führen Sie den folgenden Befehl auf dem Kerngerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

Linux

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

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\variant.TensorFlowLite.log -Tail 10 -Wait
```

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
2.5.15	Version für Greengrass Nucleus 2.12.5 aktualisiert.
2.5.14	Die Version wurde für die Version 2.12.0 von Greengrass Nucleus aktualisiert.
2.5.13	Die Version wurde für die Version 2.11.0 von Greengrass Nucleus aktualisiert.
2.5.12	Die Version wurde für die Version 2.10.0 von Greengrass Nucleus aktualisiert.
2.5.11	Die Version wurde für die Version 2.9.0 von Greengrass Nucleus aktualisiert.
2.5.10	Die Version wurde für die Version 2.8.0 von Greengrass Nucleus aktualisiert.
2.5.9	Die Version wurde für die Version 2.7.0 von Greengrass Nucleus aktualisiert.
2.5.8	Die Version wurde für die Version 2.6.0 von Greengrass Nucleus aktualisiert.
2.5.7	Fehlerkorrekturen und Verbesserungen • Aktualisiert das `UseInstaller` Installationsskript zur Installation von LibGL, das auf bestimmten Linux-Plattformen standardmäßig nicht verfügbar ist. • Aktualisiert das `UseInstaller` Installationsskript, sodass es in der virtuellen Umgebung dieser Komponente immer Python 3.9 verwendet.

Version	Änderungen
	Diese Änderung trägt dazu bei, die Kompatibilität mit anderen Bibliothe ken sicherzustellen.
2.5.6	Fehlerkorrekturen und Verbesserungen • Aktualisiert diese Komponente, um den neuesten Patch von TensorFlow Lite 2.5.0 (`tflite-runtime-2.5.0.post1`) zu installieren, sodass Sie diese Komponente mit Python 3.9 verwenden können. Wenn diese Komponente diesen Patch nicht installieren kann, wird sie `tflite-ru ntime-2.5.0` stattdessen installiert. • Aktualisiert diese Komponente, um bestehende Installationen von TensorFlow Lite und seinen Abhängigkeiten neu zu installieren. Durch diese Änderung wird sichergestellt, dass auf dem Kerngerät kompatible Versionen von TensorFlow Lite und seinen Abhängigkeiten ausgeführt werden.
2.5.5	Neue Features • Fügt Unterstützung für Kerngeräte hinzu, auf denen Windows ausgeführ t wird. • Fügt den neuen `WindowsMLRootPath` Konfigurationsparameter hinzu, mit dem Sie den Ordner mit den Inferenzergebnissen auf Windows-Core-Geräten konfigurieren können.
2.5.4	Neue Features • Fügt den neuen `UseInstaller` Konfigurationsparameter hinzu, mit dem Sie das Installationsskript in dieser Komponente deaktivieren können.
2.5.3	Die Version wurde für die Version 2.4.0 von Greengrass Nucleus aktualisiert.
2.5.2	Die Version wurde für die Version 2.3.0 von Greengrass Nucleus aktualisiert.
2.5.1	Die Version wurde für die Version 2.2.0 von Greengrass Nucleus aktualisiert.
2.5.0	Erste Version

Adapter für das Bol-RTU-Protokoll

Die microSD-RTU-Protokolladapterkomponente (`aws.greengrass.Modbus`) fragt Informationen von lokalen RTU-Geräten ab.

Um Informationen von einem lokalen Bol RTU-Gerät mit dieser Komponente anzufordern, veröffentlichen Sie eine Nachricht zu dem Thema, in dem diese Komponente abonniert wird. Geben Sie in der Nachricht die Bol RTU-Anforderung an, die an ein Gerät gesendet werden soll. Anschließend veröffentlicht diese Komponente eine Antwort, die das Ergebnis der Bol-RTU-Anforderung enthält.

> **Note**
>
> Diese Komponente bietet ähnliche Funktionen wie der Bol RTU-Protokolladapter-Konnektor in AWS IoT Greengrass V1. Weitere Informationen finden Sie unter Bol RTU-Protokolladapter-Konnektor im AWS IoT Greengrass V1-Entwicklerhandbuch.

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.1.x
- 2.0.x

Typ

Diese Komponente ist eine Lambda-Komponente (`aws.greengrass.lambda`). Der Greengrass-Kern führt die Lambda-Funktion dieser Komponente mit der Lambda-Launcher-Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann nur auf Linux-Core-Geräten installiert werden.

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Ihr Core-Gerät muss die Anforderungen für die Ausführung von Lambda-Funktionen erfüllen. Wenn Sie möchten, dass das Core-Gerät containerisierte Lambda-Funktionen ausführt, muss das Gerät die Voraussetzungen dafür erfüllen. Weitere Informationen finden Sie unter Anforderungen an die Lambda-Funktion.
- Python Version 3.7 ist auf dem Core-Gerät installiert und der PATH-Umgebungsvariablen hinzugefügt.
- Eine physische Verbindung zwischen dem AWS IoT Greengrass Core-Gerät und den Bol-Geräten. Das Core-Gerät muss physisch über einen seriellen Port, z. B. einen USB-Port, mit dem Bol RTU-Netzwerk verbunden sein.
- Um Ausgabedaten von dieser Komponente zu erhalten, müssen Sie das folgende Konfigurationsupdate für die Legacy-Abonnement-Routerkomponente (`aws.greengrass.LegacySubscriptionRouter`) zusammenführen, wenn Sie diese Komponente bereitstellen. Diese Konfiguration gibt das Thema an, in dem diese Komponente Antworten veröffentlicht.

  Legacy subscription router v2.1.x

```
{
  "subscriptions": {
    "aws-greengrass-modbus": {
      "id": "aws-greengrass-modbus",
```

```
      "source": "component:aws.greengrass.Modbus",
      "subject": "modbus/adapter/response",
      "target": "cloud"
    }
  }
}
```

Legacy subscription router v2.0.x

```
{
  "subscriptions": {
    "aws-greengrass-modbus": {
      "id": "aws-greengrass-modbus",
      "source": "arn:aws:lambda:region:aws:function:aws-greengrass-
modbus:version",
      "subject": "modbus/adapter/response",
      "target": "cloud"
    }
  }
}
```

- Ersetzen Sie *region* durch die AWS-Region , die Sie verwenden.
- Ersetzen Sie *Version* durch die Version der Lambda-Funktion, die diese Komponente ausführt. Um die Version der Lambda-Funktion zu finden, müssen Sie das Rezept für die Version dieser Komponente anzeigen, die Sie bereitstellen möchten. Öffnen Sie die Detailseite dieser Komponente in der AWS IoT Greengrass Konsole und suchen Sie nach dem Schlüssel-Wert-Paar der Lambda-Funktion. Dieses Schlüssel-Wert-Paar enthält den Namen und die Version der Lambda-Funktion.

> **Important**
>
> Sie müssen die Lambda-Funktionsversion auf dem Legacy-Abonnement-Router jedes Mal aktualisieren, wenn Sie diese Komponente bereitstellen. Dadurch wird sichergestellt, dass Sie die richtige Lambda-Funktionsversion für die von Ihnen bereitgestellte Komponentenversion verwenden.

Weitere Informationen finden Sie unter Erstellen von Bereitstellungen.

- Der Bol-RTU-Protokolladapter wird für die Ausführung in einer VPC unterstützt.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und alle ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitzustellen. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente und die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste Abhängigkeiten.

2.1.8

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.8 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.13.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.1.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.12.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.1.6

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.11.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.1.4 and 2.1.5

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.1.4 und 2.1.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.10.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.1.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.9.0	Hart
Lambda-Launcher	^2.0.0	Hart

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.1.2

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.8.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.1.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.7.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.0.8 and 2.1.0

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.0.8 und 2.1.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.6.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.0.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.5.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.0.6

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.4.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.0.5

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.3.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.0.4

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.2.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.0.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.3 <2.1.0	Hart
Lambda-Launcher	>=1.0.0	Hart
Lambda-Laufzeiten	>=1.0.0	Weich

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Token-Exchange-Service	>=1.0.0	Hart

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zum Komponentenrezept .

Konfiguration

Diese Komponente bietet die folgenden Konfigurationsparameter, die Sie anpassen können, wenn Sie die Komponente bereitstellen.

> **Note**
>
> Die Standardkonfiguration dieser Komponente enthält Lambda-Funktionsparameter. Wir empfehlen Ihnen, nur die folgenden Parameter zu bearbeiten, um diese Komponente auf Ihren Geräten zu konfigurieren.

v2.1.x

`lambdaParams`

Ein Objekt, das die Parameter für die Lambda-Funktion dieser Komponente enthält. Dieses Objekt enthält die folgenden Informationen:

`EnvironmentVariables`

Ein Objekt, das die Parameter der Lambda-Funktion enthält. Dieses Objekt enthält die folgenden Informationen:

`ModbusLocalPort`

Der absolute Pfad zum physischen seriellen Port auf dem Core-Gerät, z. B. `/dev/ttyS2`.

Um diese Komponente in einem Container auszuführen, müssen Sie diesen Pfad als Systemgerät (in `containerParams.devices`) definieren, auf das die Komponente zugreifen kann. Diese Komponente wird standardmäßig in einem Container ausgeführt.

> **Note**
>
> Diese Komponente muss über Lese-/Schreibzugriff auf das Gerät verfügen.

`ModbusBaudRate`

(Optional) Ein Zeichenfolgenwert, der die Baudrate für die serielle Kommunikation mit lokalen TCP-Geräten angibt.

Standard: `9600`

`ModbusByteSize`

(Optional) Ein Zeichenfolgenwert, der die Größe eines Bytes in der seriellen Kommunikation mit lokalen TCP-Geräten angibt. Wählen Sie die 8 Bits `5`, `67`, oder aus.

Standard: `8`

`ModbusParity`

(Optional) Der Paritätsmodus, der verwendet werden soll, um die Datenintegrität bei der seriellen Kommunikation mit lokalen TCP-Geräten zu überprüfen.

- `E` – Überprüfen Sie die Datenintegrität mit gerader Parität.
- `O` – Überprüfen Sie die Datenintegrität mit ungerade Parität.
- `N` – Überprüfen Sie nicht die Datenintegrität.

Standard: `N`

`ModbusStopBits`

(Optional) Ein Zeichenfolgenwert, der die Anzahl der Bits angibt, die das Ende eines Bytes in der seriellen Kommunikation mit lokalen TCP-Geräten angeben.

Standard: `1`

`containerMode`

(Optional) Der Containerisierungsmodus für diese Komponente. Wählen Sie aus den folgenden Optionen aus:

- `GreengrassContainer` – Die Komponente wird in einer isolierten Laufzeitumgebung innerhalb des AWS IoT Greengrass Containers ausgeführt.

Wenn Sie diese Option angeben, müssen Sie ein Systemgerät (in `containerParams.devices`) angeben, um dem Container Zugriff auf das microSD-Gerät zu gewähren.

- `NoContainer` – Die Komponente wird nicht in einer isolierten Laufzeitumgebung ausgeführt.

Standard: `GreengrassContainer`

`containerParams`

(Optional) Ein Objekt, das die Containerparameter für diese Komponente enthält. Die Komponente verwendet diese Parameter, wenn Sie `GreengrassContainer` für angeben`containerMode`.

Dieses Objekt enthält die folgenden Informationen:

`memorySize`

(Optional) Die Menge an Arbeitsspeicher (in Kilobyte), die der Komponente zugewiesen werden soll.

Der Standardwert ist 512 MB (525.312 KB).

`devices`

(Optional) Ein Objekt, das die Systemgeräte angibt, auf die die Komponente in einem Container zugreifen kann.

> **Important**
>
> Um diese Komponente in einem Container auszuführen, müssen Sie das Systemgerät angeben, das Sie in der `ModbusLocalPort` Umgebungsvariablen konfigurieren.

Dieses Objekt enthält die folgenden Informationen:

`0` – Dies ist ein Array-Index als Zeichenfolge.

Ein Objekt, das die folgenden Informationen enthält:

`path`

Der Pfad zum Systemgerät auf dem Core-Gerät. Dieser muss denselben Wert haben wie der Wert, den Sie für konfigurieren`ModbusLocalPort`.

`permission`

(Optional) Die Berechtigung für den Zugriff auf das Systemgerät vom Container aus. Dieser Wert muss sein`rw`, was angibt, dass die Komponente Lese-/Schreibzugriff auf das Systemgerät hat.

Standard: `rw`

`addGroupOwner`

(Optional) Gibt an, ob die Systemgruppe, die die Komponente ausführt, als Besitzer des Systemgeräts hinzugefügt werden soll oder nicht.

Standard: `true`

`pubsubTopics`

(Optional) Ein Objekt, das die Themen enthält, in denen die Komponente den Empfang von Nachrichten abonniert. Sie können jedes Thema angeben und angeben, ob die Komponente MQTT-Themen von AWS IoT Core oder lokale Veröffentlichungs-/Abonnementthemen abonniert.

Dieses Objekt enthält die folgenden Informationen:

`0` – Dies ist ein Array-Index als Zeichenfolge.

Ein Objekt, das die folgenden Informationen enthält:

`type`

(Optional) Der Typ des Veröffentlichungs-/Abonnement-Messagings, den diese Komponente zum Abonnieren von Nachrichten verwendet. Wählen Sie aus den folgenden Optionen aus:

- `PUB_SUB` — Abonnieren Sie lokale Veröffentlichen/Abonnement-Nachrichten. Wenn Sie diese Option wählen, darf das Thema keine MQTT-Platzhalter enthalten. Weitere Informationen zum Senden von Nachrichten von einer benutzerdefinierten Komponente, wenn Sie diese Option angeben, finden Sie unter Lokale Nachrichten veröffentlichen/abonnieren.

- `IOT_CORE` – Abonnieren Sie AWS IoT Core MQTT-Nachrichten. Wenn Sie diese Option wählen, kann das Thema MQTT-Platzhalter enthalten. Weitere Informationen zum Senden von Nachrichten von benutzerdefinierten Komponenten, wenn Sie diese Option angeben, finden Sie unter MQTT-Nachrichten veröffentlichen/abonnieren AWS IoT Core.

Standard: `PUB_SUB`

`topic`

(Optional) Das Thema, das die Komponente abonniert, um Nachrichten zu empfangen. Wenn Sie `IotCore` für angeben`type`, können Sie in diesem Thema MQTT-Platzhalter (+ und #) verwenden.

Example Beispiel: Aktualisierung der Konfigurationszusammenführung (Container-Modus)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "GreengrassContainer",
  "containerParams": {
    "devices": {
      "0": {
        "path": "/dev/ttyS2",
        "permission": "rw",
        "addGroupOwner": true
      }
    }
  }
}
```

Example Beispiel: Aktualisierung der Konfigurationszusammenführung (kein Containermodus)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
```

```
  },
  "containerMode": "NoContainer"
}
```

v2.0.x

`lambdaParams`

Ein Objekt, das die Parameter für die Lambda-Funktion dieser Komponente enthält. Dieses Objekt enthält die folgenden Informationen:

`EnvironmentVariables`

Ein Objekt, das die Parameter der Lambda-Funktion enthält. Dieses Objekt enthält die folgenden Informationen:

`ModbusLocalPort`

Der absolute Pfad zum physischen seriellen Port auf dem Core-Gerät, z. B. `/dev/ttyS2`.

Um diese Komponente in einem Container auszuführen, müssen Sie diesen Pfad als Systemgerät (in `containerParams.devices`) definieren, auf das die Komponente zugreifen kann. Diese Komponente wird standardmäßig in einem Container ausgeführt.

> **Note**
>
> Diese Komponente muss über Lese-/Schreibzugriff auf das Gerät verfügen.

`containerMode`

(Optional) Der Containerisierungsmodus für diese Komponente. Wählen Sie aus den folgenden Optionen aus:

- `GreengrassContainer` – Die Komponente wird in einer isolierten Laufzeitumgebung innerhalb des AWS IoT Greengrass Containers ausgeführt.

  Wenn Sie diese Option angeben, müssen Sie ein Systemgerät (in `containerParams.devices`) angeben, um dem Container Zugriff auf das microSD-Gerät zu gewähren.

- `NoContainer` – Die Komponente wird nicht in einer isolierten Laufzeitumgebung ausgeführt.

Standard: `GreengrassContainer`

`containerParams`

(Optional) Ein Objekt, das die Containerparameter für diese Komponente enthält. Die Komponente verwendet diese Parameter, wenn Sie `GreengrassContainer` für angeben`containerMode`.

Dieses Objekt enthält die folgenden Informationen:

`memorySize`

(Optional) Die Menge an Arbeitsspeicher (in Kilobyte), die der Komponente zugewiesen werden soll.

Der Standardwert ist 512 MB (525.312 KB).

`devices`

(Optional) Ein Objekt, das die Systemgeräte angibt, auf die die Komponente in einem Container zugreifen kann.

> **Important**
>
> Um diese Komponente in einem Container auszuführen, müssen Sie das Systemgerät angeben, das Sie in der `ModbusLocalPort` Umgebungsvariablen konfigurieren.

Dieses Objekt enthält die folgenden Informationen:

`0` – Dies ist ein Array-Index als Zeichenfolge.

Ein Objekt, das die folgenden Informationen enthält:

`path`

Der Pfad zum Systemgerät auf dem Core-Gerät. Dieser muss denselben Wert haben wie der Wert, den Sie für konfigurieren`ModbusLocalPort`.

`permission`

(Optional) Die Berechtigung für den Zugriff auf das Systemgerät vom Container aus. Dieser Wert muss sein`rw`, was angibt, dass die Komponente Lese-/Schreibzugriff auf das Systemgerät hat.

Standard: `rw`

`addGroupOwner`

(Optional) Gibt an, ob die Systemgruppe, die die Komponente ausführt, als Besitzer des Systemgeräts hinzugefügt werden soll oder nicht.

Standard: `true`

`pubsubTopics`

(Optional) Ein Objekt, das die Themen enthält, in denen die Komponente den Empfang von Nachrichten abonniert. Sie können jedes Thema angeben und angeben, ob die Komponente MQTT-Themen von AWS IoT Core oder lokale Veröffentlichungs-/Abonnementthemen abonniert.

Dieses Objekt enthält die folgenden Informationen:

`0` – Dies ist ein Array-Index als Zeichenfolge.

Ein Objekt, das die folgenden Informationen enthält:

`type`

(Optional) Der Typ des Veröffentlichungs-/Abonnement-Messagings, den diese Komponente zum Abonnieren von Nachrichten verwendet. Wählen Sie aus den folgenden Optionen aus:

- `PUB_SUB` — Abonnieren Sie lokale Veröffentlichen/Abonnement-Nachrichten. Wenn Sie diese Option wählen, darf das Thema keine MQTT-Platzhalter enthalten. Weitere Informationen zum Senden von Nachrichten von einer benutzerdefinierten Komponente, wenn Sie diese Option angeben, finden Sie unter Lokale Nachrichten veröffentlichen/abonnieren.
- `IOT_CORE` – Abonnieren Sie AWS IoT Core MQTT-Nachrichten. Wenn Sie diese Option wählen, kann das Thema MQTT-Platzhalter enthalten. Weitere Informationen zum Senden von Nachrichten von benutzerdefinierten Komponenten, wenn Sie diese

Option angeben, finden Sie unter MQTT-Nachrichten veröffentlichen/abonnieren AWS IoT Core.

Standard: PUB_SUB

`topic`

(Optional) Das Thema, das die Komponente abonniert, um Nachrichten zu empfangen. Wenn Sie `IotCore` für angeben`type`, können Sie in diesem Thema MQTT-Platzhalter (+ und #) verwenden.

Example Beispiel: Aktualisierung der Konfigurationszusammenführung (Container-Modus)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "GreengrassContainer",
  "containerParams": {
    "devices": {
      "0": {
        "path": "/dev/ttyS2",
        "permission": "rw",
        "addGroupOwner": true
      }
    }
  }
}
```

Example Beispiel: Aktualisierung der Konfigurationszusammenführung (kein Containermodus)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "NoContainer"
}
```

Eingabedaten

Diese Komponente akzeptiert Bol RTU-Anforderungsparameter zum folgenden Thema und sendet die RTU-Anforderung an das Gerät. Standardmäßig abonniert diese Komponente lokale Veröffentlichungs-/Abonnementnachrichten. Weitere Informationen zum Veröffentlichen von Nachrichten in dieser Komponente aus Ihren benutzerdefinierten Komponenten finden Sie unter Lokale Nachrichten veröffentlichen/abonnieren.

Standardthema (lokales Veröffentlichen/Abonnieren): `modbus/adapter/request`

Die Nachricht akzeptiert die folgenden Eigenschaften. Eingabenachrichten müssen im JSON-Format vorliegen.

`request`

Die Parameter für die zu sendende Bol RTU-Anforderung.

Die Form der Anforderungsnachricht hängt von der Art der Bol-RTU-Anforderung ab, die sie darstellt. Die folgenden Eigenschaften sind für alle Anforderungen erforderlich.

Typ: `object`, der die folgenden Informationen enthält:

`operation`

Der Name der auszuführenden Operation. Geben Sie zum Beispiel an, `ReadCoilsRequest` um Telefonie auf einem microSD-RTU-Gerät zu lesen. Weitere Informationen zu unterstützten Operationen finden Sie unter Modbus RTU-Anforderungen und -Antworten.

Typ: `string`

`device`

Das Zielgerät der Anfrage.

Dieser Wert muss eine Ganzzahl zwischen `0` und sein`247`.

Typ: `integer`

Die weiteren Parameter, die in die Anforderung aufgenommen werden sollen, hängen von der Operation ab. Diese Komponente übernimmt die zyklische Redundanzprüfung (CRC), um Datenanforderungen für Sie zu überprüfen.

> **Note**
>
> Wenn Ihre Anforderung eine -`address`Eigenschaft enthält, müssen Sie ihren Wert als Ganzzahl angeben. Beispiel: `"address": 1`

`id`

Eine willkürliche ID für die Anforderung. Verwenden Sie diese Eigenschaft, um eine Eingabeanforderung einer Ausgabeantwort zuzuordnen. Wenn Sie diese Eigenschaft angeben, legt die Komponente die `id` Eigenschaft im Antwortobjekt auf diesen Wert fest.

Typ: `string`

Example Beispieleingabe: Lesen Spulenanforderungen

```
{
  "request": {
    "operation": "ReadCoilsRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "MyRequest"
}
```

Ausgabedaten

Diese Komponente veröffentlicht Antworten standardmäßig als Ausgabedaten zum folgenden MQTT-Thema. Sie müssen dieses Thema als `subject` in der Konfiguration für die Legacy-Abonnement-Routerkomponente angeben. Weitere Informationen zum Abonnieren von Nachrichten zu diesem Thema in Ihren benutzerdefinierten Komponenten finden Sie unter MQTT-Nachrichten veröffentlichen/abonnieren AWS IoT Core.

Standardthema (AWS IoT Core MQTT): `modbus/adapter/response`

Die Form der Antwortnachricht hängt von der Anforderungsoperation und dem Antwortstatus ab. Beispiele finden Sie unter Beispiel: Anforderungen und Antworten.

Jede Antwort beinhaltet die folgenden Eigenschaften:

`response`

Die Antwort des Bol RTU-Geräts.

Typ: `object`, der die folgenden Informationen enthält:

`status`

Der Status der Anforderung. Der Status kann einer der folgenden Werte sein:

- `Success` – Die Anforderung war gültig, die Komponente hat die Anforderung an das Bol-RTU-Netzwerk gesendet und das-RTU-Netzwerk hat eine Antwort zurückgegeben.
- `Exception` – Die Anforderung war gültig, die Komponente hat die Anforderung an das Bol-RTU-Netzwerk gesendet und das Bol-RTU-Netzwerk hat eine Ausnahme zurückgegeben. Weitere Informationen finden Sie unter Antwortstatus: Ausnahme.
- `No Response` – Die Anforderung war ungültig, und die Komponente hat den Fehler erkannt, bevor sie die Anforderung an das Bol-RTU-Netzwerk gesendet hat. Weitere Informationen finden Sie unter Antwortstatus: Keine Antwort.

`operation`

Der Vorgang, den die Komponente angefordert hat.

`device`

Das Gerät, an das die Komponente die Anforderung gesendet hat.

`payload`

Die Antwort des Bol RTU-Geräts. Wenn `status` ist`No Response`, enthält dieses Objekt nur eine -`error`Eigenschaft mit der Beschreibung des Fehlers (z. B. `[Input/Output] No Response received from the remote unit`).

`id`

Die ID der Anforderung, mit der Sie ermitteln können, welche Antwort welcher Anforderung entspricht.

> **Note**
>
> Eine Antwort für einen Schreibvorgang ist lediglich ein Echo der Anforderung. Obwohl Schreibantworten keine aussagekräftigen Informationen enthalten, empfiehlt es sich, den

Status der Antwort zu überprüfen, um festzustellen, ob die Anforderung erfolgreich ist oder fehlschlägt.

Example Beispielausgabe: Erfolg

```
{
  "response" : {
    "status" : "success",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "function_code": 1,
      "bits": [1]
    }
  },
  "id" : "MyRequest"
}
```

Example Beispielausgabe: Fehler

```
{
  "response" : {
    "status" : "fail",
    "error_message": "Internal Error",
    "error": "Exception",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "function_code": 129,
      "exception_code": 2
    }
  },
  "id" : "MyRequest"
}
```

Weitere Beispiele finden Sie unter 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
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.

Lesen Sie sich die Abschnitte durch

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"
}
```

Lesen diskreter Eingaben

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 Warteregistern

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"
}
```

Leseeingabe-Register

Anfragebeispiel:

```
{
  "request": {
    "operation": "ReadInputRegistersRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
```

```
  "id": "TestRequest"
}
```

Schreiben einer einzelnen Kabel

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 einzelnen Registers

Anfragebeispiel:

```
{
  "request": {
    "operation": "WriteSingleRegisterRequest",
    "device": 1,
    "address": 1,
```

```
    "value": 1
  },
  "id": "TestRequest"
}
```

Schreiben mehrerer Kabel

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"
}
```

Schreibregister maskieren

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 von Schreibvorgängen für mehrere 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 Antwort enthält die Registrierungen, die die Komponente liest.

Antwortstatus: Ausnahme

Ausnahmen können auftreten, wenn das Anfrageformat gültig ist, die Anfrage aber nicht erfolgreich abgeschlossen wurde. In diesem Fall enthält die Antwort die folgenden Informationen:

- Der `status` wird auf `Exception` gesetzt.
- Der `function_code` entspricht dem Funktionscode der Anforderung + 128.
- Der `exception_code` enthält den Ausnahmecode. Weitere Informationen finden Sie unter Modbus-Ausnahmecodes.

Beispiel:

```
{
  "response": {
    "status": "fail",
    "error_message": "Internal Error",
    "error": "Exception",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "function_code": 129,
      "exception_code": 2
    }
  },
  "id": "TestRequest"
}
```

Antwortstatus: Keine Antwort

Dieser Konnektor führt Validierungsprüfungen für die Modbus-Anforderung durch. So wird beispielsweise nach ungültigen Formaten und fehlenden Feldern gesucht. Wenn die Validierung fehlschlägt, sendet der Konnektor die Anforderung nicht. Stattdessen gibt er eine Antwort zurück, die die folgenden Informationen enthält:

- Der `status` wird auf `No Response` gesetzt.

- Der `error` enthält die Fehlerursache.
- 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 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"
}
```

Lokale Protokolldatei

Diese Komponente verwendet die folgende Protokolldatei.

```
/greengrass/v2/logs/aws.greengrass.Modbus.log
```

So zeigen Sie die Protokolle dieser Komponente an

- Führen Sie den folgenden Befehl auf dem Core-Gerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie durch */greengrass/v2* den Pfad zum AWS IoT Greengrass Stammordner.

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

Lizenzen

Diese Komponente umfasst die folgende Software/Lizenzierung von Drittanbietern:

- Pymodbus/BSD-Lizenz
- Pyseriale /BSD-Lizenz

Diese Komponente wird gemäß dem Greengrass Core Software License Agreement veröffentlicht.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in jeder Version der Komponente beschrieben.

Version	Änderungen
2.1.8	Version für Greengrass-Kern Version 2.12.0 aktualisiert.
2.1.7	Version für Greengrass-Kern Version 2.11.0 aktualisiert.
2.1.6	Version für Greengrass-Kernversion 2.10.0 aktualisiert.
2.1.5	Fehlerbehebungen und Verbesserungen • Behebt ein Problem mit dem -`ReadDiscreteInput` Vorgang.
2.1.4	Version für Greengrass-Kern Version 2.9.0 aktualisiert.
2.1.3	Version für Greengrass-Kern Version 2.8.0 aktualisiert.

Version	Änderungen
2.1.2	Version für Greengrass-Kern Version 2.7.0 aktualisiert.
2.1.1	Version für Greengrass-Kern Version 2.6.0 aktualisiert.
2.1.0	Neue Features • Fügt die `ModbusStopBits` Optionen `ModbusBaudRate`, `ModbusByteSize`, und hinzu, die Sie angeben können`ModbusParity`, um die serielle Kommunikation mit Bol RTU-Geräten zu konfigurieren.
2.0.8	Version für Greengrass-Kern Version 2.5.0 aktualisiert.
2.0.7	Version für Greengrass-Kern Version 2.4.0 aktualisiert.
2.0.6	Version für Greengrass-Kern Version 2.3.0 aktualisiert.
2.0.5	Version für Greengrass-Kern Version 2.2.0 aktualisiert.
2.0.4	Version für Greengrass-Kern Version 2.1.0 aktualisiert.
2.0.3	Erste Version

MQTT-Brücke

Die MQTT-Bridge-Komponente (`aws.greengrass.clientdevices.mqtt.Bridge`) leitet MQTT-Nachrichten zwischen Client-Geräten, lokalem Greengrass Publish/Subscribe und weiter. AWS IoT Core Sie können diese Komponente verwenden, um auf MQTT-Nachrichten von Client-Geräten in benutzerdefinierten Komponenten zu reagieren und Client-Geräte mit dem zu synchronisieren. AWS Cloud

> **Note**
>
> Client-Geräte sind lokale IoT-Geräte, die eine Verbindung zu einem Greengrass-Core-Gerät herstellen, um MQTT-Nachrichten und Daten zur Verarbeitung zu senden. Weitere Informationen finden Sie unter Interagieren mit lokalen IoT-Geräten.

Sie können diese Komponente verwenden, um Nachrichten zwischen den folgenden Message Brokern weiterzuleiten:

- Lokaler MQTT — Der lokale MQTT-Broker verarbeitet Nachrichten zwischen Client-Geräten und einem Core-Gerät.
- Lokales Veröffentlichen/Abonnieren — Der lokale Greengrass-Nachrichtenbroker verarbeitet Nachrichten zwischen Komponenten auf einem Kerngerät. Weitere Informationen zur Interaktion mit diesen Nachrichten in Greengrass-Komponenten finden Sie unterLokale Nachrichten veröffentlichen/abonnieren.
- AWS IoT Core — Der AWS IoT Core MQTT-Broker verarbeitet Nachrichten zwischen IoT-Geräten und AWS Cloud Zielen. Weitere Informationen zur Interaktion mit diesen Nachrichten in Greengrass-Komponenten finden Sie unterMQTT-Nachrichten veröffentlichen/abonnieren AWS IoT Core.

> **Note**
>
> Die MQTT-Bridge verwendet QoS 1 zum Veröffentlichen und Abonnieren AWS IoT Core, auch wenn ein Client-Gerät QoS 0 verwendet, um den lokalen MQTT-Broker zu veröffentlichen und zu abonnieren. Infolgedessen können Sie zusätzliche Latenz beobachten, wenn Sie MQTT-Nachrichten von Client-Geräten auf dem lokalen MQTT-Broker an weiterleiten. AWS IoT Core Weitere Informationen zur MQTT-Konfiguration auf Kerngeräten finden Sie unter. Konfigurieren Sie MQTT-Timeouts und Cache-Einstellungen

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Typ

Diese Komponente ist eine Plugin-Komponente (`aws.greengrass.plugin`). Der Greengrass-Kern führt diese Komponente in derselben Java Virtual Machine (JVM) wie der Nucleus aus. Der Nucleus wird neu gestartet, wenn Sie die Version dieser Komponente auf dem Core-Gerät ändern.

Diese Komponente verwendet dieselbe Protokolldatei wie der Greengrass-Kern. Weitere Informationen finden Sie unter Überwachen von AWS IoT Greengrass Protokollen.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf Kerngeräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Wenn Sie die MQTT-Broker-Komponente des Kerngeräts so konfigurieren, dass sie einen anderen Port als den Standardport 8883 verwendet, müssen Sie MQTT Bridge v2.1.0 oder höher verwenden. Konfigurieren Sie es so, dass es eine Verbindung über den Port herstellt, an dem der Broker arbeitet.
- Die MQTT-Bridge-Komponente wird für die Ausführung in einer VPC unterstützt.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt sie AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und all ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitstellen zu können. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente sowie die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste der Abhängigkeiten.

2.3.2

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.3.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Authentifizierung auf dem Client-Gerät	>=2.2.0 <2.6.0	Hart

2.3.0 and 2.3.1

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.3.0 und 2.3.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Authentifizierung auf dem Client-Gerät	>=2.2.0 <2.5.0	Hart

2.2.5 and 2.2.6

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.2.5 und 2.2.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Authentifizierung auf dem Client-Gerät	>=2.2.0 <2.5.0	Hart

2.2.3 and 2.2.4

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.2.3 und 2.2.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Authentifizierung auf dem Client-Gerät	>=2.2.0 <2.4.0	Hart

2.2.0 – 2.2.2

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.2.0 bis 2.2.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Authentifizierung auf dem Client-Gerät	>=2.2.0 <2.3.0	Hart

2.1.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Authentifizierung auf dem Client-Gerät	>=2.0.0 <2.2.0	Hart

2.0.0 to 2.1.0

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.0.0 bis 2.1.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Authentifizierung auf dem Client-Gerät	>=2.0.0 <2.1.0	Hart

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zu den Komponentenrezepten.

Konfiguration

Diese Komponente stellt die folgenden Konfigurationsparameter bereit, die Sie bei der Bereitstellung der Komponente anpassen können.

2.3.0 – 2.3.2

`mqttTopicMapping`

Die Themenzuordnungen, die Sie überbrücken möchten. Diese Komponente abonniert Nachrichten zum Quellthema und veröffentlicht die Nachrichten, die sie empfängt, im Zielthema. Jede Themenzuordnung definiert das Thema, den Quelltyp und den Zieltyp.

Dieses Objekt enthält die folgenden Informationen:

`topicMappingNameKey`

Der Name dieser Themenzuordnung. Ersetzen Sie *`topicMappingNameKey`* durch einen Namen, anhand dessen Sie diese Themenzuordnung leichter identifizieren können.

Dieses Objekt enthält die folgenden Informationen:

`topic`

Das Thema oder der Themenfilter, um eine Brücke zwischen dem Quell- und dem Zielbroker zu schlagen.

Sie können die Platzhalter + und # MQTT-Themen-Platzhalter verwenden, um Nachrichten zu allen Themen weiterzuleiten, die einem Themenfilter entsprechen.

Weitere Informationen finden Sie unter MQTT-Themen im Entwicklerhandbuch.AWS IoT Core

> **Note**
>
> Um MQTT-Themen-Platzhalter mit dem `Pubsub` Quellbroker zu verwenden, müssen Sie Version 2.6.0 oder höher der Greengrass Nucleus-Komponente verwenden.

`targetTopicPrefix`

Das Präfix, das dem Zielthema hinzugefügt werden soll, wenn diese Komponente die Nachricht weiterleitet.

`source`

Der Quellnachrichtenbroker. Wählen Sie aus den folgenden Optionen aus:

- `LocalMqtt`— Der lokale MQTT-Broker, über den Client-Geräte kommunizieren.
- `Pubsub`— Der lokale Greengrass-Nachrichtenbroker zum Veröffentlichen und Abonnieren.
- `IotCore`— Der AWS IoT Core MQTT-Nachrichtenbroker.

> **Note**
>
> Die MQTT-Bridge verwendet QoS 1 zum Veröffentlichen und Abonnieren AWS IoT Core, auch wenn ein Client-Gerät QoS 0 verwendet, um den lokalen MQTT-Broker zu veröffentlichen und zu abonnieren. Infolgedessen können Sie zusätzliche Latenz beobachten, wenn Sie MQTT-Nachrichten von Client-Geräten auf dem lokalen MQTT-Broker an weiterleiten. AWS IoT Core Weitere Informationen zur MQTT-Konfiguration auf Kerngeräten finden Sie unter. Konfigurieren Sie MQTT-Timeouts und Cache-Einstellungen

`source`und `target` muss anders sein.

`target`

Der Ziel-Nachrichtenbroker. Wählen Sie aus den folgenden Optionen aus:

- `LocalMqtt`— Der lokale MQTT-Broker, über den Client-Geräte kommunizieren.

- `Pubsub`— Der lokale Greengrass-Nachrichtenbroker zum Veröffentlichen und Abonnieren.
- `IotCore`— Der AWS IoT Core MQTT-Nachrichtenbroker.

> **Note**
>
> Die MQTT-Bridge verwendet QoS 1 zum Veröffentlichen und Abonnieren AWS IoT Core, auch wenn ein Client-Gerät QoS 0 verwendet, um den lokalen MQTT-Broker zu veröffentlichen und zu abonnieren. Infolgedessen können Sie zusätzliche Latenz beobachten, wenn Sie MQTT-Nachrichten von Client-Geräten auf dem lokalen MQTT-Broker an weiterleiten. AWS IoT Core Weitere Informationen zur MQTT-Konfiguration auf Kerngeräten finden Sie unter. Konfigurieren Sie MQTT-Timeouts und Cache-Einstellungen

`source`und `target` muss anders sein.

mqtt5 RouteOptions

(Optional) Stellt Optionen für die Konfiguration von Themenzuordnungen bereit, um Nachrichten vom Quellthema zum Zielthema zu überbrücken.

Dieses Objekt enthält die folgenden Informationen:

`mqtt5 RouteOptionsNameKey`

Der Name der Routenoptionen für ein Topic-Mapping. Ersetzen Sie *`mqtt5 RouteOptionsNameKey`* durch den passenden *`topicMappingNameSchlüssel`*, der `mqttTopicMapping` im Feld definiert ist.

Dieses Objekt enthält die folgenden Informationen:

Kein Lokal

(Optional) Wenn diese Option aktiviert ist, leitet die Bridge keine Nachrichten zu einem Thema weiter, das die Bridge selbst veröffentlicht hat. Verwenden Sie dies, um Schleifen wie folgt zu verhindern:

```
{
    "mqtt5RouteOptions": {
        "toIoTCore": {
            "noLocal": true
```

```
        }
    },
    "mqttTopicMapping": {
        "toIoTCore": {
            "topic": "device",
            "source": "LocalMqtt",
            "target": "IotCore"
        },
        "toLocal": {
            "topic": "device",
            "source": "IotCore",
            "target": "LocalMqtt"
        }
    }
}
```

noLocalwird nur für Routen unterstützt, auf denen der `source` ist`LocalMqtt`.

Standard: false

retainAsPublished

(Optional) Wenn diese Option aktiviert ist, haben Nachrichten, die von der Bridge weitergeleitet werden, dieselbe `retain` Kennzeichnung wie Nachrichten, die für diese Route auf dem Broker veröffentlicht wurden.

`retainAsPublished`wird nur für Routen unterstützt, bei denen dies der `source` ist`LocalMqtt`.

Standard: false

mqtt

(Optional) MQTT-Protokolleinstellungen für die Kommunikation mit dem lokalen Broker.

version

(Optional) Die MQTT-Protokollversion, die von der Bridge für die Kommunikation mit dem lokalen Broker verwendet wird. Muss mit der in der Nucleus-Konfiguration ausgewählten MQTT-Version identisch sein.

Wählen Sie eine der folgenden Optionen aus:

- `mqtt3`
- `mqtt5`

Sie müssen einen MQTT-Broker bereitstellen, wenn das `target` Feld `source` oder des `mqttTopicMapping` Objekts auf gesetzt ist. `LocalMqtt` Wenn Sie die `mqtt5` Option wählen, müssen Sie die MQTT 5-Broker (EMAX) verwenden.

Standard: `mqtt3`

ackTimeoutSeconds

(Optional) Zeitintervall für das Warten auf PUBACK-, SUBACK- oder UNSUBACK-Pakete, bevor der Vorgang fehlschlägt.

Standard: 60

connAckTimeoutFrau

(Optional) Zeitintervall, in dem auf ein CONNACK-Paket gewartet werden soll, bevor die Verbindung unterbrochen wird.

Standard: 20000 (20 Sekunden)

pingTimeoutMs

(Optional) Die Zeit in Millisekunden, die die Bridge auf den Empfang einer PINGACK-Nachricht vom lokalen Broker wartet. Wenn die Wartezeit das Timeout überschreitet, wird die Bridge geschlossen und die MQTT-Verbindung erneut geöffnet. Dieser Wert muss kleiner als sein. `keepAliveTimeoutSeconds`

Standard: 30000 (30 Sekunden)

keepAliveTimeoutSekunden

(Optional) Die Zeitspanne in Sekunden zwischen den einzelnen PING-Nachrichten, die die Bridge sendet, um die MQTT-Verbindung aufrechtzuerhalten. Dieser Wert muss größer als `pingTimeoutMs` sein.

Standard: 60

maxReconnectDelayFrau

(Optional) Die maximale Zeit in Sekunden, für die MQTT die Verbindung wiederherstellt.

Standard: 30000 (30 Sekunden)

minReconnectDelayFrau

(Optional) Die Mindestzeit in Sekunden, die MQTT benötigt, um die Verbindung wiederherzustellen.

Maximal empfangen

(Optional) Die maximale Anzahl unbestätigter QoS1-Pakete, die die Bridge senden kann.

Standard: 100

maximumPacketSize

Die maximale Anzahl von Byte, die der Client für ein MQTT-Paket akzeptiert.

Standard: null (kein Limit)

sessionExpiryInterval

(Optional) Die Dauer in Sekunden, die Sie für die Dauer einer Sitzung zwischen der Bridge und dem lokalen Broker anfordern können.

Standard: 4294967295 (Sitzung läuft nie ab)

`brokerUri`

(Optional) Die URI des lokalen MQTT-Brokers. Sie müssen diesen Parameter angeben, wenn Sie den MQTT-Broker so konfigurieren, dass er einen anderen Port als den Standardport 8883 verwendet. Verwenden Sie das folgende Format und ersetzen Sie *`Port durch den Port`*, an dem der MQTT-Broker arbeitet:. `ssl://localhost:`*`port`*

Standard: `ssl://localhost:8883`

startupTimeoutSeconds

(Optional) Die maximale Zeit in Sekunden für den Start der Komponente. Der Status der Komponente ändert sich auf, `BROKEN` wenn dieser Timeout überschritten wird.

Standard: `120`

Example Beispiel: Aktualisierung der Zusammenführung von Konfigurationen

Das folgende Beispiel für ein Konfigurationsupdate spezifiziert Folgendes:

- Leitet Nachrichten von Client-Geräten AWS IoT Core an Themen weiter, die dem `clients/+/hello/world` Themenfilter entsprechen.
- Leitet Nachrichten von Client-Geräten zu Themen, die dem Themenfilter entsprechen, an das lokale Publish/Subscribe weiter und fügt dem `clients/+/detections` Zielthema das

`events/input/` Präfix hinzu. Das resultierende Zielthema entspricht dem `events/input/clients/+/detections` Themenfilter.

- Leiten Sie Nachrichten von Client-Geräten AWS IoT Core an Themen weiter, die dem `clients/+/status` Themenfilter entsprechen, und fügen Sie dem Zielthema das `$aws/rules/StatusUpdateRule/` Präfix hinzu. In diesem Beispiel werden diese Nachrichten direkt an eine AWS IoT Regel mit dem Namen „`StatusUpdateRule`Kosten reduzieren“ mithilfe von Basic Ingest weitergeleitet.

```
{
  "mqttTopicMapping": {
    "ClientDeviceHelloWorld": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDeviceEvents": {
      "topic": "clients/+/detections",
      "targetTopicPrefix": "events/input/",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ClientDeviceCloudStatusUpdate": {
      "topic": "clients/+/status",
      "targetTopicPrefix": "$aws/rules/StatusUpdateRule/",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

Example Beispiel: Konfiguration von MQTT 5

Die folgende Beispielkonfiguration aktualisiert Folgendes:

- Ermöglicht der Bridge, das MQTT 5-Protokoll mit dem lokalen Broker zu verwenden.
- Konfiguriert die Einstellung MQTT keep as published für das `ClientDeviceHelloWorld` Topic-Mapping.

```
{
```

```
  "mqttTopicMapping": {
    "ClientDeviceHelloWorld": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  },
  "mqtt5RouteOptions": {
    "ClientDeviceHelloWorld": {
      "retainAsPublished": true
    }
  },
  "mqtt": {
    "version": "mqtt5"
  }
}
```

2.2.6

`mqttTopicMapping`

Die Themenzuordnungen, die Sie überbrücken möchten. Diese Komponente abonniert Nachrichten zum Quellthema und veröffentlicht die Nachrichten, die sie empfängt, im Zielthema. Jede Themenzuordnung definiert das Thema, den Quelltyp und den Zieltyp.

Dieses Objekt enthält die folgenden Informationen:

`topicMappingNameKey`

Der Name dieser Themenzuordnung. Ersetzen Sie *`topicMappingNameKey`* durch einen Namen, anhand dessen Sie diese Themenzuordnung leichter identifizieren können.

Dieses Objekt enthält die folgenden Informationen:

`topic`

Das Thema oder der Themenfilter, um eine Brücke zwischen dem Quell- und dem Zielbroker zu schlagen.

Sie können die Platzhalter + und # MQTT-Themen-Platzhalter verwenden, um Nachrichten zu allen Themen weiterzuleiten, die einem Themenfilter entsprechen. Weitere Informationen finden Sie unter MQTT-Themen im Entwicklerhandbuch.AWS IoT Core

> **Note**
>
> Um MQTT-Themen-Platzhalter mit dem `Pubsub` Quellbroker zu verwenden, müssen Sie Version 2.6.0 oder höher der Greengrass Nucleus-Komponente verwenden.

`targetTopicPrefix`

Das Präfix, das dem Zielthema hinzugefügt werden soll, wenn diese Komponente die Nachricht weiterleitet.

`source`

Der Quellnachrichtenbroker. Wählen Sie aus den folgenden Optionen aus:

- `LocalMqtt`— Der lokale MQTT-Broker, über den Client-Geräte kommunizieren.
- `Pubsub`— Der lokale Greengrass-Nachrichtenbroker zum Veröffentlichen und Abonnieren.
- `IotCore`— Der AWS IoT Core MQTT-Nachrichtenbroker.

> **Note**
>
> Die MQTT-Bridge verwendet QoS 1 zum Veröffentlichen und Abonnieren AWS IoT Core, auch wenn ein Client-Gerät QoS 0 verwendet, um den lokalen MQTT-Broker zu veröffentlichen und zu abonnieren. Infolgedessen können Sie zusätzliche Latenz beobachten, wenn Sie MQTT-Nachrichten von Client-Geräten auf dem lokalen MQTT-Broker an weiterleiten. AWS IoT Core Weitere Informationen zur MQTT-Konfiguration auf Kerngeräten finden Sie unter. Konfigurieren Sie MQTT-Timeouts und Cache-Einstellungen

`source`und `target` muss anders sein.

`target`

Der Ziel-Nachrichtenbroker. Wählen Sie aus den folgenden Optionen aus:

- `LocalMqtt`— Der lokale MQTT-Broker, über den Client-Geräte kommunizieren.
- `Pubsub`— Der lokale Greengrass-Nachrichtenbroker zum Veröffentlichen und Abonnieren.

- `IotCore`— Der AWS IoT Core MQTT-Nachrichtenbroker.

> **Note**
>
> Die MQTT-Bridge verwendet QoS 1 zum Veröffentlichen und Abonnieren AWS IoT Core, auch wenn ein Client-Gerät QoS 0 verwendet, um den lokalen MQTT-Broker zu veröffentlichen und zu abonnieren. Infolgedessen können Sie zusätzliche Latenz beobachten, wenn Sie MQTT-Nachrichten von Client-Geräten auf dem lokalen MQTT-Broker an weiterleiten. AWS IoT Core Weitere Informationen zur MQTT-Konfiguration auf Kerngeräten finden Sie unter. Konfigurieren Sie MQTT-Timeouts und Cache-Einstellungen

`source`und `target` muss anders sein.

`brokerUri`

(Optional) Die URI des lokalen MQTT-Brokers. Sie müssen diesen Parameter angeben, wenn Sie den MQTT-Broker so konfigurieren, dass er einen anderen Port als den Standardport 8883 verwendet. Verwenden Sie das folgende Format und ersetzen Sie *`Port durch den Port`*, an dem der MQTT-Broker arbeitet:. `ssl://localhost:`*`port`*

Standard: `ssl://localhost:8883`

startupTimeoutSeconds

(Optional) Die maximale Zeit in Sekunden für den Start der Komponente. Der Status der Komponente ändert sich auf, `BROKEN` wenn dieser Timeout überschritten wird.

Standard: `120`

Example Beispiel: Aktualisierung der Zusammenführung von Konfigurationen

Das folgende Beispiel für ein Konfigurationsupdate spezifiziert Folgendes:

- Leitet Nachrichten von Client-Geräten AWS IoT Core an Themen weiter, die dem `clients/+/hello/world` Themenfilter entsprechen.
- Leitet Nachrichten von Client-Geräten zu Themen, die dem Themenfilter entsprechen, an das lokale Publish/Subscribe weiter und fügt dem `clients/+/detections` Zielthema das `events/input/` Präfix hinzu. Das resultierende Zielthema entspricht dem `events/input/clients/+/detections` Themenfilter.

- Leiten Sie Nachrichten von Client-Geräten AWS IoT Core an Themen weiter, die dem `clients/+/status` Themenfilter entsprechen, und fügen Sie dem Zielthema das `$aws/rules/StatusUpdateRule/` Präfix hinzu. In diesem Beispiel werden diese Nachrichten direkt an eine AWS IoT Regel mit dem Namen „`StatusUpdateRule`Kosten reduzieren“ mithilfe von Basic Ingest weitergeleitet.

```
{
  "mqttTopicMapping": {
    "ClientDeviceHelloWorld": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDeviceEvents": {
      "topic": "clients/+/detections",
      "targetTopicPrefix": "events/input/",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ClientDeviceCloudStatusUpdate": {
      "topic": "clients/+/status",
      "targetTopicPrefix": "$aws/rules/StatusUpdateRule/",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

2.2.0 - 2.2.5

`mqttTopicMapping`

Die Themenzuordnungen, die Sie überbrücken möchten. Diese Komponente abonniert Nachrichten zum Quellthema und veröffentlicht die Nachrichten, die sie empfängt, im Zielthema. Jede Themenzuordnung definiert das Thema, den Quelltyp und den Zieltyp.

Dieses Objekt enthält die folgenden Informationen:

`topicMappingNameKey`

Der Name dieser Themenzuordnung. Ersetzen Sie *`topicMappingNameKey`* durch einen Namen, anhand dessen Sie diese Themenzuordnung leichter identifizieren können.

Dieses Objekt enthält die folgenden Informationen:

`topic`

Das Thema oder der Themenfilter, um eine Brücke zwischen dem Quell- und dem Zielbroker zu schlagen.

Sie können die Platzhalter + und # MQTT-Themen-Platzhalter verwenden, um Nachrichten zu allen Themen weiterzuleiten, die einem Themenfilter entsprechen. Weitere Informationen finden Sie unter MQTT-Themen im Entwicklerhandbuch.AWS IoT Core

> **Note**
>
> Um MQTT-Themen-Platzhalter mit dem `Pubsub` Quellbroker zu verwenden, müssen Sie Version 2.6.0 oder höher der Greengrass Nucleus-Komponente verwenden.

`targetTopicPrefix`

Das Präfix, das dem Zielthema hinzugefügt werden soll, wenn diese Komponente die Nachricht weiterleitet.

`source`

Der Quellnachrichtenbroker. Wählen Sie aus den folgenden Optionen aus:

- `LocalMqtt`— Der lokale MQTT-Broker, über den Client-Geräte kommunizieren.
- `Pubsub`— Der lokale Greengrass-Nachrichtenbroker zum Veröffentlichen und Abonnieren.
- `IotCore`— Der AWS IoT Core MQTT-Nachrichtenbroker.

> **Note**
>
> Die MQTT-Bridge verwendet QoS 1 zum Veröffentlichen und Abonnieren AWS IoT Core, auch wenn ein Client-Gerät QoS 0 verwendet, um den lokalen MQTT-Broker zu veröffentlichen und zu abonnieren. Infolgedessen können Sie zusätzliche Latenz beobachten, wenn Sie MQTT-Nachrichten von Client-Geräten auf dem lokalen MQTT-Broker an weiterleiten. AWS IoT Core

Weitere Informationen zur MQTT-Konfiguration auf Kerngeräten finden Sie unter. Konfigurieren Sie MQTT-Timeouts und Cache-Einstellungen

`source`und `target` muss anders sein.

`target`

Der Ziel-Nachrichtenbroker. Wählen Sie aus den folgenden Optionen aus:

- `LocalMqtt`— Der lokale MQTT-Broker, über den Client-Geräte kommunizieren.
- `Pubsub`— Der lokale Greengrass-Nachrichtenbroker zum Veröffentlichen und Abonnieren.
- `IotCore`— Der AWS IoT Core MQTT-Nachrichtenbroker.

> **Note**
>
> Die MQTT-Bridge verwendet QoS 1 zum Veröffentlichen und Abonnieren AWS IoT Core, auch wenn ein Client-Gerät QoS 0 verwendet, um den lokalen MQTT-Broker zu veröffentlichen und zu abonnieren. Infolgedessen können Sie zusätzliche Latenz beobachten, wenn Sie MQTT-Nachrichten von Client-Geräten auf dem lokalen MQTT-Broker an weiterleiten. AWS IoT Core Weitere Informationen zur MQTT-Konfiguration auf Kerngeräten finden Sie unter. Konfigurieren Sie MQTT-Timeouts und Cache-Einstellungen

`source`und `target` muss anders sein.

`brokerUri`

(Optional) Die URI des lokalen MQTT-Brokers. Sie müssen diesen Parameter angeben, wenn Sie den MQTT-Broker so konfigurieren, dass er einen anderen Port als den Standardport 8883 verwendet. Verwenden Sie das folgende Format und ersetzen Sie *`Port durch den Port`*, an dem der MQTT-Broker arbeitet:. `ssl://localhost:`*`port`*

Standard: `ssl://localhost:8883`

Example Beispiel: Aktualisierung der Konfigurationszusammenführung

Das folgende Beispiel für ein Konfigurationsupdate spezifiziert Folgendes:

- Leitet Nachrichten von Client-Geräten AWS IoT Core an Themen weiter, die dem `clients/+/hello/world` Themenfilter entsprechen.
- Leitet Nachrichten von Client-Geräten zu Themen, die dem Themenfilter entsprechen, an das lokale Publish/Subscribe weiter und fügt dem `clients/+/detections` Zielthema das `events/input/` Präfix hinzu. Das resultierende Zielthema entspricht dem `events/input/clients/+/detections` Themenfilter.
- Leiten Sie Nachrichten von Client-Geräten AWS IoT Core an Themen weiter, die dem `clients/+/status` Themenfilter entsprechen, und fügen Sie dem Zielthema das `$aws/rules/StatusUpdateRule/` Präfix hinzu. In diesem Beispiel werden diese Nachrichten direkt an eine AWS IoT Regel mit dem Namen „`StatusUpdateRule`Kosten reduzieren“ mithilfe von Basic Ingest weitergeleitet.

```
{
  "mqttTopicMapping": {
    "ClientDeviceHelloWorld": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDeviceEvents": {
      "topic": "clients/+/detections",
      "targetTopicPrefix": "events/input/",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ClientDeviceCloudStatusUpdate": {
      "topic": "clients/+/status",
      "targetTopicPrefix": "$aws/rules/StatusUpdateRule/",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

2.1.x

`mqttTopicMapping`

Die Themenzuordnungen, die Sie überbrücken möchten. Diese Komponente abonniert Nachrichten zum Quellthema und veröffentlicht die Nachrichten, die sie empfängt, im Zielthema. Jede Themenzuordnung definiert das Thema, den Quelltyp und den Zieltyp.

Dieses Objekt enthält die folgenden Informationen:

`topicMappingNameKey`

Der Name dieser Themenzuordnung. Ersetzen Sie *`topicMappingNameKey`* durch einen Namen, anhand dessen Sie diese Themenzuordnung leichter identifizieren können.

Dieses Objekt enthält die folgenden Informationen:

`topic`

Das Thema oder der Themenfilter, um eine Brücke zwischen dem Quell- und dem Zielbroker zu schlagen.

Wenn Sie den `LocalMqtt` oder den `IotCore` Quellbroker angeben, können Sie die Topic-Platzhalter + und den # MQTT-Themen-Platzhalter verwenden, um Nachrichten zu allen Themen weiterzuleiten, die einem Themenfilter entsprechen. Weitere Informationen finden Sie unter MQTT-Themen im Entwicklerhandbuch.AWS IoT Core

`source`

Der Quell-Message-Broker. Wählen Sie aus den folgenden Optionen aus:

- `LocalMqtt`— Der lokale MQTT-Broker, über den Client-Geräte kommunizieren.
- `Pubsub`— Der lokale Greengrass-Nachrichtenbroker zum Veröffentlichen und Abonnieren.
- `IotCore`— Der AWS IoT Core MQTT-Nachrichtenbroker.

> **Note**
>
> Die MQTT-Bridge verwendet QoS 1 zum Veröffentlichen und Abonnieren AWS IoT Core, auch wenn ein Client-Gerät QoS 0 verwendet, um den lokalen MQTT-Broker zu veröffentlichen und zu abonnieren. Infolgedessen können Sie zusätzliche Latenz beobachten, wenn Sie MQTT-Nachrichten von Client-Geräten auf dem lokalen MQTT-Broker an weiterleiten. AWS IoT Core

Weitere Informationen zur MQTT-Konfiguration auf Kerngeräten finden Sie unter. Konfigurieren Sie MQTT-Timeouts und Cache-Einstellungen

`source`und `target` muss anders sein.

`target`

Der Ziel-Nachrichtenbroker. Wählen Sie aus den folgenden Optionen aus:

- `LocalMqtt`— Der lokale MQTT-Broker, über den Client-Geräte kommunizieren.
- `Pubsub`— Der lokale Greengrass-Nachrichtenbroker zum Veröffentlichen und Abonnieren.
- `IotCore`— Der AWS IoT Core MQTT-Nachrichtenbroker.

> **Note**
>
> Die MQTT-Bridge verwendet QoS 1 zum Veröffentlichen und Abonnieren AWS IoT Core, auch wenn ein Client-Gerät QoS 0 verwendet, um den lokalen MQTT-Broker zu veröffentlichen und zu abonnieren. Infolgedessen können Sie zusätzliche Latenz beobachten, wenn Sie MQTT-Nachrichten von Client-Geräten auf dem lokalen MQTT-Broker an weiterleiten. AWS IoT Core Weitere Informationen zur MQTT-Konfiguration auf Kerngeräten finden Sie unter. Konfigurieren Sie MQTT-Timeouts und Cache-Einstellungen

`source`und `target` muss anders sein.

`brokerUri`

(Optional) Die URI des lokalen MQTT-Brokers. Sie müssen diesen Parameter angeben, wenn Sie den MQTT-Broker so konfigurieren, dass er einen anderen Port als den Standardport 8883 verwendet. Verwenden Sie das folgende Format und ersetzen Sie *`Port durch den Port`*, an dem der MQTT-Broker arbeitet:. `ssl://localhost:`*`port`*

Standard: `ssl://localhost:8883`

Example Beispiel: Aktualisierung der Konfigurationszusammenführung

Das folgende Beispiel für ein Konfigurationsupdate spezifiziert die Weiterleitung von Nachrichten von Client-Geräten AWS IoT Core an die `clients/MyClientDevice2/hello/world` Themen `clients/MyClientDevice1/hello/world` und.

```
{
  "mqttTopicMapping": {
    "ClientDevice1HelloWorld": {
      "topic": "clients/MyClientDevice1/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDevice2HelloWorld": {
      "topic": "clients/MyClientDevice2/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

2.0.x

`mqttTopicMapping`

Die Themenzuordnungen, die Sie verbinden möchten. Diese Komponente abonniert Nachrichten zum Quellthema und veröffentlicht die Nachrichten, die sie empfängt, im Zielthema. Jede Themenzuordnung definiert das Thema, den Quelltyp und den Zieltyp.

Dieses Objekt enthält die folgenden Informationen:

`topicMappingNameKey`

Der Name dieser Themenzuordnung. Ersetzen Sie *`topicMappingNameKey`* durch einen Namen, anhand dessen Sie diese Themenzuordnung leichter identifizieren können.

Dieses Objekt enthält die folgenden Informationen:

`topic`

Das Thema oder der Themenfilter, um eine Brücke zwischen dem Quell- und dem Zielbroker zu schlagen.

Wenn Sie den `LocalMqtt` oder den `IotCore` Quellbroker angeben, können Sie die Topic-Platzhalter + und den # MQTT-Themen-Platzhalter verwenden, um Nachrichten zu allen Themen weiterzuleiten, die einem Themenfilter entsprechen. Weitere Informationen finden Sie unter MQTT-Themen im Entwicklerhandbuch.AWS IoT Core

`source`

Der Quell-Message-Broker. Wählen Sie aus den folgenden Optionen aus:

- `LocalMqtt`— Der lokale MQTT-Broker, über den Client-Geräte kommunizieren.
- `Pubsub`— Der lokale Greengrass-Nachrichtenbroker zum Veröffentlichen und Abonnieren.
- `IotCore`— Der AWS IoT Core MQTT-Nachrichtenbroker.

> **Note**
>
> Die MQTT-Bridge verwendet QoS 1 zum Veröffentlichen und Abonnieren AWS IoT Core, auch wenn ein Client-Gerät QoS 0 verwendet, um den lokalen MQTT-Broker zu veröffentlichen und zu abonnieren. Infolgedessen können Sie zusätzliche Latenz beobachten, wenn Sie MQTT-Nachrichten von Client-Geräten auf dem lokalen MQTT-Broker an weiterleiten. AWS IoT Core Weitere Informationen zur MQTT-Konfiguration auf Kerngeräten finden Sie unter. Konfigurieren Sie MQTT-Timeouts und Cache-Einstellungen

`source`und `target` muss anders sein.

`target`

Der Ziel-Nachrichtenbroker. Wählen Sie aus den folgenden Optionen aus:

- `LocalMqtt`— Der lokale MQTT-Broker, über den Client-Geräte kommunizieren.
- `Pubsub`— Der lokale Greengrass-Nachrichtenbroker zum Veröffentlichen und Abonnieren.
- `IotCore`— Der AWS IoT Core MQTT-Nachrichtenbroker.

> **Note**
>
> Die MQTT-Bridge verwendet QoS 1 zum Veröffentlichen und Abonnieren AWS IoT Core, auch wenn ein Client-Gerät QoS 0 verwendet, um den lokalen MQTT-Broker zu veröffentlichen und zu abonnieren. Infolgedessen können Sie zusätzliche Latenz beobachten, wenn Sie MQTT-Nachrichten von Client-Geräten auf dem lokalen MQTT-Broker an weiterleiten. AWS IoT Core Weitere Informationen zur MQTT-Konfiguration auf Kerngeräten finden Sie unter. Konfigurieren Sie MQTT-Timeouts und Cache-Einstellungen

`source`und `target` muss anders sein.

Example Beispiel: Aktualisierung der Zusammenführung von Konfigurationen

Das folgende Beispiel für ein Konfigurationsupdate spezifiziert die Weiterleitung von Nachrichten von Client-Geräten AWS IoT Core an die `clients/MyClientDevice2/hello/world` Themen `clients/MyClientDevice1/hello/world` und.

```
{
  "mqttTopicMapping": {
    "ClientDevice1HelloWorld": {
      "topic": "clients/MyClientDevice1/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDevice2HelloWorld": {
      "topic": "clients/MyClientDevice2/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

Lokale Protokolldatei

Diese Komponente verwendet dieselbe Protokolldatei wie die Greengrass Nucleus-Komponente.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

Um die Protokolle dieser Komponente einzusehen

- Führen Sie den folgenden Befehl auf dem Kerngerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

  Linux

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

  Windows (PowerShell)

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

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
2.3.2	Die Version wurde für die Version 2.5.0 der Client-Geräteauthentifizierung aktualisiert.
2.3.1	Fehlerkorrekturen und Verbesserungen Behebt ein Problem, bei dem der lokale MQTT-Client in eine Trennschleife gerät.
2.3.0	Neue Features Fügt MQTT5-Unterstützung für das Bridging zwischen AWS IoT Core und lokalen MQTT-Quellen hinzu.
2.2.6	Neue Features Fügt eine neue Konfigurationsoption hinzu. `startupTimeoutSeconds`

Version	Änderungen
2.2.5	Die Version wurde für die Version 2.4.0 der Client-Geräteauthentifizierung aktualisiert.
2.2.4	Die Version wurde für die Version 2.3.0 der Greengrass-Client-Geräteauthentifizierung aktualisiert.
2.2.3	Diese Version enthält Fehlerkorrekturen und Verbesserungen.
2.2.2	Fehlerkorrekturen und Verbesserungen • Anpassungen protokollieren.
2.2.1	Fehlerkorrekturen und Verbesserungen Behebt Probleme, die dazu führen können, dass die MQTT-Bridge MQTT-Themen nicht abonniert.
2.2.0	Neue Features • Integriert die Unterstützung für Platzhalter (#und+) für MQTT-Themen, wenn Sie local publish/subscribe als Quellnachrichtenbroker angeben. Für diese Funktion ist Version 2.6.0 oder höher der Greengrass Nucleus-Komponente erforderlich. • Fügt die `targetTopicPrefix` Option hinzu, die Sie angeben können, um die MQTT-Bridge so zu konfigurieren, dass dem Zielthema bei der Weiterleitung einer Nachricht ein Präfix hinzugefügt wird.
2.1.1	Fehlerkorrekturen und Verbesserungen • Behebt Probleme mit der Art und Weise, wie diese Komponente Updates zum Zurücksetzen der Konfiguration verarbeitet. • Reduziert die Häufigkeit von Verbindungsabbrüchen beim MQTT-Client, wenn Zertifikate rotieren.
2.1.0	Neue Features • Fügt den `brokerUri` Parameter hinzu, der es Ihnen ermöglicht, einen nicht standardmäßigen MQTT-Broker-Port zu verwenden.

Version	Änderungen
2.0.1	Diese Version enthält Fehlerkorrekturen und Verbesserungen.
2.0.0	Erste Version

MQTT 3.1.1-Broker (Moquette)

Die Moquette MQTT-Broker-Komponente (`aws.greengrass.clientdevices.mqtt.Moquette`) verarbeitet MQTT-Nachrichten zwischen Client-Geräten und einem Greengrass-Core-Gerät. Diese Komponente stellt eine modifizierte Version des Moquette MQTT-Brokers bereit. Stellen Sie diesen MQTT-Broker bereit, um einen einfachen MQTT-Broker auszuführen. Weitere Informationen zur Auswahl eines MQTT-Brokers finden Sie unter. Auswählen eines MQTT-Brokers

Dieser Broker implementiert das MQTT 3.1.1-Protokoll. Es umfasst Unterstützung für gespeicherte QoS 0-, QoS 1- und QoS-2-Nachrichten, Lastwill-Nachrichten und persistente Sitzungen.

> **Note**
>
> Client-Geräte sind lokale IoT-Geräte, die eine Verbindung zu einem Greengrass-Core-Gerät herstellen, um MQTT-Nachrichten und Daten zur Verarbeitung zu senden. Weitere Informationen finden Sie unter Interagieren mit lokalen IoT-Geräten.

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Typ

Diese Komponente ist eine Plugin-Komponente (`aws.greengrass.plugin`). Der Greengrass-Kern führt diese Komponente in derselben Java Virtual Machine (JVM) wie der Nucleus aus. Der Nucleus wird neu gestartet, wenn Sie die Version dieser Komponente auf dem Core-Gerät ändern.

Diese Komponente verwendet dieselbe Protokolldatei wie der Greengrass-Kern. Weitere Informationen finden Sie unter Überwachen von AWS IoT Greengrass Protokollen.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf Kerngeräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Das Kerngerät muss in der Lage sein, Verbindungen an dem Port anzunehmen, an dem der MQTT-Broker arbeitet. Diese Komponente führt den MQTT-Broker standardmäßig auf Port 8883 aus. Sie können bei der Konfiguration dieser Komponente einen anderen Port angeben.

  Wenn Sie einen anderen Port angeben und die MQTT-Bridge-Komponente verwenden, um MQTT-Nachrichten an andere Broker weiterzuleiten, müssen Sie MQTT Bridge v2.1.0 oder höher

verwenden. Konfigurieren Sie es so, dass es den Port verwendet, an dem der MQTT-Broker arbeitet.

Wenn Sie einen anderen Port angeben und die IP-Detector-Komponente zur Verwaltung von MQTT-Broker-Endpunkten verwenden, müssen Sie den IP-Detektor v2.1.0 oder höher verwenden. Konfigurieren Sie es so, dass der Port gemeldet wird, an dem der MQTT-Broker arbeitet.

- Die Moquette MQTT-Broker-Komponente wird für die Ausführung in einer VPC unterstützt.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt sie AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und all ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitstellen zu können. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente sowie die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können die Abhängigkeiten für jede Version der Komponente auch in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste der Abhängigkeiten.

2.3.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.3.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Authentifizierung auf dem Client-Gerät	>=2.2.0 <2.6.0	Hart

2.3.2 – 2.3.6

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.3.2 bis 2.3.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Authentifizierung auf dem Client-Gerät	>=2.2.0 <2.5.0	Hart

2.3.0 and 2.3.1

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.3.0 und 2.3.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Authentifizierung auf dem Client-Gerät	>=2.2.0 <2.4.0	Hart

2.2.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.2.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Authentifizierung auf dem Client-Gerät	>=2.2.0 <2.3.0	Hart

2.1.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Authentifizierung auf dem Client-Gerät	>=2.0.0 <2.2.0	Hart

2.0.0 - 2.0.2

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.0.0 bis 2.0.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Authentifizierung auf dem Client-Gerät	>=2.0.0 <2.1.0	Hart

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zu den Komponentenrezepten.

Konfiguration

Diese Komponente stellt die folgenden Konfigurationsparameter bereit, die Sie bei der Bereitstellung der Komponente anpassen können.

`moquette`

(Optional) Die zu verwendende Moquette MQTT-Broker-Konfiguration. Sie können eine Teilmenge der Moqeutte-Konfigurationsoptionen in dieser Komponente konfigurieren. Weitere Informationen finden Sie in den Inline-Kommentaren in der Moquette-Konfigurationsdatei.

Dieses Objekt enthält die folgenden Informationen:

`ssl_port`

(Optional) Der Port, an dem der MQTT-Broker arbeitet.

> **Note**
>
> Wenn Sie einen anderen Port angeben und die MQTT-Bridge-Komponente verwenden, um MQTT-Nachrichten an andere Broker weiterzuleiten, müssen Sie MQTT Bridge v2.1.0 oder höher verwenden. Konfigurieren Sie es so, dass es den Port verwendet, an dem der MQTT-Broker arbeitet.
> Wenn Sie einen anderen Port angeben und die IP-Detector-Komponente zur Verwaltung von MQTT-Broker-Endpunkten verwenden, müssen Sie den IP-Detektor v2.1.0 oder höher verwenden. Konfigurieren Sie es so, dass der Port gemeldet wird, an dem der MQTT-Broker arbeitet.

Standard: `8883`

`host`

(Optional) Die Schnittstelle, an die der MQTT-Broker bindet. Sie können diesen Parameter beispielsweise so ändern, dass der MQTT-Broker nur an ein bestimmtes lokales Netzwerk bindet.

Standard: `0.0.0.0` (bindet an alle Netzwerkschnittstellen)

startupTimeoutSeconds

(Optional) Die maximale Zeit in Sekunden, die die Komponente zum Starten benötigt. Der Status der Komponente ändert sich auf, `BROKEN` wenn dieser Timeout überschritten wird.

Standard: `120`

Example Beispiel: Aktualisierung der Zusammenführung von Konfigurationen

Die folgende Beispielkonfiguration legt fest, dass der MQTT-Broker auf Port 443 betrieben werden soll.

```
{
  "moquette": {
    "ssl_port": "443"
  }
}
```

Lokale Protokolldatei

Diese Komponente verwendet dieselbe Protokolldatei wie die Greengrass Nucleus-Komponente.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

Um die Protokolle dieser Komponente einzusehen

- Führen Sie den folgenden Befehl auf dem Kerngerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

  Linux

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

Windows (PowerShell)

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

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
2.3.7	Die Version wurde für die Version 2.5.0 der Client-Geräteauthentifizierung aktualisiert.
2.3.6	Fehlerkorrekturen und Verbesserungen • Allgemeine Fehlerbehebungen und Verbesserungen.
2.3.5	Fehlerkorrekturen und Verbesserungen • Moquette wurde auf Version 0.17 aktualisiert.
2.3.4	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem es auf Clients aufgrund doppelter Client-IDs beim Senden oder Empfangen von Nachrichten zu ungültigen Sitzungsfehlern kommen kann. Dieses Problem führte dazu, dass die Sitzung des Clients geschlossen wurde.
2.3.3	Neue Features Fügt eine neue `startupTimeoutSeconds` Konfigurationsoption hinzu.
2.3.2	Die Version wurde für die Version 2.4.0 der Client-Geräteauthentifizierung aktualisiert.
2.3.1	Fehlerkorrekturen und Verbesserungen • Behebt ein Race-Problem, bei dem die Verbindung von Clients nach einem erneuten Verbindungsversuch aufgrund einer ungültigen Sitzung unterbrochen werden kann.

Version	Änderungen
2.3.0	Fügt Unterstützung für Zertifikatsketten hinzu.
2.2.0	Die Version wurde für die Version 2.2.0 der Client-Geräteauthentifizierung aktualisiert.
2.1.0	Fehlerkorrekturen und Verbesserungen • Aktualisiert diese Komponente auf die Verwendung von Moquette Version 0.16, was die Leistung verbessert und mehrere andere Verbesserungen beinhaltet. • Behebt ein Problem, bei dem das lokale MQTT-Serverzertifikat in bestimmten Szenarien häufiger rotiert als vorgesehen. Um diesen Fix anzuwenden, müssen Sie auch Version 2.1.0 oder höher der Authentifizierungskomponente für Clientgeräte verwenden.
2.0.2	Fehlerkorrekturen und Verbesserungen • Erhöht die maximale MQTT-Nachrichtengröße von 8.092 Byte auf 128 Kilobyte. Das effektive Limit für die Nutzlast von MQTT-Nachrichten ist etwas geringer, da das Limit für die Nachrichtengröße Nachricht enkopfzeilen einschließt. • Fügt Unterstützung für Ganzzahlwerte im Parameter hinzu. `ssl_port`
2.0.1	Die Version wurde für die Version 2.4.0 von Greengrass Nucleus aktualisiert.
2.0.0	Erste Version

MQTT 5-Broker (EMAX)

Die EMQX MQTT-Broker-Komponente (`aws.greengrass.clientdevices.mqtt.EMQX`) verarbeitet MQTT-Nachrichten zwischen Client-Geräten und einem Greengrass-Core-Gerät. Diese Komponente stellt eine modifizierte Version des EMQX MQTT 5.0-Brokers bereit. Stellen Sie diesen MQTT-Broker bereit, um MQTT 5-Funktionen bei der Kommunikation zwischen Client-Geräten und einem Core-Gerät zu verwenden. Weitere Informationen zur Auswahl eines MQTT-Brokers finden Sie unter. Auswählen eines MQTT-Brokers

Dieser Broker implementiert das MQTT 5.0-Protokoll. Es unterstützt Ablaufintervalle für Sitzungen und Nachrichten, Benutzereigenschaften, gemeinsame Abonnements, Themenaliase und mehr. MQTT 5 ist abwärtskompatibel mit MQTT 3.1.1. Wenn Sie also den Moquette MQTT 3.1.1-Broker ausführen, können Sie ihn durch den EMQX MQTT 5-Broker ersetzen, und die Client-Geräte können weiterhin eine Verbindung herstellen und wie gewohnt arbeiten.

> **Note**
>
> Client-Geräte sind lokale IoT-Geräte, die eine Verbindung zu einem Greengrass-Core-Gerät herstellen, um MQTT-Nachrichten und Daten zur Verarbeitung zu senden. Weitere Informationen finden Sie unter Interagieren mit lokalen IoT-Geräten.

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.0.x
- 1.2.x
- 1.1.x
- 1.0.x

Typ

Diese Komponente ist eine generische Komponente (`aws.greengrass.generic`). Der Greengrass-Kern führt die Lebenszyklusskripte der Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf Kerngeräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Das Kerngerät muss in der Lage sein, Verbindungen an dem Port anzunehmen, an dem der MQTT-Broker arbeitet. Diese Komponente führt den MQTT-Broker standardmäßig auf Port 8883 aus. Sie können bei der Konfiguration dieser Komponente einen anderen Port angeben.

  Wenn Sie einen anderen Port angeben und die MQTT-Bridge-Komponente verwenden, um MQTT-Nachrichten an andere Broker weiterzuleiten, müssen Sie MQTT Bridge v2.1.0 oder höher verwenden. Konfigurieren Sie es so, dass es den Port verwendet, an dem der MQTT-Broker arbeitet.

  Wenn Sie einen anderen Port angeben und die IP-Detector-Komponente zur Verwaltung von MQTT-Broker-Endpunkten verwenden, müssen Sie den IP-Detektor v2.1.0 oder höher verwenden. Konfigurieren Sie es so, dass der Port gemeldet wird, an dem der MQTT-Broker arbeitet.
- Auf Linux-Core-Geräten wurde Docker auf dem Core-Gerät installiert und konfiguriert:
  - Docker Engine 1.9.1 oder höher ist auf dem Greengrass-Core-Gerät installiert. Version 20.10 ist die neueste Version, für die verifiziert wurde, dass sie mit der Core-Software funktioniert. AWS IoT Greengrass Sie müssen Docker direkt auf dem Kerngerät installieren, bevor Sie Komponenten bereitstellen, auf denen Docker-Container ausgeführt werden.
  - Der Docker-Daemon wurde auf dem Kerngerät gestartet und ausgeführt, bevor Sie diese Komponente bereitstellen.

- Der Systembenutzer, der diese Komponente ausführt, muss über Root- oder Administratorrechte verfügen. Alternativ können Sie diese Komponente als Systembenutzer in der `docker` Gruppe ausführen und die `requiresPrivileges` Option dieser Komponente so konfigurieren, dass der EQMX MQTT-Broker ohne Rechte ausgeführt wird. `false`

- Die EMQX MQTT-Broker-Komponente wird für die Ausführung in einer VPC unterstützt.
- Die EMQX MQTT-Broker-Komponente wird auf der Plattform nicht unterstützt. `armv7`

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt sie AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und all ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitstellen zu können. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente sowie die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste der Abhängigkeiten.

2.0.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Authentifizierung auf dem Client-Gerät	>=2.2.0 <2.6.0	Hart

2.0.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Authentifizierung auf dem Client-Gerät	>=2.2.0 <2.5.0	Hart

1.2.2 – 1.2.3

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 1.2.2 bis 1.2.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Authentifizierung auf dem Client-Gerät	>=2.2.0 <2.5.0	Hart

1.2.0 and 1.2.1

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 1.2.0 und 1.2.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Authentifizierung auf dem Client-Gerät	>=2.2.0 <2.4.0	Hart

1.0.0 and 1.1.0

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 1.0.0 und 1.1.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Authentifizierung auf dem Client-Gerät	>=2.2.0 <2.3.0	Hart

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zu den Komponentenrezepten.

Konfiguration

2.0.0 - 2.0.1

Diese Komponente stellt die folgenden Konfigurationsparameter bereit, die Sie bei der Bereitstellung der Komponente anpassen können.

> **Important**
>
> Wenn Sie Version 2 der MQTT 5-Broker-Komponente (EMQX) verwenden, müssen Sie Ihre Konfigurationsdatei aktualisieren. Die Konfigurationsdateien der Version 1 funktionieren nicht mit Version 2.

emqxConfig

(Optional) Die zu verwendende EMQX MQTT-Broker-Konfiguration. Sie können EMQX-Konfigurationsoptionen in dieser Komponente festlegen.

Wenn Sie den EMQX-Broker verwenden, verwendet Greengrass eine Standardkonfiguration. Diese Konfiguration wird verwendet, sofern Sie sie nicht mithilfe dieses Felds ändern.

Wenn Sie die folgenden Konfigurationseinstellungen ändern, wird die EMQX-Broker-Komponente neu gestartet. Andere Konfigurationsänderungen gelten, ohne dass die Komponente neu gestartet wird.

- `emqxConfig/cluster`
- `emqxConfig/node`
- `emqxConfig/rpc`

> **Note**
>
> `aws.greengrass.clientdevices.mqtt.EMQX`ermöglicht es Ihnen, sicherheitsrelevante Optionen zu konfigurieren. Dazu gehören TLS-Einstellungen, Authentifizierungs- und Autorisierungsanbieter. Wir haben die Standardkonfiguration empfohlen, die die gegenseitige TLS-Authentifizierung und den Greengrass-Client-Geräteauthentifizierungsanbieter verwendet.

Example Beispiel: Standardkonfiguration

Das folgende Beispiel zeigt die für den MQTT 5 (EMQX) -Broker festgelegten Standardeinstellungen. Sie können diese Einstellungen mithilfe der Konfigurationseinstellung überschreiben. `emqxConfig`

```
{
  "authorization": {
    "no_match": "deny",
    "sources": []
  },
  "node": {
    "cookie": "<placeholder>"
  },
  "listeners": {
     "ssl": {
       "default": {
         "ssl_options": {
           "keyfile": "{work:path}\\data\\key.pem",
           "certfile": "{work:path}\\data\\cert.pem",
           "cacertfile": null,
           "verify": "verify_peer",
           "versions": ["tlsv1.3", "tlsv1.2"],
           "fail_if_no_peer_cert": true
         }
       }
     },
     "tcp": {
       "default": {
         "enabled": false
       }
     },
     "ws": {
       "default": {
         "enabled": false
       }
     },
     "wss": {
       "default": {
         "enabled": false
       }
     }
  },
```

```
  "plugins": {
    "states": [{"name_vsn": "gg-1.0.0", "enable": true}],
    "install_dir": "plugins"
  }
}
```

AuthMode

(Optional) Legt den Autorisierungsanbieter für den Broker fest. Dabei kann es sich um einen der folgenden Werte handeln:

- `enabled`— (Standard) Verwenden Sie den Greengrass-Authentifizierungs- und Autorisierungsanbieter.
- `bypass_on_failure`— Verwenden Sie den Greengrass-Authentifizierungsanbieter und dann alle verbleibenden Authentifizierungsanbieter in der EMQX-Anbieterkette, falls Greengrass entweder die Authentifizierung oder Autorisierung verweigert.
- `bypass`— Der Greengrass-Anbieter ist deaktiviert. Authentifizierung und Autorisierung werden von der EMQX-Anbieterkette abgewickelt.

`requiresPrivilege`

(Optional) Auf Linux-Core-Geräten können Sie angeben, dass der EMQX MQTT-Broker ohne Root- oder Administratorrechte ausgeführt werden soll. Wenn Sie diese Option auf setzen`false`, muss der Systembenutzer, der diese Komponente ausführt, Mitglied der Gruppe sein. `docker`

Standard: `true`

`startupTimeoutSeconds`

(Optional) Die maximale Zeit in Sekunden, die der EMQX MQTT-Broker zum Starten benötigt. Der Status der Komponente ändert sich auf, `BROKEN` wenn dieser Timeout überschritten wird.

Standard: `90`

`ipcTimeoutSeconds`

(Optional) Die maximale Zeit in Sekunden, die die Komponente benötigt, um darauf zu warten, dass der Greengrass-Kern auf IPC-Anfragen (Interprocess Communication) reagiert. Erhöhen Sie diesen Wert, wenn diese Komponente bei der Überprüfung, ob ein Client-Gerät autorisiert ist, Timeout-Fehler meldet.

Standard: 5

`crtLogLevel`

(Optional) Die Protokollebene für die AWS Common Runtime (CRT) -Bibliothek.

Standardmäßig wird die Protokollebene (in) des EMQX MQTT-Brokers verwendet. `log.level` `emqx`

`restartIdentifier`

(Optional) Konfigurieren Sie diese Option, um den EMQX MQTT-Broker neu zu starten. Wenn sich dieser Konfigurationswert ändert, startet diese Komponente den MQTT-Broker neu. Sie können diese Option verwenden, um zu erzwingen, dass Client-Geräte die Verbindung trennen.

`dockerOptions`

(Optional) Konfigurieren Sie diese Option nur auf Linux-Betriebssystemen, um der Docker-Befehlszeile Parameter hinzuzufügen. Um beispielsweise zusätzliche Ports zuzuordnen, verwenden Sie den `-p` Docker-Parameter:

```
"-p 1883:1883"
```

Example Beispiel: Aktualisierung einer v1.x-Konfigurationsdatei auf v2.x

Das folgende Beispiel zeigt die Änderungen, die erforderlich sind, um eine v1.x-Konfigurationsdatei auf Version 2.x zu aktualisieren.

Die Konfigurationsdatei der Version 1.x:

```
{
    "emqx": {
        "listener.ssl.external": "443",
        "listener.ssl.external.max_connections": "1024000",
        "listener.ssl.external.max_conn_rate": "500",
        "listener.ssl.external.rate_limit": "50KB,5s",
        "listener.ssl.external.handshake_timeout": "15s",
        "log.level": "warning"
    },
    "mergeConfigurationFiles": {
        "etc/plugins/aws_greengrass_emqx_auth.conf": "auth_mode=enabled\n
 use_greengrass_managed_certificates=true\n"
```

```
    }
}
```

Die entsprechende Konfigurationsdatei für v2:

```
{
    "emqxConfig": {
        "listeners": {
            "ssl": {
                "default": {
                   "bind": "8883",
                   "max_connections": "1024000",
                   "max_conn_rate": "500",
                   "handshake_timeout": "15s"
                }
            }
        },
        "log": {
            "console": {
              "enable": true,
              "level": "warning"
            }
        }
    },
    "authMode": "enabled"
}
```

Es gibt kein Äquivalent zum `listener.ssl.external.rate_limit` Konfigurationseintrag. Die `use_greengrass_managed_certificates` Konfigurationsoption wurde entfernt.

Example Beispiel: Legen Sie einen neuen Port für den Broker fest

Das folgende Beispiel ändert den Port, an dem der MQTT-Broker arbeitet, vom Standardport 8883 auf den Port 1234. Wenn Sie Linux verwenden, fügen Sie das `dockerOptions` Feld hinzu.

```
{
  "emqxConfig": {
    "listeners": {
      "ssl": {
        "default": {
          "bind": 1234
        }
```

```
      }
    }
  },
  "dockerOptions": "-p 1234:1234"
}
```

Example Beispiel: Passen Sie den Log-Level des MQTT-Brokers an

Im folgenden Beispiel wird die Protokollebene des MQTT-Brokers auf geändert. `debug` Sie können aus den folgenden Protokollebenen wählen:

- `debug`
- `info`
- `notice`
- `warning`
- `error`
- `critical`
- `alert`
- `emergency`

Die Standard-Protokollebene ist`warning`.

```
{
  "emqxConfig": {
    "log": {
      "console": {
         "level": "debug"
      }
    }
  }
}
```

Example Beispiel: Aktivieren Sie das EMQX-Dashboard

Das folgende Beispiel aktiviert das EMQX-Dashboard, sodass Sie Ihren Broker überwachen und verwalten können. Wenn Sie Linux verwenden, fügen Sie das `dockerOptions` Feld hinzu.

```
{
```

```
  "emqxConfig": {
    "dashboard": {
      "listeners": {
        "http": {
          "bind": 18083
        }
      }
    }
  },
  "dockerOptions": "-p 18083:18083"
}
```

1.0.0 - 1.2.2

Diese Komponente stellt die folgenden Konfigurationsparameter bereit, die Sie bei der Bereitstellung der Komponente anpassen können.

`emqx`

(Optional) Die zu verwendende EMQX MQTT-Broker-Konfiguration. Sie können eine Teilmenge der EMQX-Konfigurationsoptionen in dieser Komponente konfigurieren.

Dieses Objekt enthält die folgenden Informationen:

`listener.ssl.external`

(Optional) Der Port, an dem der MQTT-Broker arbeitet.

> **Note**
>
> Wenn Sie einen anderen Port angeben und die MQTT-Bridge-Komponente verwenden, um MQTT-Nachrichten an andere Broker weiterzuleiten, müssen Sie MQTT Bridge v2.1.0 oder höher verwenden. Konfigurieren Sie es so, dass es den Port verwendet, an dem der MQTT-Broker arbeitet.
> Wenn Sie einen anderen Port angeben und die IP-Detector-Komponente zur Verwaltung von MQTT-Broker-Endpunkten verwenden, müssen Sie den IP-Detektor v2.1.0 oder höher verwenden. Konfigurieren Sie es so, dass der Port gemeldet wird, an dem der MQTT-Broker arbeitet.

Standard: `8883`

`listener.ssl.external.max_connections`

(Optional) Die maximale Anzahl gleichzeitiger Verbindungen, die der MQTT-Broker unterstützt.

Standard: `1024000`

`listener.ssl.external.max_conn_rate`

(Optional) Die maximale Anzahl neuer Verbindungen pro Sekunde, die der MQTT-Broker empfangen kann.

Standard: `500`

`listener.ssl.external.rate_limit`

(Optional) Das Bandbreitenlimit für alle Verbindungen zum MQTT-Broker. Geben Sie die Bandbreite und Dauer für diese Bandbreite, getrennt durch ein Komma (`,`), im folgenden Format an:. `bandwidth,duration` Sie können beispielsweise angeben`50KB,5s`, dass der MQTT-Broker alle 5 Sekunden auf 50 Kilobyte (KB) an Daten begrenzt werden soll.

`listener.ssl.external.handshake_timeout`

(Optional) Die Zeitspanne, die der MQTT-Broker wartet, bis die Authentifizierung einer neuen Verbindung abgeschlossen ist.

Standard: `15s`

`mqtt.max_packet_size`

(Optional) Die maximale Größe einer MQTT-Nachricht.

Standard: `268435455` (256 MB minus 1)

`log.level`

(Optional) Die Protokollebene für den MQTT-Broker. Wählen Sie aus den folgenden Optionen aus:

- `debug`
- `info`
- `notice`
- `warning`
- `error`

- `critical`
- `alert`
- `emergency`

Die Standard-Protokollebene ist`warning`.

`requiresPrivilege`

(Optional) Auf Linux-Core-Geräten können Sie angeben, dass der EMQX MQTT-Broker ohne Root- oder Administratorrechte ausgeführt werden soll. Wenn Sie diese Option auf setzen`false`, muss der Systembenutzer, der diese Komponente ausführt, Mitglied der Gruppe sein. `docker`

Standard: `true`

`startupTimeoutSeconds`

(Optional) Die maximale Zeit in Sekunden, die der EMQX MQTT-Broker zum Starten benötigt. Der Status der Komponente ändert sich auf, `BROKEN` wenn dieser Timeout überschritten wird.

Standard: `90`

`ipcTimeoutSeconds`

(Optional) Die maximale Zeit in Sekunden, die die Komponente benötigt, um darauf zu warten, dass der Greengrass-Kern auf IPC-Anfragen (Interprocess Communication) reagiert. Erhöhen Sie diesen Wert, wenn diese Komponente bei der Überprüfung, ob ein Client-Gerät autorisiert ist, Timeout-Fehler meldet.

Standard: `5`

`crtLogLevel`

(Optional) Die Protokollebene für die AWS Common Runtime (CRT) -Bibliothek.

Standardmäßig wird die Protokollebene (in) des EMQX MQTT-Brokers verwendet. `log.level` `emqx`

`restartIdentifier`

(Optional) Konfigurieren Sie diese Option, um den EMQX MQTT-Broker neu zu starten. Wenn sich dieser Konfigurationswert ändert, startet diese Komponente den MQTT-Broker neu. Sie können diese Option verwenden, um zu erzwingen, dass Client-Geräte die Verbindung trennen.

`dockerOptions`

(Optional) Konfigurieren Sie diese Option nur auf Linux-Betriebssystemen, um der Docker-Befehlszeile Parameter hinzuzufügen. Um beispielsweise zusätzliche Ports zuzuordnen, verwenden Sie den `-p` Docker-Parameter:

```
"-p 1883:1883"
```

`mergeConfigurationFiles`

(Optional) Konfigurieren Sie diese Option, um die Standardeinstellungen in den angegebenen EMQX-Konfigurationsdateien zu ergänzen oder zu überschreiben. Informationen zu den Konfigurationsdateien und ihren Formaten finden Sie unter Konfiguration in der EMQX 4.0-Dokumentation. Die von Ihnen angegebenen Werte werden an die Konfigurationsdatei angehängt.

Das folgende Beispiel aktualisiert die `etc/emqx.conf` Datei.

```
"mergeConfigurationFiles": {
    "etc/emqx.conf": "broker.sys_interval=30s\nbroker.sys_heartbeat=10s"
},
```

Zusätzlich zu den von EMQX unterstützten Konfigurationsdateien unterstützt Greengrass eine Datei, die das Greengrass-Authentifizierungs-Plugin für EMQX konfiguriert namens. `etc/plugins/aws_greengrass_emqx_auth.conf` Es gibt zwei unterstützte Optionen, und. `auth_mode` `use_greengrass_managed_certificates` Um einen anderen Authentifizierungsanbieter zu verwenden, setzen Sie die `auth_mode` Option auf einen der folgenden Werte:

- `enabled`— (Standard) Verwenden Sie den Greengrass-Authentifizierungs- und Autorisierungsanbieter.
- `bypass_on_failure`— Verwenden Sie den Greengrass-Authentifizierungsanbieter und dann alle verbleibenden Authentifizierungsanbieter in der EMQX-Anbieterkette, falls Greengrass entweder die Authentifizierung oder Autorisierung verweigert.
- `bypass`— Der Greengrass-Anbieter ist deaktiviert. Authentifizierung und Autorisierung werden dann von der EMQX-Anbieterkette abgewickelt.

Wenn ja`use_greengrass_managed_certificates`, bedeutet diese Option`true`, dass Greengrass die Broker-TLS-Zertifikate verwaltet. Falls`false`, bedeutet dies, dass Sie die Zertifikate über eine andere Quelle bereitstellen.

Im folgenden Beispiel werden die Standardeinstellungen in der `etc/plugins/aws_greengrass_emqx_auth.conf` Konfigurationsdatei aktualisiert.

```
"mergeConfigurationFiles": {
    "etc/plugins/aws_greengrass_emqx_auth.conf": "auth_mode=enabled\n
 use_greengrass_managed_certificates=true\n"
  },
```

> **Note**
>
> `aws.greengrass.clientdevices.mqtt.EMQX`ermöglicht es Ihnen, sicherheitsrelevante Optionen zu konfigurieren. Dazu gehören TLS-Einstellungen, Authentifizierungs- und Autorisierungsanbieter. Die empfohlene Konfiguration ist die Standardkonfiguration, die die gegenseitige TLS-Authentifizierung und den Greengrass Client Device Auth Provider verwendet.

`replaceConfigurationFiles`

(Optional) Konfigurieren Sie diese Option so, dass sie die angegebenen EMQX-Konfigurationsdateien ersetzt. Die von Ihnen angegebenen Werte ersetzen die gesamte vorhandene Konfigurationsdatei. Sie können die `etc/emqx.conf` Datei in diesem Abschnitt nicht angeben. Sie müssen `mergeConfigurationFile` zum Ändern verwenden`etc/emqx.conf`.

Example Beispiel: Aktualisierung der Konfigurationszusammenführung

Die folgende Beispielkonfiguration legt fest, dass der MQTT-Broker auf Port 443 betrieben werden soll.

```
{
  "emqx": {
    "listener.ssl.external": "443",
    "listener.ssl.external.max_connections": "1024000",
    "listener.ssl.external.max_conn_rate": "500",
    "listener.ssl.external.rate_limit": "50KB,5s",
    "listener.ssl.external.handshake_timeout": "15s",
    "log.level": "warning"
  },
  "requiresPrivilege": "true",
```

```
  "startupTimeoutSeconds": "90",
  "ipcTimeoutSeconds": "5"
}
```

Lokale Protokolldatei

Diese Komponente verwendet die folgende Protokolldatei.

Linux

```
/greengrass/v2/logs/aws.greengrass.clientdevices.mqtt.EMQX.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.clientdevices.mqtt.EMQX.log
```

Um die Protokolle dieser Komponente einzusehen

- Führen Sie den folgenden Befehl auf dem Kerngerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.clientdevices.mqtt.EMQX.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.clientdevices.mqtt.EMQX.log -
  Tail 10 -Wait
  ```

Lizenzen

Auf Windows-Betriebssystemen enthält diese Software Code, der unter den Microsoft-Softwarelizenzbedingungen - Microsoft Visual Studio Community 2022 vertrieben wird. Durch das Herunterladen dieser Software erklären Sie sich mit den Lizenzbedingungen dieses Codes einverstanden.

Diese Komponente wird im Rahmen der Greengrass Core Software-Lizenzvereinbarung veröffentlicht.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen der Komponente beschrieben.

v2.x

Version	Änderungen
2.0.1	Die Version wurde für die Version 2.5.0 der Client-Geräteauthentifizierung aktualisiert.
2.0.0	Diese Version des MQTT 5-Brokers (EMQX) erwartet andere Konfigura tionsparameter als Version 1.x. Wenn Sie eine nicht standardmäßige Konfiguration für Version 1.x verwenden, müssen Sie die Konfiguration der Komponente für 2.x aktualisieren. Weitere Informationen finden Sie unter Konfiguration. Neue Features • Führt ein Upgrade des MQTT-Brokers auf EMQX 5.1.1 durch. • Ermöglicht Änderungen der Broker-Konfiguration, ohne die Komponente neu zu starten. Aktualisierungen • Fügt ein neues `emqxConfig` Konfigurationsfeld hinzu, das die `replaceConfigurationFiles` Konfigurationsfelder `emqxmergeConfigurationFiles` , und ersetzt.

v1.x

Version	Änderungen
1.2.3	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem Clients nach einer vorherigen Authentif izierung nicht mit EMQX interagieren konnten, indem der Client getrennt und erneut authentifiziert wurde.
1.2.2	Die Version wurde für die Version 2.4.0 der Client-Geräteauthentifizierung aktualisiert.
1.2.1	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem die Komponente unter Windows nicht gestartet werden kann, wenn Visual C++ Redistributable nicht bereits vorhanden ist. • Aktualisiert EMQX auf Version 4.4.14.
1.2.0	Fügt Unterstützung für Zertifikatsketten hinzu.
1.1.0	Neue Features • Integriert die Unterstützung für EMQX-Konfigurationen, einschließlich Broker-Optionen und Plug-ins. Fehlerkorrekturen und Verbesserungen • Aktualisiert EMQX auf Version 4.4.9.
1.0.1	Behebt ein Problem beim TLS-Handshake, das dazu führt, dass einige MQTT-Clients keine Verbindung herstellen können.
1.0.0	Erste Version

Telemetrie-Emitter

Die Kerntelemetrie-Emitterkomponente (`aws.greengrass.telemetry.NucleusEmitter`) sammelt Telemetriedaten zum Systemzustand und veröffentlicht sie kontinuierlich in einem lokalen Thema und einem AWS IoT Core MQTT-Thema. Mit dieser Komponente können Sie Echtzeit-Systemtelemetrie auf Ihren Greengrass-Core-Geräten erfassen. Informationen zum Greengrass-

Telemetrieagenten, der Systemtelemetriedaten in Amazon veröffentlicht EventBridge, finden Sie unter Erfassen von Telemetriedaten zum Systemstatus von -AWS IoT GreengrassCore-Geräten.

Standardmäßig veröffentlicht die Komponente des Telemetrie-Emitters alle 60 Sekunden Telemetriedaten im folgenden lokalen Veröffentlichungs-/Abonnementthema.

```
$local/greengrass/telemetry
```

Die Kerntelemetrie-Emitterkomponente veröffentlicht standardmäßig nicht zu einem AWS IoT Core MQTT-Thema. Sie können diese Komponente so konfigurieren, dass sie bei der Bereitstellung in einem AWS IoT Core MQTT-Thema veröffentlicht. Die Verwendung eines MQTT-Themas zum Veröffentlichen von Daten in der AWS Cloud unterliegt den AWS IoT Core Preisen .

AWS IoT Greengrass bietet mehrere Community-Komponenten, mit denen Sie Telemetriedaten mithilfe von InfluxDB und Grafana lokal auf Ihrem Core-Gerät analysieren und visualisieren können. Diese Komponenten verwenden Telemetriedaten aus der Kern-Emitterkomponente. Weitere Informationen finden Sie in der README für die Herausgeberkomponente von InfluxDB.

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 1.0.x

Typ

Diese Komponente ist eine Plugin-Komponente (`aws.greengrass.plugin`). Der Greengrass-Kern führt diese Komponente in derselben Java Virtual Machine (JVM) wie der Kern aus. Der Kern wird neu gestartet, wenn Sie die Version dieser Komponente auf dem Core-Gerät ändern.

Diese Komponente verwendet dieselbe Protokolldatei wie der Greengrass-Kern. Weitere Informationen finden Sie unter Überwachen von AWS IoT Greengrass Protokollen.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf -Core-Geräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und alle ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitzustellen. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente und die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste Abhängigkeiten.

1.0.8

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.0.8 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.4.0 <2.13.0	Hart

1.0.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.0.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.4.0 <2.12.0	Hart

1.0.6

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.0.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.4.0 <2.11.0	Hart

1.0.5

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.0.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.4.0 <2.10.0	Hart

1.0.4

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.0.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.4.0 <2.9.0	Hart

1.0.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.0.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.4.0 <2.8.0	Hart

1.0.2

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.0.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.4.0 <2.7.0	Hart

1.0.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.0.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.4.0 <2.6.0	Hart

1.0.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.0.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.4.0 <2.5.0	Hart

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zum Komponentenrezept .

Konfiguration

Diese Komponente bietet die folgenden Konfigurationsparameter, die Sie anpassen können, wenn Sie die Komponente bereitstellen.

`pubSubPublish`

(Optional) Definiert, ob Telemetriedaten zum `$local/greengrass/telemetry` Thema veröffentlicht werden sollen. Unterstützte Werte sind `true` und `false`.

Standard: `true`

`mqttTopic`

(Optional) Das AWS IoT Core MQTT-Thema, zu dem diese Komponente Telemetriedaten veröffentlicht.

Legen Sie diesen Wert auf das AWS IoT Core MQTT-Thema fest, zu dem Sie Telemetriedaten veröffentlichen möchten. Wenn dieser Wert leer ist, veröffentlicht der Kern-Emitter keine Telemetriedaten in der AWS Cloud.

> **Note**
>
> Die Verwendung eines MQTT-Themas zum Veröffentlichen von Daten in der AWS Cloud unterliegt den AWS IoT Core Preisen .

Standard: `""`

`telemetryPublishIntervalMs`

(Optional) Die Zeitspanne (in Millisekunden), zwischen der die Komponente Telemetriedaten veröffentlicht. Wenn Sie diesen Wert niedriger als den unterstützten Mindestwert festlegen, verwendet die Komponente stattdessen den Mindestwert.

> **Note**
>
> Niedrigere Veröffentlichungsintervalle führen zu einer höheren CPU-Auslastung auf Ihrem Core-Gerät. Wir empfehlen Ihnen, mit dem Standard-Veröffentlichungsintervall zu beginnen und es an die CPU-Auslastung Ihres Geräts anzupassen.

Minimum: `500`

Standard: `60000`

Example Beispiel: Aktualisierung der Konfigurationszusammenführung

Das folgende Beispiel zeigt ein Beispiel für eine Aktualisierung der Konfigurationszusammenführung, die die Veröffentlichung von Telemetriedaten alle 5 Sekunden für das `$local/greengrass/telemetry` Thema und das `greengrass/myTelemetry` AWS IoT Core MQTT-Thema ermöglicht.

```
{
  "pubSubPublish": "true",
  "mqttTopic": "greengrass/myTelemetry",
  "telemetryPublishIntervalMs": 5000
}
```

Ausgabedaten

Diese Komponente veröffentlicht Telemetriemetriken als JSON-Array im folgenden Thema.

Lokales Thema: `$local/greengrass/telemetry`

Optional können Sie auch Telemetriemetriken in einem AWS IoT Core MQTT-Thema veröffentlichen. Weitere Informationen zu -Themen finden Sie unter MQTT-Themen im AWS IoT Core - Entwicklerhandbuch.

Example Beispiel für Daten

```
[
  {
    "A": "Average",
    "N": "CpuUsage",
    "NS": "SystemMetrics",
    "TS": 1627597331445,
    "U": "Percent",
    "V": 26.21981271562346
  },
  {
    "A": "Count",
    "N": "TotalNumberOfFDs",
    "NS": "SystemMetrics",
    "TS": 1627597331445,
    "U": "Count",
    "V": 7316
  },
  {
    "A": "Count",
```

```
    "N": "SystemMemUsage",
    "NS": "SystemMetrics",
    "TS": 1627597331445,
    "U": "Megabytes",
    "V": 10098
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsStarting",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsInstalled",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsStateless",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsStopping",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsBroken",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
```

```
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsRunning",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 7
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsErrored",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsNew",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsFinished",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 2
  }
]
```

Das Ausgabe-Array enthält eine Liste von Metriken mit den folgenden Eigenschaften:

A

Der Aggregationstyp für die Metrik.

Für die -`CpuUsage`Metrik ist diese Eigenschaft auf gesetzt, `Average` da der veröffentlichte Wert der Metrik die durchschnittliche CPU-Auslastung seit dem letzten Veröffentlichungsereignis ist.

Für alle anderen Metriken aggregiert der Kern-Emitter den Metrikwert nicht, und diese Eigenschaft ist auf festgelegt`Count`.

`N`

Name der Metrik.

`NS`

Der Metrik-Namespace.

`TS`

Der Zeitstempel der Datenerfassung.

`U`

Die Einheit des Metrikwerts.

`V`

Der -Metrikwert

Der Kern-Emitter veröffentlicht die folgenden Metriken:

Name	Beschreibung
System (System)	
`SystemMemUsage`	Die Speichermenge, die derzeit von allen Anwendungen auf dem Greengrass-Core-Gerät verwendet wird, einschließlich des Betriebssystems.
`CpuUsage`	Die Menge an CPU, die derzeit von allen Anwendungen auf dem Greengrass-Core-Gerät verwendet wird, einschließlich des Betriebssystems.

Name	Beschreibung
`TotalNumberOfFDs`	Die Anzahl der vom Betriebssystem des Greengrass-Core-Geräts gespeicherten Dateideskriptoren. Ein Dateideskriptor identifiziert eindeutig eine geöffnete Datei.
Greengrass-Kern	
`NumberOfComponentsRunning`	Die Anzahl der Komponenten, die auf dem Greengrass-Kerngerät ausgeführt werden.
`NumberOfComponentsErrored`	Die Anzahl der Komponenten, die sich auf dem Greengrass-Core-Gerät im Fehlerzustand befinden.
`NumberOfComponentsInstalled`	Die Anzahl der Komponenten, die auf dem Greengrass-Core-Gerät installiert sind.
`NumberOfComponentsStarting`	Die Anzahl der Komponenten, die auf dem Greengrass-Kerngerät beginnen.
`NumberOfComponentsNew`	Die Anzahl der Komponenten, die auf dem Greengrass-Kerngerät neu sind.
`NumberOfComponentsStopping`	Die Anzahl der Komponenten, die auf dem Greengrass-Kerngerät anhalten.
`NumberOfComponentsFinished`	Die Anzahl der Komponenten, die auf dem Greengrass-Kerngerät fertig sind.

Name	Beschreibung	
`NumberOfComponentsBroken`	Die Anzahl der Komponenten, die auf dem Greengrass-Kerngerät unterbrochen sind.	
`NumberOfComponentsStateless`	Die Anzahl der Komponenten, die auf dem Greengrass-Kerngerät zustandslos sind.	

Verwendung

Um Telemetriedaten des Systemzustands zu verwenden, können Sie benutzerdefinierte Komponenten erstellen, die die Themen abonnieren, zu denen der Kern-Emitter die Telemetriedaten veröffentlicht, und bei Bedarf auf diese Daten reagieren. Da die Kern-Emitter-Komponente die Möglichkeit bietet, Telemetriedaten in einem lokalen Thema zu veröffentlichen, können Sie dieses Thema abonnieren und die veröffentlichten Daten verwenden, um lokal auf Ihrem Core-Gerät zu handeln. Das Core-Gerät kann dann auf Telemetriedaten reagieren, auch wenn es über eine eingeschränkte Konnektivität zur Cloud verfügt.

Sie können beispielsweise eine Komponente konfigurieren, die das `$local/greengrass/telemetry` Thema auf Telemetriedaten überwacht, und die Daten an die Stream-Manager-Komponente senden, um Ihre Daten an die zu streamenAWS Cloud. Weitere Informationen zum Erstellen einer solchen Komponente finden Sie unter Lokale Nachrichten veröffentlichen/abonnieren und Erstellen Sie benutzerdefinierte Komponenten, die Stream Manager verwenden.

Lokale Protokolldatei

Diese Komponente verwendet dieselbe Protokolldatei wie die Greengrass-Kernkomponente.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

So zeigen Sie die Protokolle dieser Komponente an

- Führen Sie den folgenden Befehl auf dem Core-Gerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

  Linux

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

  Windows (PowerShell)

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

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in jeder Version der Komponente beschrieben.

Version	Änderungen
1.0.8	Version für Greengrass-Kern Version 2.12.0 aktualisiert.
1.0.7	Version für Greengrass-Kern Version 2.11.0 aktualisiert.
1.0.6	Version für Greengrass-Kern Version 2.10.0 aktualisiert.
1.0.5	Version für Greengrass-Kern Version 2.9.0 aktualisiert.
1.0.4	Version für Greengrass-Kern Version 2.8.0 aktualisiert.
1.0.3	Version für Greengrass-Kern Version 2.7.0 aktualisiert.
1.0.2	Version für Greengrass-Kern Version 2.6.0 aktualisiert.
1.0.1	Version für Greengrass-Kern Version 2.5.0 aktualisiert.
1.0.0	Erste Version

PKCS #11-Anbieter

Mit der PKCS#11-Anbieterkomponente (`aws.greengrass.crypto.Pkcs11Provider`) können Sie die -AWS IoT GreengrassCore-Software so konfigurieren, dass sie ein Hardware-Sicherheitsmodul (HSM) über die PKCS#11-Schnittstelle verwendet. Mit dieser Komponente können Sie Zertifikats- und private Schlüsseldateien sicher speichern, sodass sie nicht in der Software verfügbar gemacht oder dupliziert werden. Weitere Informationen finden Sie unter Integration von Hardware-Sicherheit.

Um ein Greengrass-Core-Gerät bereitzustellen, das sein Zertifikat und seinen privaten Schlüssel in einem HSM speichert, müssen Sie diese Komponente als Bereitstellungs-Plugin angeben, wenn Sie die AWS IoT Greengrass Core-Software installieren. Weitere Informationen finden Sie unter Installieren Sie die AWS IoT Greengrass Core-Software mit manueller Ressourcenbereitstellung.

AWS IoT Greengrass stellt diese Komponente als JAR-Datei bereit, die Sie herunterladen können, um sie während der Installation als Bereitstellungs-Plugin anzugeben. Sie können die neueste Version der JAR-Datei der Komponente unter der folgenden URL herunterladen: https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/aws.greengrass.crypto.Pkcs11Provider-latest.jar.

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.0.x

Typ

Diese Komponente ist eine Plugin-Komponente (`aws.greengrass.plugin`). Der Greengrass-Kern führt diese Komponente in derselben Java Virtual Machine (JVM) wie der Kern aus. Der Kern wird neu gestartet, wenn Sie die Version dieser Komponente auf dem Core-Gerät ändern.

Diese Komponente verwendet dieselbe Protokolldatei wie der Greengrass-Kern. Weitere Informationen finden Sie unter Überwachen von AWS IoT Greengrass Protokollen.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann nur auf Linux-Core-Geräten installiert werden.

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Ein Hardware-Sicherheitsmodul, das das PKCS#1 v1.5-Signaturschema und RSA-Schlüssel mit einer RSA-2048-Schlüsselgröße (oder größer) oder ECC-Schlüsseln unterstützt.

  > **Note**
  >
  > Um ein Hardware-Sicherheitsmodul mit ECC-Schlüsseln zu verwenden, müssen Sie Greengrass-Kern v2.5.6 oder höher verwenden.
  > Um ein Hardware-Sicherheitsmodul und einen Secret Manager zu verwenden, müssen Sie ein Hardware-Sicherheitsmodul mit RSA-Schlüsseln verwenden.

- Eine PKCS#11-Anbieterbibliothek, die die AWS IoT Greengrass -Core-Software zur Laufzeit (mit libdl) laden kann, um PKCS#11-Funktionen aufzurufen. Die PKCS#11-Anbieterbibliothek muss die folgenden PKCS#11-API-Operationen implementieren:
  - `C_Initialize`
  - `C_Finalize`
  - `C_GetSlotList`
  - `C_GetSlotInfo`
  - `C_GetTokenInfo`
  - `C_OpenSession`
  - `C_GetSessionInfo`

  - `C_CloseSession`
  - `C_Login`
  - `C_Logout`
  - `C_GetAttributeValue`
  - `C_FindObjectsInit`
  - `C_FindObjects`
  - `C_FindObjectsFinal`
  - `C_DecryptInit`
  - `C_Decrypt`
  - `C_DecryptUpdate`
  - `C_DecryptFinal`
  - `C_SignInit`
  - `C_Sign`
  - `C_SignUpdate`
  - `C_SignFinal`
  - `C_GetMechanismList`
  - `C_GetMechanismInfo`
  - `C_GetInfo`
  - `C_GetFunctionList`
- Das Hardwaremodul muss nach Slot-Label auflösbar sein, wie in der PKCS#11-Spezifikation definiert.
- Sie müssen den privaten Schlüssel und das Zertifikat im HSM im selben Slot speichern und dieselbe Objektbezeichnung und Objekt-ID verwenden, wenn das HSM Objekt-IDs unterstützt.
- Das Zertifikat und der private Schlüssel müssen durch Objektbezeichnungen auflösbar sein.
- Der private Schlüssel muss über die folgenden Berechtigungen verfügen:
  - `sign`
  - `decrypt`
- (Optional) Um die Secret-Manager-Komponente zu verwenden, müssen Sie Version 2.1.0 oder höher verwenden und der private Schlüssel muss über die folgenden Berechtigungen verfügen:
  - `unwrap`
  - `wrap`

- (Optional) Wenn Sie die TPM2-Bibliothek verwenden und den Greengrass Core als Service ausführen, müssen Sie eine Umgebungsvariable mit dem Speicherort des PKCS#11-Speichers angeben. Das folgende Beispiel ist eine systemd-Servicedatei mit der erforderlichen Umgebungsvariablen:

```
[Unit]
Description=Greengrass Core
After=network.target

[Service]
Type=simple
PIDFile=/var/run/greengrass.pid
Environment=TPM2_PKCS11_STORE=/path/to/store/directory
RemainAfterExit=no
Restart=on-failure
RestartSec=10
ExecStart=/bin/sh /greengrass/v2/alts/current/distro/bin/loader

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

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und alle ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitzustellen. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente und die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste Abhängigkeiten.

2.0.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.5.3 <2.13.0	Weich

2.0.6

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.5.3 <2.12.0	Weich

2.0.5

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.5.3 <2.11.0	Weich

2.0.4

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.5.3 <2.10.0	Weich

2.0.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.5.3 <2.9.0	Weich

2.0.2

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.5.3 <2.8.0	Weich

2.0.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.5.3 <2.7.0	Weich

2.0.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.5.3 <2.6.0	Weich

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zum Komponentenrezept .

Konfiguration

Diese Komponente bietet die folgenden Konfigurationsparameter, die Sie anpassen können, wenn Sie die Komponente bereitstellen.

`name`

Ein Name für die PKCS#11-Konfiguration.

`library`

Der absolute Dateipfad zur Bibliothek der PKCS#11-Implementierung, die die AWS IoT Greengrass Core-Software mit libdl laden kann.

`slot`

Die ID des Slots, der den privaten Schlüssel und das Gerätezertifikat enthält. Dieser Wert unterscheidet sich vom Slot-Index oder der Slot-Bezeichnung.

`userPin`

Die Benutzer-PIN, die für den Zugriff auf den Slot verwendet werden soll.

Example Beispiel: Aktualisierung der Konfigurationszusammenführung

```
{
  "name": "softhsm_pkcs11",
  "library": "/usr/lib/softhsm/libsofthsm2.so",
  "slot": 1,
  "userPin": "1234"
}
```

Lokale Protokolldatei

Diese Komponente verwendet dieselbe Protokolldatei wie die Greengrass-Kernkomponente.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

So zeigen Sie die Protokolle dieser Komponente an

- Führen Sie den folgenden Befehl auf dem Core-Gerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

  Linux

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

Windows (PowerShell)

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

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in jeder Version der Komponente beschrieben.

Version	Änderungen
2.0.7	Version für Greengrass-Kern Version 2.12.0 aktualisiert.
2.0.6	Version für Greengrass-Kern Version 2.11.0 aktualisiert.
2.0.5	Version für Greengrass-Kernversion 2.10.0 aktualisiert.
2.0.4	Version für Greengrass-Kern Version 2.9.0 aktualisiert.
2.0.3	Version für Greengrass-Kern Version 2.8.0 aktualisiert.
2.0.2	Version für Greengrass-Kern Version 2.7.0 aktualisiert.
2.0.1	Version für Greengrass-Kern Version 2.6.0 aktualisiert.
2.0.0	Erste Version

Geheimer Manager

Die Secret Manager-Komponente (`aws.greengrass.SecretManager`) verteilt Geheimnisse von AWS Secrets Manager Greengrass-Kerngeräten. Verwenden Sie diese Komponente, um Anmeldeinformationen wie Passwörter sicher in benutzerdefinierten Komponenten auf Ihren Greengrass-Kerngeräten zu verwenden. Weitere Informationen zu Secrets Manager finden Sie unter Was ist AWS Secrets Manager? im AWS Secrets Manager Benutzerhandbuch.

Um auf die Geheimnisse dieser Komponente in Ihren benutzerdefinierten Greengrass-Komponenten zuzugreifen, verwenden Sie die GetSecretValue-Operation in der AWS IoT Device SDK. Weitere Informationen finden Sie unter Verwenden Sie den AWS IoT Device SDK , um mit dem Greengrass-

Kern und anderen Komponenten zu kommunizieren und AWS IoT Core und Abrufen von Secret-Werten.

Diese Komponente verschlüsselt Geheimnisse auf dem Kerngerät, um Ihre Anmeldeinformationen und Passwörter zu schützen, bis Sie sie verwenden müssen. Sie verwendet den privaten Schlüssel des Kerngeräts, um Geheimnisse zu verschlüsseln und zu entschlüsseln.

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.1.x
- 2.0.x

Typ

Diese Komponente ist eine Plugin-Komponente (`aws.greengrass.plugin`). Der Greengrass-Kern führt diese Komponente in derselben Java Virtual Machine (JVM) wie der Nucleus aus. Der Nucleus wird neu gestartet, wenn Sie die Version dieser Komponente auf dem Core-Gerät ändern.

Diese Komponente verwendet dieselbe Protokolldatei wie der Greengrass-Kern. Weitere Informationen finden Sie unter Überwachen von AWS IoT Greengrass Protokollen.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf Kerngeräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Die Greengrass-Geräterolle muss die `secretsmanager:GetSecretValue` Aktion zulassen, wie in der folgenden Beispiel-IAM-Richtlinie gezeigt.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:secretsmanager:region:123456789012:secret:MySecret"
      ]
    }
  ]
}
```

> **Note**
>
> Wenn Sie einen vom Kunden verwalteten AWS Key Management Service Schlüssel zum Verschlüsseln von Geheimnissen verwenden, muss die Geräterolle die Aktion ebenfalls zulassen. `kms:Decrypt`

Weitere Informationen zu den IAM-Richtlinien für Secrets Manager finden Sie im folgenden AWS Secrets Manager Benutzerhandbuch:

  - Authentifizierung und Zugriffskontrolle für AWS Secrets Manager
  - Aktionen, Ressourcen und Kontextschlüssel, die Sie in einer IAM-Richtlinie oder Geheimrichtlinie verwenden können AWS Secrets Manager
- Benutzerdefinierte Komponenten müssen eine Autorisierungsrichtlinie definieren, die `aws.greengrass#GetSecretValue` den Zugriff auf Geheimnisse ermöglicht, die Sie mit dieser Komponente speichern. In dieser Autorisierungsrichtlinie können Sie den Zugriff von Komponenten auf bestimmte Geheimnisse einschränken. Weitere Informationen finden Sie unter Secret Manager IPC-Autorisierung.
- (Optional) Wenn Sie den privaten Schlüssel und das Zertifikat des Core-Geräts in einem Hardware-Sicherheitsmodul (HSM) speichern, muss das HSM RSA-Schlüssel unterstützen, der private Schlüssel muss über die `unwrap` entsprechende Berechtigung verfügen und der öffentliche Schlüssel muss über die entsprechende Berechtigung verfügen. `wrap`

Endpunkte und Ports

Diese Komponente muss in der Lage sein, ausgehende Anfragen an die folgenden Endpunkte und Ports auszuführen, zusätzlich zu den Endpunkten und Ports, die für den Basisbetrieb erforderlich sind. Weitere Informationen finden Sie unter Zulassen von Gerätedatenverkehr über einen Proxy oder eine Firewall.

Endpunkt	Port	Erforderlich	Beschreibung
`secretsmanager.` *`region`*`.amazonaws.com`	443	Ja	Laden Sie Geheimnisse auf das Kerngerät herunter.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und all ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitstellen zu können. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen

dieser Komponente sowie die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können die Abhängigkeiten für jede Version der Komponente auch in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste der Abhängigkeiten.

2.1.7 – 2.1.8

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.1.7 und 2.1.8 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2,5,0 <2,13,0	Weich

2.1.6

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.5.0 <2.12.0	Weich

2.1.5

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2,5,0 <2,11,0	Weich

2.1.4

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.5.0 <2.10.0	Weich

2.1.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2,5,0 <2,9,0	Weich

2.1.2

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2,5,0 <2,8,0	Weich

2.1.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2,5,0 <2,7,0	Weich

2.1.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2,5,0 <2,6,0	Weich

2.0.9

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.9 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.5.0	Weich

2.0.8

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.8 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.4.0	Weich

2.0.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.3.0	Weich

2.0.6

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.2.0	Weich

2.0.4 and 2.0.5

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.0.4 und 2.0.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.3 <2.1.0	Weich

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zu den Komponentenrezepten.

Konfiguration

Diese Komponente stellt die folgenden Konfigurationsparameter bereit, die Sie bei der Bereitstellung der Komponente anpassen können.

`cloudSecrets`

Eine Liste von Secrets Manager Manager-Geheimnissen, die auf dem Kerngerät bereitgestellt werden sollen. Sie können Labels angeben, um zu definieren, welche Versionen der einzelnen Secrets bereitgestellt werden sollen. Wenn Sie keine Version angeben, stellt diese Komponente die Version mit dem angehängten Staging-Label `AWSCURRENT` bereit. Weitere Informationen finden Sie unter Staging-Labels im AWS Secrets Manager Benutzerhandbuch.

Die Secret Manager-Komponente speichert Geheimnisse lokal zwischen. Wenn sich der geheime Wert in Secrets Manager ändert, ruft diese Komponente den neuen Wert nicht automatisch ab. Um die lokale Kopie zu aktualisieren, geben Sie dem Geheimnis eine neue Bezeichnung und konfigurieren Sie diese Komponente so, dass sie das durch das neue Label identifizierte Geheimnis abruft.

Jedes Objekt enthält die folgenden Informationen:

`arn`

Der ARN des Secrets, das bereitgestellt werden soll. Der ARN des Geheimnisses kann entweder ein vollständiger ARN oder ein teilweiser ARN sein. Wir empfehlen, dass Sie einen vollständigen ARN anstelle eines teilweisen ARN angeben. Weitere Informationen finden Sie unter Finden eines Geheimnisses aus einem partiellen ARN. Im Folgenden finden Sie ein Beispiel für einen vollständigen ARN und einen teilweisen ARN:

- Vollständiger ARN: `arn:aws:secretsmanager:us-east-2:111122223333:secret:`*`SecretName`*`-abcdef`
- Teilweiser ARN: `arn:aws:secretsmanager:us-east-2:111122223333:secret:`*`SecretName`*

`labels`

(Optional) Eine Liste von Labels zur Identifizierung der Versionen des Secrets, die auf dem Core-Gerät bereitgestellt werden sollen.

Jedes Label muss eine Zeichenfolge sein.

Example Beispiel: Aktualisierung der Zusammenführung von Konfigurationen

```
{
  "cloudSecrets": [
    {
      "arn": "arn:aws:secretsmanager:us-west-2:123456789012:secret:MyGreengrassSecret-
abcdef"
    }
  ]
}
```

Lokale Protokolldatei

Diese Komponente verwendet dieselbe Protokolldatei wie die Greengrass Nucleus-Komponente.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

Um die Protokolle dieser Komponente einzusehen

- Führen Sie den folgenden Befehl auf dem Kerngerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

  Linux

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

  Windows (PowerShell)

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

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
2.1.8	Fehlerkorrekturen und Verbesserungen Behebt ein Problem, bei dem der Secret Manager einen teilweisen ARN nicht akzeptiert.
2.1.7	Die Version wurde für die Version 2.12.0 von Greengrass Nucleus aktualisiert.
2.1.6	Die Version wurde für die Version 2.11.0 von Greengrass Nucleus aktualisiert.
2.1.5	Die Version wurde für die Version 2.10.0 von Greengrass Nucleus aktualisiert.
2.1.4	Fehlerkorrekturen und Verbesserungen Behebt ein Problem, bei dem zwischengespeicherte Geheimnisse entfernt wurden, wenn Secret Manager bereitgestellt und Greengrass Nucleus neu gestartet wurde. Die Version wurde für die Version 2.9.0 von Greengrass Nucleus aktualisiert.
2.1.3	Die Version wurde für die Version 2.8.0 von Greengrass Nucleus aktualisiert.
2.1.2	Die Version wurde für die Version 2.7.0 von Greengrass Nucleus aktualisiert.
2.1.1	Die Version wurde für die Version 2.6.0 von Greengrass Nucleus aktualisiert.
2.1.0	Neue Features • Fügt Unterstützung für die Integration von Hardwaresicherheit hinzu. Die Secret Manager-Komponente kann Geheimnisse mithilfe eines privaten Schlüssels verschlüsseln und entschlüsseln, den Sie in einem Hardware-Sicherheitsmodul (HSM) speichern. Weitere Informationen finden Sie unter Integration von Hardware-Sicherheit.

Version	Änderungen
	Fehlerkorrekturen und Verbesserungen • Die Version wurde für die Version 2.5.0 von Greengrass Nucleus aktualisiert.
2.0.9	Die Version wurde für die Version 2.4.0 von Greengrass Nucleus aktualisiert.
2.0.8	Die Version wurde für die Version 2.3.0 von Greengrass Nucleus aktualisiert.
2.0.7	Die Version wurde für die Version 2.2.0 von Greengrass Nucleus aktualisiert.
2.0.6	Die Version wurde für die Version 2.1.0 von Greengrass Nucleus aktualisiert.
2.0.5	Verbesserungen • Unterstützung für AWS chinesische Regionen und AWS GovCloud (US) Regionen hinzugefügt.
2.0.4	Erste Version

Sicheres Tunneling

Mit der `aws.greengrass.SecureTunneling` Komponente können Sie eine sichere bidirektionale Kommunikation mit einem Greengrass-Core-Gerät aufbauen, das sich hinter eingeschränkten Firewalls befindet.

Stellen Sie sich zum Beispiel vor, Sie haben ein Greengrass-Core-Gerät hinter einer Firewall, die alle eingehenden Verbindungen verbietet. Secure Tunneling verwendet MQTT, um ein Zugriffstoken auf das Gerät zu übertragen, und stellt dann über die WebSockets Firewall eine SSH-Verbindung zum Gerät her. Mit diesem AWS IoT verwalteten Tunnel können Sie die für Ihr Gerät benötigte SSH-Verbindung öffnen. Weitere Informationen zur Verwendung von AWS IoT Secure Tunneling für die Verbindung zu Remote-Geräten finden Sie unter AWS IoT Secure Tunneling im Entwicklerhandbuch.AWS IoT

Diese Komponente abonniert den AWS IoT Core MQTT-Nachrichtenbroker zu diesem `$aws/things/`*`greengrass-core-device`*`/tunnels/notify` Thema, um Benachrichtigungen über sicheres Tunneling zu erhalten.

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 1.0.x

Typ

Diese Komponente ist eine generische Komponente (`aws.greengrass.generic`). Der Greengrass-Kern führt die Lebenszyklusskripte der Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann nur auf Linux-Core-Geräten installiert werden.

Architekturen:

- Armv71
- Armv8 (AArch64)
- x86_64

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Für die Secure Tunneling-Komponente stehen mindestens 32 MB Festplattenspeicher zur Verfügung. Diese Anforderung umfasst nicht die Greengrass-Kernsoftware oder andere Komponenten, die auf demselben Gerät ausgeführt werden.
- Für die Secure Tunneling-Komponente sind mindestens 16 MB RAM verfügbar. Diese Anforderung umfasst nicht die Greengrass-Kernsoftware oder andere Komponenten, die auf demselben Gerät ausgeführt werden. Weitere Informationen finden Sie unter Steuern Sie die Speicherzuweisung mit JVM-Optionen.
- GNU C Library (Glibc) Version 2.25 oder höher mit einem Linux-Kernel von 3.2 oder höher sind für die Secure Tunneling-Komponente Version 1.0.12 und höher erforderlich. Versionen des Betriebssystems und der Bibliotheken, deren Haltbarkeitsdatum für den langfristigen Support abgelaufen ist, werden nicht unterstützt. Sie sollten ein Betriebssystem und Bibliotheken mit langfristigem Support verwenden.
- Sowohl das Betriebssystem als auch die Java-Runtime müssen als 64-Bit-Version installiert sein.
- Python 3.5 oder höher wurde auf dem Greengrass-Core-Gerät installiert und der Umgebungsvariablen PATH hinzugefügt.
- `libcrypto.so.1.1`auf dem Greengrass-Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.
- Öffnen Sie ausgehenden Verkehr auf Port 443 auf dem Greengrass-Core-Gerät.
- Aktivieren Sie die Unterstützung für den Kommunikationsdienst, den Sie für die Kommunikation mit dem Greengrass Core-Gerät verwenden möchten. Um beispielsweise eine SSH-Verbindung zu dem Gerät herzustellen, müssen Sie SSH auf diesem Gerät aktivieren.

Endpunkte und Ports

Diese Komponente muss in der Lage sein, ausgehende Anfragen an die folgenden Endpunkte und Ports auszuführen, zusätzlich zu den Endpunkten und Ports, die für den Basisbetrieb erforderlich sind. Weitere Informationen finden Sie unter Zulassen von Gerätedatenverkehr über einen Proxy oder eine Firewall.

Endpunkt	Port	Erforderlich	Beschreibung
`data.tunneling.iot.region.amazonaws.com`	443	Ja	Richten Sie sichere Tunnel ein.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt sie AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und all ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitstellen zu können. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente sowie die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste der Abhängigkeiten.

1.0.19

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.0.19 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <3.0.0	Weich

1.0.18

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.0.18 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.13.0	Weich

1.0.16 – 1.0.17

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 1.0.16 bis 1.0.17 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.12.0	Weich

1.0.14 – 1.0.15

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 1.0.14 bis 1.0.15 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.11.0	Weich

1.0.11 – 1.0.13

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 1.0.11 — 1.0.13 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.10.0	Weich

1.0.10

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.0.10 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.9.0	Weich

1.0.9

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.0.9 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.8.0	Weich

1.0.8

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.0.8 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.7.0	Weich

1.0.5 - 1.0.7

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 1.0.5 bis 1.0.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.6.0	Weich

1.0.4

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.0.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.5.0	Weich

1.0.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.0.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.4.0	Weich

1.0.2

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.0.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.3.0	Weich

1.0.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.0.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.2.0	Weich

1.0.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.0.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.3 <2.1.0	Weich

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zu den Komponentenrezepten.

Konfiguration

Diese Komponente stellt die folgenden Konfigurationsparameter bereit, die Sie bei der Bereitstellung der Komponente anpassen können.

`OS_DIST_INFO`

(Optional) Das Betriebssystem Ihres Kerngeräts. Standardmäßig versucht die Komponente, das auf Ihrem Kerngerät ausgeführte Betriebssystem automatisch zu identifizieren. Wenn die Komponente nicht mit dem Standardwert startet, verwenden Sie diesen Wert, um das Betriebssystem anzugeben. Eine Liste der unterstützten Betriebssysteme für diese Komponente finden Sie unterAnforderungen an Speichergeräte.

Dieser Wert kann einer der folgenden sein:`auto`,`ubuntu`,`amzn2`,`raspberrypi`.

Standard: `auto`

`accessControl`

(Optional) Das Objekt, das die Autorisierungsrichtlinie enthält, die es der Komponente ermöglicht, das Thema Secure Tunneling-Benachrichtigungen zu abonnieren.

> **Note**
>
> Ändern Sie diesen Konfigurationsparameter nicht, wenn Ihre Bereitstellung auf eine Dinggruppe abzielt. Wenn Ihre Bereitstellung auf ein einzelnes Kerngerät abzielt und Sie dessen Abonnement auf das Thema des Geräts beschränken möchten, geben Sie den Ding-Namen des Kerngeräts an. Ersetzen Sie in dem `resources` Wert in der Autorisierungsrichtlinie des Geräts den Platzhalter für das MQTT-Thema durch den Ding-Namen des Geräts.

```
{
  "aws.greengrass.ipc.mqttproxy": {
    "aws.iot.SecureTunneling:mqttproxy:1": {
      "policyDescription": "Access to tunnel notification pubsub topic",
      "operations": [
        "aws.greengrass#SubscribeToIoTCore"
      ],
      "resources": [
        "$aws/things/+/tunnels/notify"
      ]
    }
  }
}
```

Example Beispiel: Aktualisierung der Zusammenführung von Konfigurationen

Die folgende Beispielkonfiguration gibt an, **`MyGreengrassCore`** dass diese Komponente sichere Tunnel auf einem Kerngerät mit dem Namen Ubuntu öffnen kann.

```
{
  "OS_DIST_INFO": "ubuntu",
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "aws.iot.SecureTunneling:mqttproxy:1": {
        "policyDescription": "Access to tunnel notification pubsub topic",
        "operations": [
          "aws.greengrass#SubscribeToIoTCore"
        ],
        "resources": [
          "$aws/things/MyGreengrassCore/tunnels/notify"
        ]
      }
    }
  }
}
```

Lokale Protokolldatei

Diese Komponente verwendet die folgende Protokolldatei.

```
/greengrass/v2/logs/aws.greengrass.SecureTunneling.log
```

Um die Protokolle dieser Komponente einzusehen

- Führen Sie den folgenden Befehl auf dem Kerngerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. */greengrass/v2*Ersetzen Sie durch den Pfad zum AWS IoT Greengrass Stammordner.

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

Lizenzen

Diese Komponente umfasst die folgende Software/Lizenzierung von Drittanbietern:

- AWS IoT Geräteclient /Apache License 2.0
- AWS IoT Device SDK for Java/Apache-Lizenz 2.0
- gson /Apache Lizenz 2.0
- log4j /Apache Lizenz 2.0
- slf4j /Apache Lizenz 2.0

Verwendung

Gehen Sie wie folgt vor, um die Secure Tunneling-Komponente auf Ihrem Gerät zu verwenden:

1. Stellen Sie die Secure Tunneling-Komponente auf Ihrem Gerät bereit.
2. Öffnen Sie die AWS IoT -Konsole. Wählen Sie im linken Menü die Option Remote-Aktionen und dann Sichere Tunnel aus.
3. Erstellen Sie einen Tunnel zu Ihrem Greengrass-Gerät.
4. Laden Sie das Quellzugriffstoken herunter.
5. Verwenden Sie den lokalen Proxy mit dem Quellzugriffstoken, um eine Verbindung zu Ihrem Ziel herzustellen. Weitere Informationen finden Sie unter So verwenden Sie den lokalen Proxy im AWS IoT Entwicklerhandbuch.

Weitere Informationen finden Sie auch unter

- AWS IoTsicheres Tunneling im Entwicklerhandbuch AWS IoT
- Wie benutzt man den lokalen Proxy im Developer Guide AWS IoT

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
1.0.19	Fehlerkorrekturen und Verbesserungen • Führt ein Upgrade des zugrunde liegenden AWS IoT Geräteclients, der von der Komponente aufgerufen wird, von Version 1.8.0 auf Version 1.9.0 durch.

Version	Änderungen
	• Erhöht das Limit für gleichzeitige Tunnel auf Komponentenebene auf 20 Tunnel. • Erhöht das standardmäßige AWS IoT Greengrass Core-IPC-Timeout von 3 Sekunden auf 10 Sekunden. **Warning** Wenn Sie den lokalen Secure Tunneling-Proxy als Tunnelquellclient verwenden, aktualisieren Sie Ihre Komponente erst auf diese Version, wenn Sie auch den lokalen Proxy auf Version 3.1.1 oder höher aktualisiert haben.
1.0.18	Die Version wurde für die Version 2.12.0 von Greengrass Nucleus aktualisiert.
1.0.17	Fehlerkorrekturen und Verbesserungen • Behebt das Problem mit der Thread-Bereinigung, das Benutzer daran hinderte, Tunnel zu erstellen. Diese Komponente bereinigt nun einen Thread, entweder sobald er das CloseTunnel Signal empfängt oder wenn der Tunnel nach 12 Stunden abgelaufen ist.
1.0.16	Die Version wurde für die Version 2.11.0 von Greengrass Nucleus aktualisiert.
1.0.15	Fehlerkorrekturen und Verbesserungen • Behebt ein Startproblem für Benutzer, die kein Home-Verzeichnis auf dem Gerät haben. Die Secure Tunneling-Komponente startet jetzt, ohne ein Verzeichnis für Shadow-Dokumente zu erstellen.
1.0.14	Die Version wurde für die Version 2.10.0 von Greengrass Nucleus aktualisiert.
1.0.13	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem ein verwaister Client-Prozess verhindert, dass mehr als ein Tunnel auf das Gerät abzielt.

Version	Änderungen
1.0.12	Fehlerkorrekturen und Verbesserungen • Fügt Unterstützung für x86_64 (AMD64) und ARMv8 (Aarch64) hinzu, wenn sie unter Raspberry Pi OS ausgeführt werden.
1.0.11	Die Version wurde für die Version 2.9.0 von Greengrass Nucleus aktualisiert.
1.0.10	Die Version wurde für die Version 2.8.0 von Greengrass Nucleus aktualisiert.
1.0.9	Die Version wurde für die Version 2.7.0 von Greengrass Nucleus aktualisiert.
1.0.8	Die Version wurde für die Version 2.6.0 von Greengrass Nucleus aktualisiert.
1.0.7	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem die Verbindung der Komponente unterbroc hen wird, wenn Sie große Dateien über SCP übertragen.
1.0.6	Diese Version enthält Fehlerkorrekturen.
1.0.5	Die Version wurde für die Version 2.5.0 von Greengrass Nucleus aktualisiert.
1.0.4	Die Version wurde für die Version 2.4.0 von Greengrass Nucleus aktualisiert.
1.0.3	Die Version wurde für die Version 2.3.0 von Greengrass Nucleus aktualisiert.
1.0.2	Die Version wurde für die Version 2.2.0 von Greengrass Nucleus aktualisiert.
1.0.1	Die Version wurde für die Version 2.1.0 von Greengrass Nucleus aktualisiert.
1.0.0	Erste Version

Schattenmanager

Die Shadow-Manager-Komponente (`aws.greengrass.ShadowManager`) aktiviert den lokalen Shadow-Dienst auf Ihrem Kerngerät. Der lokale Shadow-Dienst ermöglicht es Komponenten, die Kommunikation zwischen Prozessen zu nutzen, um mit lokalen Shadows zu interagieren. Die Shadow-Manager-Komponente verwaltet die Speicherung lokaler Shadow-Dokumente und kümmert

sich auch um die Synchronisation lokaler Shadow-Zustände mit dem AWS IoT Device Shadow-Dienst.

Weitere Informationen darüber, wie Greengrass-Core-Geräte mit Schatten interagieren können, finden Sie unterInteragieren mit Geräteschatten.

Themen

- Versionen
- Typ
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Konfiguration
- Lokale Protokolldatei
- Änderungsprotokoll

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Typ

Diese Komponente ist eine Plugin-Komponente (`aws.greengrass.plugin`). Der Greengrass-Kern führt diese Komponente in derselben Java Virtual Machine (JVM) wie der Nucleus aus. Der Nucleus wird neu gestartet, wenn Sie die Version dieser Komponente auf dem Kerngerät ändern.

Diese Komponente verwendet dieselbe Protokolldatei wie der Greengrass-Kern. Weitere Informationen finden Sie unter Überwachen von AWS IoT Greengrass Protokollen.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf Kerngeräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- (Optional) Um Shadows mit dem AWS IoT Device Shadow-Dienst zu synchronisieren, muss die AWS IoT Richtlinie des Greengrass-Core-Geräts die folgenden AWS IoT Core Shadow-Richtlinienaktionen zulassen:
  - `iot:GetThingShadow`
  - `iot:UpdateThingShadow`
  - `iot:DeleteThingShadow`

  Weitere Informationen zu diesen AWS IoT Core Richtlinien finden Sie unter AWS IoT Core Richtlinienaktionen im AWS IoT Entwicklerhandbuch.

  Weitere Informationen zur AWS IoT Mindestrichtlinie finden Sie unter Minimale AWS IoT Richtlinie für -AWS IoT Greengrass V2Core-Geräte
- Die Shadow Manager-Komponente wird für die Ausführung in einer VPC unterstützt.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt sie AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und all ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitstellen zu können. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente sowie die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste der Abhängigkeiten.

2.3.5 – 2.3.8

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.3.5 bis 2.3.8 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2,5,0 <2,13,0	Weich

2.3.3 and 2.3.4

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.3.3 und 2.3.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.5.0 <2.12.0	Weich

2.3.2

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.3.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2,5,0 <2,11,0	Weich

2.3.0 and 2.3.1

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.3.0 und 2.3.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.5.0 <2.10.0	Weich

2.2.3 and 2.2.4

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.2.3 und 2.2.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.2.0 <3.0.0	Weich

2.2.2

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.2.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.2.0 <2.9.0	Weich

2.2.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.2.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.2.0 <2.8.0	Weich

2.1.1 and 2.2.0

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.1.1 und 2.2.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.2.0 <2.7.0	Weich

2.0.5 - 2.1.0

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.0.5 bis 2.1.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.2.0 <2.6.0	Weich

2.0.3 and 2.0.4

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.0.3 und 2.0.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.2.0 <2.5.0	Weich

2.0.1 and 2.0.2

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.0.1 und 2.0.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.2.0 <2.4.0	Weich

2.0.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.2.0 <2.3.0	Weich

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zu den Komponentenrezepten.

Konfiguration

Diese Komponente stellt die folgenden Konfigurationsparameter bereit, die Sie bei der Bereitstellung der Komponente anpassen können.

2.3.x

`strategy`

(Optional) Die Strategie, mit der diese Komponente Schatten zwischen AWS IoT Core und dem Kerngerät synchronisiert.

Dieses Objekt enthält die folgenden Informationen.

`type`

(Optional) Die Art der Strategie, die diese Komponente verwendet, um Schatten zwischen AWS IoT Core und dem Kerngerät zu synchronisieren. Wählen Sie aus den folgenden Optionen aus:

- `realTime`— Synchronisiert Schatten bei AWS IoT Core jedem Shadow-Update.
- `periodic`— Synchronisieren Sie Schatten in einem regelmäßigen Intervall, das Sie mit dem `delay` Konfigurationsparameter angeben. AWS IoT Core

Standard: `realTime`

`delay`

(Optional) Das Intervall in Sekunden, mit dem diese Komponente Schatten synchronisiert AWS IoT Core, wenn Sie die `periodic` Synchronisierungsstrategie angeben.

> **Note**
>
> Dieser Parameter ist erforderlich, wenn Sie die `periodic` Synchronisierungsstrategie angeben.

`synchronize`

(Optional) Die Synchronisierungseinstellungen, die bestimmen, wie Schatten mit dem AWS Cloud synchronisiert werden.

> **Note**
>
> Sie müssen ein Konfigurationsupdate mit dieser Eigenschaft erstellen, um Schatten mit dem AWS Cloud zu synchronisieren.

Dieses Objekt enthält die folgenden Informationen.

`coreThing`

(Optional) Das Core-Gerät wird zur Synchronisation gescrollt. Dieses Objekt enthält die folgenden Informationen.

`classic`

(Optional) Standardmäßig synchronisiert der Shadow-Manager den lokalen Status des klassischen Shadows für Ihr Kerngerät mit dem AWS Cloud. Wenn Sie den klassischen Geräteshadow nicht synchronisieren möchten, setzen Sie ihn auf`false`.

Standard: `true`

`namedShadows`

(Optional) Die Liste der benannten Core-Device-Shadows, die synchronisiert werden sollen. Sie müssen die genauen Namen der Schatten angeben.

> **Warning**
>
> Der AWS IoT Greengrass Dienst verwendet den `AWSManagedGreengrassV2Deployment` benannten Shadow, um Bereitstellungen zu verwalten, die auf einzelne Kerngeräte abzielen. Dieser benannte Shadow ist für die Verwendung durch den AWS IoT Greengrass Dienst reserviert. Aktualisieren oder löschen Sie diesen benannten Schatten nicht.

`shadowDocumentsMap`

(Optional) Die zusätzlichen Geräteschatten, die synchronisiert werden sollen. Die Verwendung dieses Konfigurationsparameters erleichtert die Angabe von Schattendokumenten. Es wird empfohlen, diesen Parameter anstelle des `shadowDocuments` Objekts zu verwenden.

> **Note**
>
> Wenn Sie ein `shadowDocumentsMap` Objekt angeben, dürfen Sie kein `shadowDocuments` Objekt angeben.

Jedes Objekt enthält die folgenden Informationen:

`thingName`

Die Shadow-Konfiguration für den *`thingName`* für diese Shadow-Konfiguration.

`classic`

(Optional) Wenn Sie den klassischen Geräteshadow für das `thingName` Gerät nicht synchronisieren möchten, setzen Sie ihn auf`false`.

`namedShadows`

Die Liste der benannten Schatten, die Sie synchronisieren möchten. Sie müssen die genauen Namen der Schatten angeben.

`shadowDocuments`

(Optional) Die Liste der zusätzlichen Geräteschatten, die synchronisiert werden sollen. Wir empfehlen, stattdessen den `shadowDocumentsMap` Parameter zu verwenden.

> **Note**
>
> Wenn Sie ein `shadowDocuments` Objekt angeben, dürfen Sie kein `shadowDocumentsMap` Objekt angeben.

Jedes Objekt in dieser Liste enthält die folgenden Informationen.

`thingName`

Der Dingname des Geräts, für das Schatten synchronisiert werden sollen.

`classic`

(Optional) Wenn Sie den klassischen Geräteshadow für das `thingName` Gerät nicht synchronisieren möchten, setzen Sie ihn auf`false`.

Standard: `true`

`namedShadows`

(Optional) Die Liste der benannten Geräteschatten, die Sie synchronisieren möchten. Sie müssen die genauen Namen der Schatten angeben.

`direction`

(Optional) Die Richtung, in der Schatten zwischen dem lokalen Shadow-Dienst und dem synchronisiert AWS Cloud werden sollen. Sie können diese Option konfigurieren, um die Bandbreite und die Verbindungen zum zu reduzieren AWS Cloud. Wählen Sie aus den folgenden Optionen aus:

- `betweenDeviceAndCloud`— Synchronisiert Shadows zwischen dem lokalen Shadow-Dienst und dem AWS Cloud.
- `deviceToCloud`— Sendet Shadow-Updates vom lokalen Shadow-Dienst an den AWS Cloud und ignoriert Shadow-Updates vom AWS Cloud.
- `cloudToDevice`— Empfangen Sie Shadow-Updates vom AWS Cloud und senden Sie keine Shadow-Updates vom lokalen Shadow-Dienst an den AWS Cloud.

Standard: `BETWEEN_DEVICE_AND_CLOUD`

`rateLimits`

(Optional) Die Einstellungen, die die Ratenlimits für Shadow-Service-Anfragen festlegen.

Dieses Objekt enthält die folgenden Informationen.

`maxOutboundSyncUpdatesPerSecond`

(Optional) Die maximale Anzahl von Synchronisierungsanfragen pro Sekunde, die das Gerät überträgt.

Standard: 100 Anfragen/Sekunde

`maxTotalLocalRequestsRate`

(Optional) Die maximale Anzahl lokaler IPC-Anfragen pro Sekunde, die an das Kerngerät gesendet werden.

Standard: 200 Anfragen/Sekunde

`maxLocalRequestsPerSecondPerThing`

(Optional) Die maximale Anzahl lokaler IPC-Anfragen pro Sekunde, die für jedes verbundene IoT-Ding gesendet werden.

Standard: 20 Anfragen/Sekunde für jedes Ding

> **Note**
>
> Diese Parameter für die Ratenbegrenzung definieren die maximale Anzahl von Anfragen pro Sekunde für den lokalen Shadow-Dienst. Die maximale Anzahl von Anfragen pro Sekunde für den AWS IoT Device Shadow-Dienst hängt von Ihrem ab AWS-Region. Weitere Informationen finden Sie in den Grenzwerten für die AWS IoT Device Shadow Service API in der Allgemeine Amazon Web Services-Referenz.

`shadowDocumentSizeLimitBytes`

(Optional) Die maximal zulässige Größe jedes JSON-Statusdokuments für lokale Schatten.

Wenn Sie diesen Wert erhöhen, müssen Sie auch das Ressourcenlimit für das JSON-Statusdokument für Wolkenschatten erhöhen. Weitere Informationen finden Sie in den Grenzwerten für die AWS IoT Device Shadow Service API in der Allgemeine Amazon Web Services-Referenz.

Standard: 8192 Byte

Maximum: 30720 Byte

Example Beispiel: Update zur Zusammenführung von Konfigurationen

Das folgende Beispiel zeigt ein Beispiel für ein Update zur Zusammenführung von Konfigurationen mit allen verfügbaren Konfigurationsparametern für die Shadow Manager-Komponente.

```
{
   "strategy":{
      "type":"periodic",
      "delay":300
   },
   "synchronize":{
      "shadowDocumentsMap":{
         "MyDevice1":{
            "classic":false,
            "namedShadows":[
               "MyShadowA",
               "MyShadowB"
            ]
         },
         "MyDevice2":{
            "classic":true,
            "namedShadows":[]
         }
      },
      "direction":"betweenDeviceAndCloud"
   },
   "rateLimits":{
      "maxOutboundSyncUpdatesPerSecond":100,
      "maxTotalLocalRequestsRate":200,
      "maxLocalRequestsPerSecondPerThing":20
   },
   "shadowDocumentSizeLimitBytes":8192
}
```

2.2.x

`strategy`

(Optional) Die Strategie, mit der diese Komponente Schatten zwischen AWS IoT Core und dem Kerngerät synchronisiert.

Dieses Objekt enthält die folgenden Informationen.

`type`

(Optional) Die Art der Strategie, die diese Komponente verwendet, um Schatten zwischen AWS IoT Core und dem Kerngerät zu synchronisieren. Wählen Sie aus den folgenden Optionen aus:

- `realTime`— Synchronisiert Schatten bei AWS IoT Core jedem Shadow-Update.
- `periodic`— Synchronisieren Sie Schatten in einem regelmäßigen Intervall, das Sie mit dem `delay` Konfigurationsparameter angeben. AWS IoT Core

Standard: `realTime`

`delay`

(Optional) Das Intervall in Sekunden, mit dem diese Komponente Schatten synchronisiert AWS IoT Core, wenn Sie die `periodic` Synchronisierungsstrategie angeben.

> **Note**
>
> Dieser Parameter ist erforderlich, wenn Sie die `periodic` Synchronisierungsstrategie angeben.

`synchronize`

(Optional) Die Synchronisierungseinstellungen, die bestimmen, wie Schatten mit dem AWS Cloud synchronisiert werden.

> **Note**
>
> Sie müssen ein Konfigurationsupdate mit dieser Eigenschaft erstellen, um Schatten mit dem AWS Cloud zu synchronisieren.

Dieses Objekt enthält die folgenden Informationen.

`coreThing`

(Optional) Das Core-Gerät wird zur Synchronisation gescrollt. Dieses Objekt enthält die folgenden Informationen.

`classic`

(Optional) Standardmäßig synchronisiert der Shadow-Manager den lokalen Status des klassischen Shadows für Ihr Kerngerät mit dem AWS Cloud. Wenn Sie den klassischen Geräteshadow nicht synchronisieren möchten, setzen Sie ihn auf`false`.

Standard: `true`

`namedShadows`

(Optional) Die Liste der benannten Core-Device-Shadows, die synchronisiert werden sollen. Sie müssen die genauen Namen der Schatten angeben.

> **Warning**
>
> Der AWS IoT Greengrass Dienst verwendet den `AWSManagedGreengrassV2Deployment` benannten Shadow, um Bereitstellungen zu verwalten, die auf einzelne Kerngeräte abzielen. Dieser benannte Shadow ist für die Verwendung durch den AWS IoT Greengrass Dienst reserviert. Aktualisieren oder löschen Sie diesen benannten Schatten nicht.

`shadowDocumentsMap`

(Optional) Die zusätzlichen Geräteschatten, die synchronisiert werden sollen. Die Verwendung dieses Konfigurationsparameters erleichtert die Angabe von Schattendokumenten. Es wird empfohlen, diesen Parameter anstelle des `shadowDocuments` Objekts zu verwenden.

> **Note**
>
> Wenn Sie ein `shadowDocumentsMap` Objekt angeben, dürfen Sie kein `shadowDocuments` Objekt angeben.

Jedes Objekt enthält die folgenden Informationen:

`thingName`

Die Shadow-Konfiguration für den *`thingName`* für diese Shadow-Konfiguration.

`classic`

(Optional) Wenn Sie den klassischen Geräteshadow für das `thingName` Gerät nicht synchronisieren möchten, setzen Sie ihn auf`false`.

`namedShadows`

Die Liste der benannten Schatten, die Sie synchronisieren möchten. Sie müssen die genauen Namen der Schatten angeben.

`shadowDocuments`

(Optional) Die Liste der zusätzlichen Geräteschatten, die synchronisiert werden sollen. Wir empfehlen, stattdessen den `shadowDocumentsMap` Parameter zu verwenden.

> **Note**
>
> Wenn Sie ein `shadowDocuments` Objekt angeben, dürfen Sie kein `shadowDocumentsMap` Objekt angeben.

Jedes Objekt in dieser Liste enthält die folgenden Informationen.

`thingName`

Der Dingname des Geräts, für das Schatten synchronisiert werden sollen.

`classic`

(Optional) Wenn Sie den klassischen Geräteshadow für das `thingName` Gerät nicht synchronisieren möchten, setzen Sie ihn auf`false`.

Standard: `true`

`namedShadows`

(Optional) Die Liste der benannten Geräteschatten, die Sie synchronisieren möchten. Sie müssen die genauen Namen der Schatten angeben.

`direction`

(Optional) Die Richtung, in der Schatten zwischen dem lokalen Shadow-Dienst und dem synchronisiert AWS Cloud werden sollen. Sie können diese Option konfigurieren, um die Bandbreite und die Verbindungen zum zu reduzieren AWS Cloud. Wählen Sie aus den folgenden Optionen aus:

- `betweenDeviceAndCloud`— Synchronisiert Shadows zwischen dem lokalen Shadow-Dienst und dem AWS Cloud.
- `deviceToCloud`— Sendet Shadow-Updates vom lokalen Shadow-Dienst an den AWS Cloud und ignoriert Shadow-Updates vom AWS Cloud.

- `cloudToDevice`— Empfangen Sie Shadow-Updates vom AWS Cloud und senden Sie keine Shadow-Updates vom lokalen Shadow-Dienst an den AWS Cloud.

Standard: `BETWEEN_DEVICE_AND_CLOUD`

`rateLimits`

(Optional) Die Einstellungen, die die Ratenlimits für Shadow-Service-Anfragen festlegen.

Dieses Objekt enthält die folgenden Informationen.

`maxOutboundSyncUpdatesPerSecond`

(Optional) Die maximale Anzahl von Synchronisierungsanfragen pro Sekunde, die das Gerät überträgt.

Standard: 100 Anfragen/Sekunde

`maxTotalLocalRequestsRate`

(Optional) Die maximale Anzahl lokaler IPC-Anfragen pro Sekunde, die an das Kerngerät gesendet werden.

Standard: 200 Anfragen/Sekunde

`maxLocalRequestsPerSecondPerThing`

(Optional) Die maximale Anzahl lokaler IPC-Anfragen pro Sekunde, die für jedes verbundene IoT-Ding gesendet werden.

Standard: 20 Anfragen/Sekunde für jedes Ding

> **Note**
>
> Diese Parameter für die Ratenbegrenzung definieren die maximale Anzahl von Anfragen pro Sekunde für den lokalen Shadow-Dienst. Die maximale Anzahl von Anfragen pro Sekunde für den AWS IoT Device Shadow-Dienst hängt von Ihrem ab AWS-Region. Weitere Informationen finden Sie in den Grenzwerten für die AWS IoT Device Shadow Service API in der Allgemeine Amazon Web Services-Referenz.

`shadowDocumentSizeLimitBytes`

(Optional) Die maximal zulässige Größe jedes JSON-Statusdokuments für lokale Schatten.

Wenn Sie diesen Wert erhöhen, müssen Sie auch das Ressourcenlimit für das JSON-Statusdokument für Wolkenschatten erhöhen. Weitere Informationen finden Sie in den Grenzwerten für die AWS IoT Device Shadow Service API in der Allgemeine Amazon Web Services-Referenz.

Standard: 8192 Byte

Maximum: 30720 Byte

Example Beispiel: Update zur Zusammenführung von Konfigurationen

Das folgende Beispiel zeigt ein Beispiel für ein Update zur Zusammenführung von Konfigurationen mit allen verfügbaren Konfigurationsparametern für die Shadow Manager-Komponente.

```
{
   "strategy":{
      "type":"periodic",
      "delay":300
   },
   "synchronize":{
      "shadowDocumentsMap":{
         "MyDevice1":{
            "classic":false,
            "namedShadows":[
               "MyShadowA",
               "MyShadowB"
            ]
         },
         "MyDevice2":{
            "classic":true,
            "namedShadows":[]
         }
      },
      "direction":"betweenDeviceAndCloud"
   },
   "rateLimits":{
      "maxOutboundSyncUpdatesPerSecond":100,
      "maxTotalLocalRequestsRate":200,
      "maxLocalRequestsPerSecondPerThing":20
   },
   "shadowDocumentSizeLimitBytes":8192
}
```

2.1.x

`strategy`

(Optional) Die Strategie, mit der diese Komponente Schatten zwischen AWS IoT Core und dem Kerngerät synchronisiert.

Dieses Objekt enthält die folgenden Informationen.

`type`

(Optional) Die Art der Strategie, die diese Komponente verwendet, um Schatten zwischen AWS IoT Core und dem Kerngerät zu synchronisieren. Wählen Sie aus den folgenden Optionen aus:

- `realTime`— Synchronisiert Schatten bei AWS IoT Core jedem Shadow-Update.
- `periodic`— Synchronisieren Sie Schatten in einem regelmäßigen Intervall, das Sie mit dem `delay` Konfigurationsparameter angeben. AWS IoT Core

Standard: `realTime`

`delay`

(Optional) Das Intervall in Sekunden, mit dem diese Komponente Schatten synchronisiert AWS IoT Core, wenn Sie die `periodic` Synchronisierungsstrategie angeben.

> **Note**
>
> Dieser Parameter ist erforderlich, wenn Sie die `periodic` Synchronisierungsstrategie angeben.

`synchronize`

(Optional) Die Synchronisierungseinstellungen, die bestimmen, wie Schatten mit dem AWS Cloud synchronisiert werden.

> **Note**
>
> Sie müssen ein Konfigurationsupdate mit dieser Eigenschaft erstellen, um Schatten mit dem AWS Cloud zu synchronisieren.

Dieses Objekt enthält die folgenden Informationen.

`coreThing`

(Optional) Das Core-Gerät wird zur Synchronisation gescrollt. Dieses Objekt enthält die folgenden Informationen.

`classic`

(Optional) Standardmäßig synchronisiert der Shadow-Manager den lokalen Status des klassischen Shadows für Ihr Kerngerät mit dem AWS Cloud. Wenn Sie den klassischen Geräteshadow nicht synchronisieren möchten, setzen Sie ihn auf`false`.

Standard: `true`

`namedShadows`

(Optional) Die Liste der benannten Core-Device-Shadows, die synchronisiert werden sollen. Sie müssen die genauen Namen der Schatten angeben.

> **Warning**
>
> Der AWS IoT Greengrass Dienst verwendet den `AWSManagedGreengrassV2Deployment` benannten Shadow, um Bereitstellungen zu verwalten, die auf einzelne Kerngeräte abzielen. Dieser benannte Shadow ist für die Verwendung durch den AWS IoT Greengrass Dienst reserviert. Aktualisieren oder löschen Sie diesen benannten Schatten nicht.

`shadowDocumentsMap`

(Optional) Die zusätzlichen Geräteschatten, die synchronisiert werden sollen. Die Verwendung dieses Konfigurationsparameters erleichtert die Angabe von Schattendokumenten. Es wird empfohlen, diesen Parameter anstelle des `shadowDocuments` Objekts zu verwenden.

> **Note**
>
> Wenn Sie ein `shadowDocumentsMap` Objekt angeben, dürfen Sie kein `shadowDocuments` Objekt angeben.

Jedes Objekt enthält die folgenden Informationen:

thingName

Die Shadow-Konfiguration für den *thingName* für diese Shadow-Konfiguration.

`classic`

(Optional) Wenn Sie den klassischen Geräteshadow für das `thingName` Gerät nicht synchronisieren möchten, setzen Sie ihn auf`false`.

`namedShadows`

Die Liste der benannten Schatten, die Sie synchronisieren möchten. Sie müssen die genauen Namen der Schatten angeben.

`shadowDocuments`

(Optional) Die Liste der zusätzlichen Geräteschatten, die synchronisiert werden sollen. Wir empfehlen, stattdessen den `shadowDocumentsMap` Parameter zu verwenden.

> **Note**
>
> Wenn Sie ein `shadowDocuments` Objekt angeben, dürfen Sie kein `shadowDocumentsMap` Objekt angeben.

Jedes Objekt in dieser Liste enthält die folgenden Informationen.

`thingName`

Der Dingname des Geräts, für das Schatten synchronisiert werden sollen.

`classic`

(Optional) Wenn Sie den klassischen Geräteshadow für das `thingName` Gerät nicht synchronisieren möchten, setzen Sie ihn auf`false`.

Standard: `true`

`namedShadows`

(Optional) Die Liste der benannten Geräteschatten, die Sie synchronisieren möchten. Sie müssen die genauen Namen der Schatten angeben.

`rateLimits`

(Optional) Die Einstellungen, die die Ratenlimits für Shadowdienstanforderungen festlegen.

Dieses Objekt enthält die folgenden Informationen.

`maxOutboundSyncUpdatesPerSecond`

(Optional) Die maximale Anzahl von Synchronisierungsanfragen pro Sekunde, die das Gerät überträgt.

Standard: 100 Anfragen/Sekunde

`maxTotalLocalRequestsRate`

(Optional) Die maximale Anzahl lokaler IPC-Anfragen pro Sekunde, die an das Kerngerät gesendet werden.

Standard: 200 Anfragen/Sekunde

`maxLocalRequestsPerSecondPerThing`

(Optional) Die maximale Anzahl lokaler IPC-Anfragen pro Sekunde, die für jedes verbundene IoT-Ding gesendet werden.

Standard: 20 Anfragen/Sekunde für jedes Ding

> **Note**
>
> Diese Parameter für die Ratenbegrenzung definieren die maximale Anzahl von Anfragen pro Sekunde für den lokalen Shadow-Dienst. Die maximale Anzahl von Anfragen pro Sekunde für den AWS IoT Device Shadow-Dienst hängt von Ihrem ab AWS-Region. Weitere Informationen finden Sie in den Grenzwerten für die AWS IoT Device Shadow Service API in der Allgemeine Amazon Web Services-Referenz.

`shadowDocumentSizeLimitBytes`

(Optional) Die maximal zulässige Größe jedes JSON-Statusdokuments für lokale Schatten.

Wenn Sie diesen Wert erhöhen, müssen Sie auch das Ressourcenlimit für das JSON-Statusdokument für Wolkenschatten erhöhen. Weitere Informationen finden Sie in den Grenzwerten für die AWS IoT Device Shadow Service API in der Allgemeine Amazon Web Services-Referenz.

Standard: 8192 Byte

Maximum: 30720 Byte

Example Beispiel: Update zur Zusammenführung von Konfigurationen

Das folgende Beispiel zeigt ein Beispiel für ein Update zur Zusammenführung von Konfigurationen mit allen verfügbaren Konfigurationsparametern für die Shadow Manager-Komponente.

```
{
   "strategy":{
      "type":"periodic",
      "delay":300
   },
   "synchronize":{
      "shadowDocumentsMap":{
         "MyDevice1":{
            "classic":false,
            "namedShadows":[
               "MyShadowA",
               "MyShadowB"
            ]
         },
         "MyDevice2":{
            "classic":true,
            "namedShadows":[]
         }
      },
      "direction":"betweenDeviceAndCloud"
   },
   "rateLimits":{
      "maxOutboundSyncUpdatesPerSecond":100,
      "maxTotalLocalRequestsRate":200,
      "maxLocalRequestsPerSecondPerThing":20
   },
   "shadowDocumentSizeLimitBytes":8192
}
```

2.0.x

`synchronize`

(Optional) Die Synchronisierungseinstellungen, die bestimmen, wie Schatten mit dem AWS Cloud synchronisiert werden.

> **Note**
>
> Sie müssen ein Konfigurationsupdate mit dieser Eigenschaft erstellen, um Schatten mit dem AWS Cloud zu synchronisieren.

Dieses Objekt enthält die folgenden Informationen.

`coreThing`

(Optional) Das Core-Gerät wird zur Synchronisation gescrollt. Dieses Objekt enthält die folgenden Informationen.

`classic`

(Optional) Standardmäßig synchronisiert der Shadow-Manager den lokalen Status des klassischen Shadows für Ihr Kerngerät mit dem AWS Cloud. Wenn Sie den klassischen Geräteshadow nicht synchronisieren möchten, setzen Sie ihn auf`false`.

Standard: `true`

`namedShadows`

(Optional) Die Liste der benannten Core-Device-Shadows, die synchronisiert werden sollen. Sie müssen die genauen Namen der Schatten angeben.

> **Warning**
>
> Der AWS IoT Greengrass Dienst verwendet den `AWSManagedGreengrassV2Deployment` benannten Shadow, um Bereitstellungen zu verwalten, die auf einzelne Kerngeräte abzielen. Dieser benannte Shadow ist für die Verwendung durch den AWS IoT Greengrass Dienst reserviert. Aktualisieren oder löschen Sie diesen benannten Schatten nicht.

`shadowDocumentsMap`

(Optional) Die zusätzlichen Geräteschatten, die synchronisiert werden sollen. Die Verwendung dieses Konfigurationsparameters erleichtert die Angabe von Schattendokumenten. Es wird empfohlen, diesen Parameter anstelle des `shadowDocuments` Objekts zu verwenden.

> **Note**
>
> Wenn Sie ein `shadowDocumentsMap` Objekt angeben, dürfen Sie kein `shadowDocuments` Objekt angeben.

Jedes Objekt enthält die folgenden Informationen:

`thingName`

Die Shadow-Konfiguration für den *`thingName`* für diese Shadow-Konfiguration.

`classic`

(Optional) Wenn Sie den klassischen Geräteshadow für das `thingName` Gerät nicht synchronisieren möchten, setzen Sie ihn auf`false`.

`namedShadows`

Die Liste der benannten Schatten, die Sie synchronisieren möchten. Sie müssen die genauen Namen der Schatten angeben.

`shadowDocuments`

(Optional) Die Liste der zusätzlichen Geräteschatten, die synchronisiert werden sollen. Wir empfehlen, stattdessen den `shadowDocumentsMap` Parameter zu verwenden.

> **Note**
>
> Wenn Sie ein `shadowDocuments` Objekt angeben, dürfen Sie kein `shadowDocumentsMap` Objekt angeben.

Jedes Objekt in dieser Liste enthält die folgenden Informationen.

`thingName`

Der Dingname des Geräts, für das Schatten synchronisiert werden sollen.

`classic`

(Optional) Wenn Sie den klassischen Geräteshadow für das `thingName` Gerät nicht synchronisieren möchten, setzen Sie ihn auf`false`.

Standard: `true`

`namedShadows`

(Optional) Die Liste der benannten Geräteschatten, die Sie synchronisieren möchten. Sie müssen die genauen Namen der Schatten angeben.

`rateLimits`

(Optional) Die Einstellungen, die die Ratenlimits für Shadowdienstanforderungen festlegen.

Dieses Objekt enthält die folgenden Informationen.

`maxOutboundSyncUpdatesPerSecond`

(Optional) Die maximale Anzahl von Synchronisierungsanfragen pro Sekunde, die das Gerät überträgt.

Standard: 100 Anfragen/Sekunde

`maxTotalLocalRequestsRate`

(Optional) Die maximale Anzahl lokaler IPC-Anfragen pro Sekunde, die an das Kerngerät gesendet werden.

Standard: 200 Anfragen/Sekunde

`maxLocalRequestsPerSecondPerThing`

(Optional) Die maximale Anzahl lokaler IPC-Anfragen pro Sekunde, die für jedes verbundene IoT-Ding gesendet werden.

Standard: 20 Anfragen/Sekunde für jedes Ding

> **Note**
>
> Diese Parameter für die Ratenbegrenzung definieren die maximale Anzahl von Anfragen pro Sekunde für den lokalen Shadow-Dienst. Die maximale Anzahl von Anfragen pro Sekunde für den AWS IoT Device Shadow-Dienst hängt von Ihrem ab AWS-Region. Weitere Informationen finden Sie in den Grenzwerten für die AWS IoT Device Shadow Service API in der Allgemeine Amazon Web Services-Referenz.

`shadowDocumentSizeLimitBytes`

(Optional) Die maximal zulässige Größe jedes JSON-Statusdokuments für lokale Schatten.

Wenn Sie diesen Wert erhöhen, müssen Sie auch das Ressourcenlimit für das JSON-Statusdokument für Wolkenschatten erhöhen. Weitere Informationen finden Sie in den Grenzwerten für die AWS IoT Device Shadow Service API in der Allgemeine Amazon Web Services-Referenz.

Standard: 8192 Byte

Maximum: 30720 Byte

Example Beispiel: Update zur Zusammenführung von Konfigurationen

Das folgende Beispiel zeigt ein Beispiel für ein Update zur Zusammenführung von Konfigurationen mit allen verfügbaren Konfigurationsparametern für die Shadow Manager-Komponente.

```
{
  "synchronize": {
    "coreThing": {
      "classic": true,
      "namedShadows": [
        "MyCoreShadowA",
        "MyCoreShadowB"
      ]
    },
    "shadowDocuments": [
      {
        "thingName": "MyDevice1",
        "classic": false,
        "namedShadows": [
          "MyShadowA",
          "MyShadowB"
        ]
      },
      {
        "thingName": "MyDevice2",
        "classic": true,
        "namedShadows": []
      }
    ]
  },
  "rateLimits": {
    "maxOutboundSyncUpdatesPerSecond": 100,
    "maxTotalLocalRequestsRate": 200,
```

```
    "maxLocalRequestsPerSecondPerThing": 20
  },
  "shadowDocumentSizeLimitBytes": 8192
}
```

Lokale Protokolldatei

Diese Komponente verwendet dieselbe Protokolldatei wie die Greengrass Nucleus-Komponente.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

Um die Protokolle dieser Komponente einzusehen

- Führen Sie den folgenden Befehl auf dem Kerngerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

  Linux

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

  Windows (PowerShell)

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

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
2.3.8	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem der Shadow Manager während der MQTT-Client-Verbindung eine Deadlock-Situation erzeugt.
2.3.7	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem Shadow Manager während einer Shadow Manager-Synchronisierung regelmäßig einen `NullPointerExcepti on` Fehler protokolliert.
2.3.6	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem Shadow-Eigenschaften, die durch AWS Cloud Updates gelöscht wurden, während das Gerät offline ist, nach Wiederherstellung der Konnektivität weiterhin im lokalen Shadow vorhanden sind.
2.3.5	Die Version wurde für die Version 2.12.0 von Greengrass Nucleus aktualisiert.
2.3.4	Fehlerkorrekturen und Verbesserungen • Integriert die Unterstützung für Dokumente mit Nullstatus und leeren Shadow-State-Dokumenten.
2.3.3	Die Version wurde für die Version 2.11.0 von Greengrass Nucleus aktualisiert.
2.3.2	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem der Shadow Manager in den `BROKEN` Status wechselt, wenn die lokale Shadow-Datenbank beschädigt ist. • Die Version wurde für die Version 2.10.0 von Greengrass Nucleus aktualisiert.
2.3.1	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, das die Synchronisierung von Cloud-Shadow-Updates verhindern kann. • Behebt ein Problem, bei dem Änderungen an der Sync-Konfiguration für benannte Schatten nur für einen benannten Schatten gelten.

Version	Änderungen
2.3.0	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, das dazu führen kann, dass Schatten nicht synchronisiert werden, wenn der private Schlüssel des Greengrass-Geräts in einem Hardware-Sicherheitsmodul gespeichert ist.
2.2.4	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem die Überprüfung der Schattengröße nicht mit der der Cloud übereinstimmte, wenn das lokale Schattendokument aktualisiert wurde. • Behebt ein Problem, bei dem der Shadow-Manager keine Konfigura tionsupdates mehr abhört, wenn eine Bereitstellung `RESET` auf den Konfigurationsknoten ausgeführt wird.
2.2.3	Die Version wurde für die Version 2.9.0 von Greengrass Nucleus aktualisiert.
2.2.2	Die Version wurde für die Version 2.8.0 von Greengrass Nucleus aktualisiert.
2.2.1	Die Version wurde für die Version 2.7.0 von Greengrass Nucleus aktualisiert.

Version	Änderungen
2.2.0	Neue Features • Integriert die Unterstützung für den lokalen Shadow-Service über die lokale Publish/Subscribe-Schnittstelle. Sie können jetzt mit dem lokalen Publish/Subscribe-Nachrichtenbroker über Shadow-MQTT-Themen kommunizieren, um Schatten auf dem Kerngerät abzurufen, zu aktualisieren und zu löschen. Mit dieser Funktion können Sie Client-Geräte mit dem lokalen Shadow-Service verbinden, indem Sie die MQTT-Bridge verwenden, um Nachrichten zu Shadow-Themen zwischen Client-Geräten und der lokalen Publish/Subscribe-Schnittstelle weiterzuleiten. Für diese Funktion ist Version 2.6.0 oder höher der Greengrass Nucleus-Komponente erforderlich. Um Client-Geräte mit dem lokalen Shadow-Service zu verbinden, müssen Sie auch Version 2.2.0 oder höher der MQTT-Bridge-Komponente verwenden. • Fügt die `direction` Option hinzu, die Sie konfigurieren können, um die Richtung für die Synchronisation von Shadows zwischen dem lokalen Shadow-Dienst und dem anzupassen. AWS Cloud Sie können diese Option konfigurieren, um die Bandbreite und die Verbindungen zum zu reduzieren AWS Cloud.
2.1.1	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem die maximale Tiefe in den `reported` Abschnitten `desired` und im Dokument mit dem Schattenstatus des JSON-Geräts 4 statt 5 Stufen betrug. • Die Version wurde für die Version 2.6.0 von Greengrass Nucleus aktualisiert.
2.1.0	Neue Features • Integriert die Unterstützung für periodische Shadow-Synchronisierungsintervalle, sodass Sie das Kerngerät so konfigurieren können, dass Bandbreitennutzung und Gebühren reduziert werden.
2.0.6	Diese Version enthält Fehlerkorrekturen und Verbesserungen.
2.0.5	Die Version wurde für die Version 2.5.0 von Greengrass Nucleus aktualisiert.

Version	Änderungen
2.0.4	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, das dazu führte, dass der Schattenmanager neu erstellte Versionen aller zuvor gelöschten Schatten löschte. • Aktualisiert den `DeleteThingShadow` IPC-Vorgang so, dass die Shadow-Version beim Aufruf inkrementiert wird.
2.0.3	Die Version wurde für die Version 2.4.0 von Greengrass Nucleus aktualisiert.
2.0.2	Fehlerkorrekturen und Verbesserungen • Es wurde ein Problem behoben, das dazu führte, dass der Schattenm anager die `delta` Eigenschaft beim Synchronisieren von Schattenstatus von AWS IoT Core nicht erkannte. • Es wurde ein Problem behoben, das manchmal dazu führte, dass Synchronisierungsanforderungen für einen Schatten falsch zusammeng eführt wurden.
2.0.1	Die Version wurde für die Version 2.3.0 von Greengrass Nucleus aktualisiert.
2.0.0	Erste Version

Amazon SNS

Die Amazon SNS-Komponente (`aws.greengrass.SNS`) veröffentlicht Nachrichten zu einem Amazon Simple Notification Service (Amazon SNS)-Thema. Sie können diese Komponente verwenden, um Ereignisse von Greengrass-Core-Geräten an Webserver, E-Mail-Adressen und andere Nachrichtenabonnenten zu senden. Weitere Informationen finden Sie unter Was ist Amazon SNS ?im Amazon Simple Notification Service-Entwicklerhandbuch.

Um in einem Amazon SNS-Thema mit dieser Komponente zu veröffentlichen, veröffentlichen Sie eine Nachricht zu dem Thema, in dem diese Komponente abonniert wird. Standardmäßig abonniert diese Komponente das `sns/message` lokale Veröffentlichungs-/Abonnementthema. Sie können bei der Bereitstellung dieser Komponente andere Themen angeben, einschließlich AWS IoT Core MQTT-Themen.

In Ihrer benutzerdefinierten Komponente möchten Sie möglicherweise Filter- oder Formatierungslogik implementieren, um Nachrichten aus anderen Quellen zu verarbeiten, bevor Sie sie in dieser Komponente veröffentlichen. Auf diese Weise können Sie Ihre Nachrichtenverarbeitungslogik auf einer einzigen Komponente zentralisieren.

> **Note**
>
> Diese Komponente bietet ähnliche Funktionen wie der Amazon SNS-Konnektor in AWS IoT Greengrass V1. Weitere Informationen finden Sie unter Amazon SNS-Konnektor im AWS IoT Greengrass V1-Entwicklerhandbuch.

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.1.x
- 2.0.x

Typ

Diese Komponente ist eine Lambda-Komponente (`aws.greengrass.lambda`). Der Greengrass-Kern führt die Lambda-Funktion dieser Komponente mit der Lambda-Launcher-Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann nur auf Linux-Core-Geräten installiert werden.

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Ihr Core-Gerät muss die Anforderungen für die Ausführung von Lambda-Funktionen erfüllen. Wenn Sie möchten, dass das Core-Gerät containerisierte Lambda-Funktionen ausführt, muss das Gerät die Voraussetzungen dafür erfüllen. Weitere Informationen finden Sie unter Anforderungen an die Lambda-Funktion.
- Python Version 3.7 ist auf dem Core-Gerät installiert und der PATH-Umgebungsvariablen hinzugefügt.
- Amazon SNS-Thema. Weitere Informationen finden Sie unter Amazon SNS-Thema anlegen im Amazon Simple Notification Service-Entwicklerhandbuch.
- Die Greengrass-Geräterolle muss die `sns:Publish` Aktion zulassen, wie in der folgenden Beispiel-IAM-Richtlinie gezeigt.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "sns:Publish"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:sns:region:account-id:topic-name"
      ]
    }
  ]
}
```

Sie können das Standardthema in der Nutzlast der Eingabenachricht für diese Komponente dynamisch überschreiben. Wenn Ihre Anwendung diese Funktion verwendet, muss die IAM-Richtlinie alle Zielthemen als Ressourcen enthalten. Sie können Ressourcen granularen oder bedingten Zugriff gewähren (etwa mit einem Benennungsschema mit Platzhaltern *).

- Um Ausgabedaten von dieser Komponente zu erhalten, müssen Sie das folgende Konfigurationsupdate für die Legacy-Abonnement-Routerkomponente (`aws.greengrass.LegacySubscriptionRouter`) zusammenführen, wenn Sie diese Komponente bereitstellen. Diese Konfiguration gibt das Thema an, in dem diese Komponente Antworten veröffentlicht.

  Legacy subscription router v2.1.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-sns": {
        "id": "aws-greengrass-sns",
        "source": "component:aws.greengrass.SNS",
        "subject": "sns/message/status",
        "target": "cloud"
      }
    }
  }
  ```

  Legacy subscription router v2.0.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-sns": {
        "id": "aws-greengrass-sns",
        "source": "arn:aws:lambda:region:aws:function:aws-greengrass-sns:version",
        "subject": "sns/message/status",
        "target": "cloud"
      }
    }
  }
  ```

  - Ersetzen Sie *`region`* durch die AWS-Region , die Sie verwenden.
  - Ersetzen Sie *`Version`* durch die Version der Lambda-Funktion, die diese Komponente ausführt. Um die Version der Lambda-Funktion zu finden, müssen Sie das Rezept für

die Version dieser Komponente anzeigen, die Sie bereitstellen möchten. Öffnen Sie die Detailseite dieser Komponente in der AWS IoT Greengrass Konsole und suchen Sie nach dem Schlüssel-Wert-Paar der Lambda-Funktion. Dieses Schlüssel-Wert-Paar enthält den Namen und die Version der Lambda-Funktion.

> **Important**
>
> Sie müssen die Lambda-Funktionsversion auf dem Legacy-Abonnement-Router jedes Mal aktualisieren, wenn Sie diese Komponente bereitstellen. Dadurch wird sichergestellt, dass Sie die richtige Lambda-Funktionsversion für die von Ihnen bereitgestellte Komponentenversion verwenden.

Weitere Informationen finden Sie unter Erstellen von Bereitstellungen.

- Die Amazon SNS-Komponente wird für die Ausführung in einer VPC unterstützt. Um diese Komponente in einer VPC bereitzustellen, ist Folgendes erforderlich.
  - Die Amazon SNS-Komponente muss über Konnektivität mit `sns.region.amazonaws.com` dem VPC-Endpunkt verfügen`com.amazonaws.us-east-1.sns`.

Endpunkte und Ports

Diese Komponente muss in der Lage sein, ausgehende Anfragen an die folgenden Endpunkte und Ports auszuführen, zusätzlich zu den für den grundlegenden Betrieb erforderlichen Endpunkten und Ports. Weitere Informationen finden Sie unter Zulassen von Gerätedatenverkehr über einen Proxy oder eine Firewall.

Endpunkt	Port	Erforderlich	Beschreibung
`sns.region.amazonaws.com`	443	Ja	Veröffentlichen Sie Nachrichten in Amazon SNS.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und alle ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitzustellen. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente und die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste Abhängigkeiten.

2.1.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.13.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.1.6

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.12.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.1.5

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.11.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.1.4

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.10.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.1.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.9.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Token-Exchange-Service	^2.0.0	Hart

2.1.2

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.2 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.8.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.1.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.7.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.0.8 - 2.1.0

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.0.8 und 2.1.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.6.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.0.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.5.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.0.6

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.4.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.0.5

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.3.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.0.4

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.2.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2.0.0	Weich
Token-Exchange-Service	^2.0.0	Hart

2.0.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.3 <2.1.0	Hart
Lambda-Launcher	>=1.0.0	Hart
Lambda-Laufzeiten	>=1.0.0	Weich

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Token-Exchange-Service	>=1.0.0	Hart

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zum Komponentenrezept .

Konfiguration

Diese Komponente bietet die folgenden Konfigurationsparameter, die Sie anpassen können, wenn Sie die Komponente bereitstellen.

> **Note**
>
> Die Standardkonfiguration dieser Komponente enthält Lambda-Funktionsparameter. Wir empfehlen Ihnen, nur die folgenden Parameter zu bearbeiten, um diese Komponente auf Ihren Geräten zu konfigurieren.

`lambdaParams`

Ein Objekt, das die Parameter für die Lambda-Funktion dieser Komponente enthält. Dieses Objekt enthält die folgenden Informationen:

`EnvironmentVariables`

Ein Objekt, das die Parameter der Lambda-Funktion enthält. Dieses Objekt enthält die folgenden Informationen:

`DEFAULT_SNS_ARN`

Der ARN des standardmäßigen Amazon SNS-Themas, in dem diese Komponente Nachrichten veröffentlicht. Sie können das Zielthema mit der `-sns_topic_arn`Eigenschaft in der Nutzlast der Eingabenachricht überschreiben.

`containerMode`

(Optional) Der Containerisierungsmodus für diese Komponente. Wählen Sie aus den folgenden Optionen aus:

- `NoContainer` – Die Komponente wird nicht in einer isolierten Laufzeitumgebung ausgeführt.

- `GreengrassContainer` – Die Komponente wird in einer isolierten Laufzeitumgebung innerhalb des AWS IoT Greengrass Containers ausgeführt.

Standard: `GreengrassContainer`

`containerParams`

(Optional) Ein Objekt, das die Containerparameter für diese Komponente enthält. Die Komponente verwendet diese Parameter, wenn Sie `GreengrassContainer` für angeben`containerMode`.

Dieses Objekt enthält die folgenden Informationen:

`memorySize`

(Optional) Die Menge an Arbeitsspeicher (in Kilobyte), die der Komponente zugewiesen werden soll.

Der Standardwert ist 512 MB (525.312 KB).

`pubsubTopics`

(Optional) Ein Objekt, das die Themen enthält, in denen die Komponente den Empfang von Nachrichten abonniert. Sie können jedes Thema angeben und angeben, ob die Komponente MQTT-Themen von AWS IoT Core oder lokale Veröffentlichungs-/Abonnementthemen abonniert.

Dieses Objekt enthält die folgenden Informationen:

`0` – Dies ist ein Array-Index als Zeichenfolge.

Ein Objekt, das die folgenden Informationen enthält:

`type`

(Optional) Der Typ des Veröffentlichungs-/Abonnement-Messagings, den diese Komponente zum Abonnieren von Nachrichten verwendet. Wählen Sie aus den folgenden Optionen aus:

- `PUB_SUB` — Abonnieren Sie lokale Veröffentlichen/Abonnement-Nachrichten. Wenn Sie diese Option wählen, darf das Thema keine MQTT-Platzhalter enthalten. Weitere Informationen zum Senden von Nachrichten von einer benutzerdefinierten Komponente, wenn Sie diese Option angeben, finden Sie unter Lokale Nachrichten veröffentlichen/abonnieren.
- `IOT_CORE` – Abonnieren Sie AWS IoT Core MQTT-Nachrichten. Wenn Sie diese Option wählen, kann das Thema MQTT-Platzhalter enthalten. Weitere Informationen zum

Senden von Nachrichten von benutzerdefinierten Komponenten, wenn Sie diese Option angeben, finden Sie unter MQTT-Nachrichten veröffentlichen/abonnieren AWS IoT Core.

Standard: PUB_SUB

`topic`

(Optional) Das Thema, das die Komponente abonniert, um Nachrichten zu empfangen. Wenn Sie `IotCore` für angeben`type`, können Sie in diesem Thema MQTT-Platzhalter (+ und #) verwenden.

Example Beispiel: Aktualisierung der Konfigurationszusammenführung (Container-Modus)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_SNS_ARN": "arn:aws:sns:us-west-2:123456789012:mytopic"
    }
  },
  "containerMode": "GreengrassContainer"
}
```

Example Beispiel: Aktualisierung der Konfigurationszusammenführung (kein Containermodus)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_SNS_ARN": "arn:aws:sns:us-west-2:123456789012:mytopic"
    }
  },
  "containerMode": "NoContainer"
}
```

Eingabedaten

Diese Komponente akzeptiert Nachrichten zum folgenden Thema und veröffentlicht die Nachricht unverändert im Amazon SNS-Zielthema. Standardmäßig abonniert diese Komponente lokale Veröffentlichungs-/Abonnementnachrichten. Weitere Informationen zum Veröffentlichen von Nachrichten in dieser Komponente aus Ihren benutzerdefinierten Komponenten finden Sie unter Lokale Nachrichten veröffentlichen/abonnieren.

Standardthema (lokales Veröffentlichen/Abonnieren): `sns/message`

Die Nachricht akzeptiert die folgenden Eigenschaften. Eingabenachrichten müssen im JSON-Format vorliegen.

`request`

Die Informationen über die Nachricht, die an das Amazon SNS-Thema gesendet werden soll.

Typ: `object`, der die folgenden Informationen enthält:

`message`

Der Inhalt der Nachricht als Zeichenfolge.

Um ein JSON-Objekt zu senden, serialisieren Sie es als Zeichenfolge und geben Sie `json` für die -`message_structure`Eigenschaft an.

Typ: `string`

`subject`

(Optional) Der Betreff der Nachricht.

Typ: `string`

Das Subjekt kann ASCII-Text und bis zu 100 Zeichen sein. Er muss mit einem Buchstaben, einer Zahl oder einem Satzzeichen beginnen. Sie darf keine Zeilenumbrüche oder Steuerzeichen enthalten.

`sns_topic_arn`

(Optional) Der ARN des Amazon SNS-Themas, in dem diese Komponente die Nachricht veröffentlicht. Geben Sie diese Eigenschaft an, um das standardmäßige Amazon SNS-Thema zu überschreiben.

Typ: `string`

`message_structure`

(Optional) Die Struktur der Nachricht. Geben Sie an, `json` um eine JSON-Nachricht zu senden, die Sie als Zeichenfolge in der -`content`Eigenschaft serialisieren.

Typ: `string`

Zulässige Werte: `json`

`id`

Eine willkürliche ID für die Anforderung. Verwenden Sie diese Eigenschaft, um eine Eingabeanforderung einer Ausgabeantwort zuzuordnen. Wenn Sie diese Eigenschaft angeben, legt die Komponente die `id` Eigenschaft im Antwortobjekt auf diesen Wert fest.

Typ: `string`

> **Note**
>
> Die Nachrichtengröße kann maximal 256 KB betragen.

Example Beispieleingabe: Zeichenfolgen-Nachricht

```
{
  "request": {
    "subject": "Message subject",
    "message": "Message data",
    "sns_topic_arn": "arn:aws:sns:region:account-id:topic2-name"
  },
  "id": "request123"
}
```

Example Beispieleingabe: JSON-Nachricht

```
{
  "request": {
    "subject": "Message subject",
    "message": "{ \"default\": \"Message data\" }",
    "message_structure": "json"
  },
  "id": "request123"
}
```

Ausgabedaten

Diese Komponente veröffentlicht Antworten standardmäßig als Ausgabedaten für das folgende MQTT-Thema. Sie müssen dieses Thema als `subject` in der Konfiguration für die Legacy-

Abonnement-Routerkomponente angeben. Weitere Informationen zum Abonnieren von Nachrichten zu diesem Thema in Ihren benutzerdefinierten Komponenten finden Sie unter MQTT-Nachrichten veröffentlichen/abonnieren AWS IoT Core.

Standardthema (AWS IoT Core MQTT): `sns/message/status`

Example Beispielausgabe: Erfolg

```
{
  "response": {
    "sns_message_id": "f80a81bc-f44c-56f2-a0f0-d5af6a727c8a",
    "status": "success"
  },
  "id": "request123"
}
```

Example Beispielausgabe: Fehler

```
{
  "response" : {
    "error": "InvalidInputException",
    "error_message": "SNS Topic Arn is invalid",
    "status": "fail"
  },
  "id": "request123"
}
```

Lokale Protokolldatei

Diese Komponente verwendet die folgende Protokolldatei.

```
/greengrass/v2/logs/aws.greengrass.SNS.log
```

So zeigen Sie die Protokolle dieser Komponente an

- Führen Sie den folgenden Befehl auf dem Core-Gerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

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

Lizenzen

Diese Komponente 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

Diese Komponente wird gemäß dem Greengrass Core Software License Agreement veröffentlicht.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in jeder Version der Komponente beschrieben.

Version	Änderungen
2.1.7	Version für Greengrass-Kern Version 2.12.0 aktualisiert.
2.1.6	Version für Greengrass-Kern Version 2.11.0 aktualisiert.
2.1.5	Version für Greengrass-Kern Version 2.10.0 aktualisiert.
2.1.4	Version für Greengrass-Kern Version 2.9.0 aktualisiert.
2.1.3	Version für Greengrass-Kern Version 2.8.0 aktualisiert.
2.1.2	Version für Greengrass-Kern Version 2.7.0 aktualisiert.
2.1.1	Version für Greengrass-Kern Version 2.6.0 aktualisiert.
2.1.0	Neue Features • Fügt Unterstützung für HTTPS-Netzwerk-Proxy-Konfigurationen hinzu. Weitere Informationen finden Sie unter Verbindungsherstellung auf

Version	Änderungen
	Port 443 oder über einen Netzwerk-Proxy und Ermöglichen Sie dem Kerngerät, einem HTTPS-Proxy zu vertrauen.
2.0.8	Version für Greengrass-Kern Version 2.5.0 aktualisiert.
2.0.7	Version für Greengrass-Kern Version 2.4.0 aktualisiert.
2.0.6	Version für Greengrass-Kern Version 2.3.0 aktualisiert.
2.0.5	Version für Greengrass-Kern Version 2.2.0 aktualisiert.
2.0.4	Version für Greengrass-Kern Version 2.1.0 aktualisiert.
2.0.3	Erste Version

Stream-Manager

Mit der Stream-Manager-Komponente (`aws.greengrass.StreamManager`) können Sie Datenströme verarbeiten, die AWS Cloud von Greengrass-Core-Geräten in übertragen werden.

Weitere Informationen zum Konfigurieren und Verwenden von Stream Manager in benutzerdefinierten Komponenten finden Sie unter Verwalten von Datenströmen auf Greengrass-Core-Geräten.

Themen

- Versionen
- Typ
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Konfiguration
- Lokale Protokolldatei
- Änderungsprotokoll

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.1.x
- 2.0.x

> **Note**
>
> Wenn Sie Stream Manager verwenden, um Daten in die Cloud zu exportieren, können Sie Version 2.0.7 der Stream Manager-Komponente nicht auf eine Version zwischen v2.0.8 und v2.0.11 aktualisieren. Wenn Sie Stream Manager zum ersten Mal bereitstellen, empfehlen wir dringend, die neueste Version der Stream-Manager-Komponente bereitzustellen.

Typ

Diese Komponente ist eine generische Komponente (`aws.greengrass.generic`). Der Greengrass-Kern führt die Lebenszyklusskripte der Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf -Core-Geräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Die Token-Exchange-Rolle muss den Zugriff auf die AWS Cloud Ziele ermöglichen, die Sie mit dem Stream-Manager verwenden. Weitere Informationen finden Sie hier:
  - the section called “AWS IoT Analytics-Kanäle”
  - the section called “Amazon Kinesis-Datenströme”

  - the section called “AWS IoT SiteWise Komponenteneigenschaften”
  - the section called “Amazon S3-Objekte”
- Die Stream-Manager-Komponente wird für die Ausführung in einer VPC unterstützt. Um diese Komponente in einer VPC bereitzustellen, ist Folgendes erforderlich.
  - Die Stream-Manager-Komponente muss über Konnektivität zu dem AWS Service verfügen, in dem Sie Daten veröffentlichen.
    - Amazon S3: `com.amazonaws.region.s3`
    - Amazon Kinesis Data Streams: `com.amazonaws.region.kinesis-streams`
    - AWS IoT SiteWise: `com.amazonaws.region.iotsitewise.data`
  - Wenn Sie Daten in Amazon S3 in der `us-east-1` Region veröffentlichen, versucht diese Komponente standardmäßig, den globalen S3-Endpunkt zu verwenden. Dieser Endpunkt ist jedoch nicht über den Amazon S3-VPC-Schnittstellenendpunkt verfügbar. Weitere Informationen finden Sie unter Einschränkungen und Begrenzungen von AWS PrivateLink für Amazon S3. Um dies zu beheben, können Sie aus den folgenden Optionen wählen.
    - Konfigurieren Sie die Stream-Manager-Komponente für die Verwendung des regionalen S3-Endpunkts in der `us-east-1` Region, indem Sie die `AWS_S3_US_EAST_1_REGIONAL_ENDPOINT=regional` Umgebungsvariable angeben.
    - Erstellen Sie einen Amazon S3-Gateway-VPC-Endpunkt anstelle eines Amazon S3-Schnittstellen-VPC-Endpunkts. S3-Gateway-Endpunkte unterstützen den Zugriff auf den globalen S3-Endpunkt. Weitere Informationen finden Sie unter Erstellen eines Gateway-Endpunkts.

Endpunkte und Ports

Diese Komponente muss in der Lage sein, ausgehende Anfragen an die folgenden Endpunkte und Ports auszuführen, zusätzlich zu den für den grundlegenden Betrieb erforderlichen Endpunkten und Ports. Weitere Informationen finden Sie unter Zulassen von Gerätedatenverkehr über einen Proxy oder eine Firewall.

Endpunkt	Port	Erforderlich	Beschreibung
`iotanalytics.region.amazonaws.com`	443	Nein	Erforderlich, wenn Sie Daten

Endpunkt	Port	Erforderlich	Beschreibung
			in veröffentlichen AWS IoT Analytics.
`kinesis.region.amazonaws.com`	443	Nein	Erforderlich, wenn Sie Daten in Firehose veröffentlichen.
`data.iotsitewise. region.amazonaws.com`	443	Nein	Erforderlich, wenn Sie Daten in veröffentlichen AWS IoT SiteWise.

Endpunkt	Port	Erforderlich	Beschreibung
`*.s3.amazonaws.com`	443	Nein	Erforderlich, wenn Sie Daten in S3-Buckets veröffentlichen. Sie können durch * den Namen jedes Buckets ersetzen, in dem Sie Daten veröffentlichen.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und alle ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitstellen zu können. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente und die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste Abhängigkeiten.

2.1.11

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.11 bis 2.1.10 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.13.0	Weich
Token-Exchange-Service	>=0.0.0	Hart

2.1.9 – 2.1.10

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.1.9 bis 2.1.10 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.12.0	Weich
Token-Exchange-Service	>=0.0.0	Hart

2.1.5 – 2.1.8

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.1.5 bis 2.1.8 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.11.0	Weich
Token-Exchange-Service	>=0.0.0	Hart

2.1.2 – 2.1.4

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.1.2 bis 2.1.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.10.0	Weich
Token-Exchange-Service	>=0.0.0	Hart

2.1.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.9.0	Weich
Token-Exchange-Service	>=0.0.0	Hart

2.1.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.8.0	Weich
Token-Exchange-Service	>=0.0.0	Hart

2.0.15

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.15 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.7.0	Weich
Token-Exchange-Service	>=0.0.0	Hart

2.0.13 and 2.0.14

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.0.13 und 2.0.14 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.6.0	Weich
Token-Exchange-Service	>=0.0.0	Hart

2.0.11 and 2.0.12

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.0.11 und 2.0.12 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.5.0	Weich
Token-Exchange-Service	>=0.0.0	Hart

2.0.10

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.10 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.4.0	Weich
Token-Exchange-Service	>=0.0.0	Hart

2.0.9

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.9 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.3.0	Weich
Token-Exchange-Service	>=0.0.0	Hart

2.0.8

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.8 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.0 <2.2.0	Weich
Token-Exchange-Service	>=0.0.0	Hart

2.0.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.0.3 <2.1.0	Weich
Token-Exchange-Service	>=0.0.0	Hart

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zum Komponentenrezept .

Konfiguration

Diese Komponente bietet die folgenden Konfigurationsparameter, die Sie bei der Bereitstellung der Komponente anpassen können.

`STREAM_MANAGER_STORE_ROOT_DIR`

(Optional) Der absolute Pfad des lokalen Verzeichnisses, das zum Speichern von Streams verwendet wird. Dieser Wert muss mit einem Schrägstrich (z. B. `/data`) beginnen.

Sie müssen einen vorhandenen Ordner angeben und der Systembenutzer, der die Stream-Manager-Komponente ausführt, muss über Lese- und Schreibberechtigungen für diesen Ordner verfügen. Sie können beispielsweise die folgenden Befehle ausführen, um einen Ordner, , zu erstellen und zu konfigurieren`/var/greengrass/streams`, den Sie als Stammordner des Stream-Managers angeben. Mit diesen Befehlen kann der Standardsystembenutzer, `ggc_user`, diesen Ordner lesen und in ihn schreiben.

```
sudo mkdir /var/greengrass/streams
sudo chown ggc_user /var/greengrass/streams
sudo chmod 700 /var/greengrass/streams
```

Standard: */greengrass/v2*`/work/aws.greengrass.StreamManager`

`STREAM_MANAGER_SERVER_PORT`

(Optional) Die lokale Portnummer, die für die Kommunikation mit dem Stream-Manager verwendet werden soll.

Sie können angeben`0`, um einen zufällig verfügbaren Port zu verwenden.

Standard: `8088`

`STREAM_MANAGER_AUTHENTICATE_CLIENT`

(Optional) Sie können es für Clients obligatorisch machen, sich zu authentifizieren, bevor sie mit dem Stream-Manager interagieren können. Das Stream Manager SDK steuert die Interaktion zwischen Clients und Stream Manager. Dieser Parameter bestimmt, welche Clients das Stream Manager SDK aufrufen können, um mit Streams zu arbeiten. Weitere Informationen finden Sie unter Stream-Manager-Client-Authentifizierung.

Wenn Sie angeben`true`, lässt das Stream Manager SDK nur Greengrass-Komponenten als Clients zu.

Wenn Sie angeben`false`, lässt das Stream Manager SDK zu, dass alle Prozesse auf dem Core-Gerät Clients sind.

Standard: `true`

`STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH`

(Optional) Die durchschnittliche maximale Bandbreite (in Kilobit pro Sekunde), die der Stream-Manager zum Exportieren von Daten verwenden kann.

Standard: keine Einschränkung

`STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE`

(Optional) Die maximale Anzahl aktiver Threads, die Stream Manager zum Exportieren von Daten verwenden kann.

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. Versuchen Sie daher, die Anzahl der Exportstreams auf eingeschränkten Geräten zu begrenzen.

Standard: 5 Threads

`STREAM_MANAGER_EXPORTER_S3_DESTINATION_MULTIPART_UPLOAD_MIN_PART_SIZE_BYTES`

(Optional) Die Mindestgröße (in Byte) eines Teils in einem mehrteiligen Upload in Amazon S3. Stream Manager verwendet diese Einstellung und die Größe der Eingabedatei, um zu bestimmen, wie Daten in einer mehrteiligen PUT-Anforderung gebündelt werden.

> **Note**
>
> Stream Manager verwendet die Streams-`sizeThresholdForMultipartUploadBytes`Eigenschaft, um zu bestimmen, ob als einteiliger oder mehrteiliger Upload nach Amazon S3 exportiert werden soll. AWS IoT Greengrass -Komponenten können diesen Schwellenwert festlegen, wenn sie einen Stream erstellen, der nach Amazon S3 exportiert wird.

Standard: `5242880` (5 MB). Dies ist auch der Mindestwert.

`LOG_LEVEL`

(Optional) Die Protokollierungsebene für die Komponente. Wählen Sie aus den folgenden Protokollebenen aus, die hier in der Reihenfolge der Ebenen aufgeführt sind:

- TRACE
- DEBUG
- INFO
- WARN
- ERROR

Standard: INFO

JVM_ARGS

(Optional) Die benutzerdefinierten Java Virtual Machine-Argumente, die beim Start an den Stream-Manager übergeben werden sollen. Trennen Sie mehrere Argumente durch Leerzeichen.

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.

Example Beispiel: Aktualisierung der Konfigurationszusammenführung

Die folgende Beispielkonfiguration legt fest, dass ein nicht standardmäßiger Port verwendet werden soll.

```
{
  "STREAM_MANAGER_SERVER_PORT": "18088"
}
```

Lokale Protokolldatei

Diese Komponente verwendet die folgende Protokolldatei.

Linux

```
/greengrass/v2/logs/aws.greengrass.StreamManager.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.StreamManager.log
```

So zeigen Sie die Protokolle dieser Komponente an

- Führen Sie den folgenden Befehl auf dem Core-Gerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

  Linux

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

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.StreamManager.log -Tail 10 -
  Wait
  ```

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in jeder Version der Komponente beschrieben.

Version	Änderungen
2.1.12	Fehlerbehebungen und Verbesserungen Aktualisiert die Reihenfolge, in der Anmeldeinformationen verwendet werden, sodass Greengrass-Anmeldeinformationen für AWS Servicean fragen bevorzugt werden.
2.1.11	Version für Greengrass-Kern Version 2.12.0 aktualisiert.
2.1.10	Fehlerbehebungen und Verbesserungen Behebt ein Problem, bei dem die HTTPS-Proxy-Konfiguration der Greengrass-Zertifizierungsstelle (CA)-Zertifikatkette nicht vertraut.
2.1.9	Version für Greengrass-Kern Version 2.11.0 aktualisiert.

Version	Änderungen
2.1.8	Fehlerbehebungen und Verbesserungen Behebt ein Problem, bei dem der Stream-Manager unendlich oft versucht, SiteWise Exporte mit zu wiederholen`InvalidRequestException` .
2.1.7	Fehlerbehebungen und Verbesserungen Behebt ein Problem, bei dem der Stream-Manager die Proxy-Konfiguration nicht korrekt liest.
2.1.6	Fehlerbehebungen und Verbesserungen Behebt ein Problem, das beim Start auf bestimmten ARMv8-Prozessoren, einschließlich Jetson Nano, zu einem Absturz führen konnte.
2.1.5	Version für Greengrass-Kernversion 2.10.0 aktualisiert.
2.1.4	Fehlerbehebungen und Verbesserungen • Behebt ein Problem, bei dem Einträge für dieselbe Eigenschaftskompon ente mit demselben Zeitstempel innerhalb eines einzigen Batches `ConflictingOperationException` von der SiteWise API zurückgegeben werden, wodurch der Stream-Manager kontinuierlich wiederholt wird. • Aktualisiert das Standard-Verbindungs-Timeout von 3 Sekunden auf 1 Minute.
2.1.3	Fehlerbehebungen und Verbesserungen Behebt ein Startup-Problem unter Windows OS, wenn es als SYSTEM-Be nutzer ausgeführt wird.
2.1.2	Fehlerbehebungen und Verbesserungen • Behebt ein Problem unter Windows OS, das eine nicht englische Sprache verwendet. • Version für Greengrass-Kern Version 2.9.0 aktualisiert.
2.1.1	Version für Greengrass-Kern Version 2.8.0 aktualisiert.

Version	Änderungen
2.1.0	Neue Features • Aktualisiert diese Komponente, um Telemetriemetriken automatisch an Amazon zu senden EventBridge. Weitere Informationen finden Sie unter Erfassen von Telemetriedaten zum Systemstatus von -AWS IoT GreengrassCore-Geräten. Diese Funktion erfordert v2.7.0 oder höher der Greengrass-Kernkomponente . • Version für Greengrass-Kern Version 2.7.0 aktualisiert.
2.0.15	Version für Greengrass-Kern Version 2.6.0 aktualisiert.
2.0.14	Diese Version enthält Fehlerbehebungen und Verbesserungen.
2.0.13	Version für Greengrass-Kern Version 2.5.0 aktualisiert.
2.0.12	Fehlerbehebungen und Verbesserungen Behebt ein Problem, das das Upgrade von Stream Manager v2.0.7 auf eine Version zwischen v2.0.8 und v2.0.11 verhinderte. Wenn Sie Stream Manager verwenden, um Daten in die Cloud zu exportieren, können Sie jetzt auf v2.0.12 aktualisieren.
2.0.11	Version für Greengrass-Kern Version 2.4.0 aktualisiert.
2.0.10	Version für Greengrass-Kern Version 2.3.0 aktualisiert.
2.0.9	Version für Greengrass-Kern Version 2.2.0 aktualisiert.
2.0.8	Version für Greengrass-Kern Version 2.1.0 aktualisiert.
2.0.7	Erste Version

Systemmanager-Agent

Die AWS Systems Manager Agent-Komponente (`aws.greengrass.SystemsManagerAgent`) installiert den Systems Manager Agent, sodass Sie Kerngeräte mit Systems Manager verwalten

können. Systems Manager ist ein AWS Service, mit dem Sie Ihre Infrastruktur anzeigen und steuern können AWS, einschließlich Amazon EC2 EC2-Instances, lokalen Servern und virtuellen Maschinen (VMs) sowie Edge-Geräten. Mit Systems Manager können Sie Betriebsdaten anzeigen, Betriebsaufgaben automatisieren und Sicherheit und Compliance gewährleisten. Weitere Informationen finden Sie unter Was ist AWS Systems Manager? und Über Systems Manager Agent im AWS Systems Manager Benutzerhandbuch.

Die Tools und Funktionen von Systems Manager werden Funktionen genannt. Greengrass-Kerngeräte unterstützen alle Systems Manager Manager-Funktionen. Weitere Informationen zu diesen Funktionen und zur Verwendung von Systems Manager zur Verwaltung von Kerngeräten finden Sie unter Systems Manager Manager-Funktionen im AWS Systems Manager Benutzerhandbuch.

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 1.1.x
- 1.0.x

Typ

Diese Komponente ist eine generische Komponente (`aws.greengrass.generic`). Der Greengrass-Kern führt die Lebenszyklusskripte der Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann nur auf Linux-Core-Geräten installiert werden.

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Ein Greengrass-Core-Gerät, das auf einer 64-Bit-Linux-Plattform läuft: Armv8 (AArch64) oder x86_64.
- Sie müssen über eine AWS Identity and Access Management (IAM-) Servicerolle verfügen, die Systems Manager übernehmen kann. Diese Rolle muss die von AmazonSSM ManagedInstanceCore verwaltete Richtlinie oder eine benutzerdefinierte Richtlinie enthalten, die entsprechende Berechtigungen definiert. Weitere Informationen finden Sie im Benutzerhandbuch unter Erstellen einer IAM-Servicerolle für Edge-Geräte.AWS Systems Manager

  Wenn Sie diese Komponente bereitstellen, müssen Sie den Namen dieser Rolle für den `SSMRegistrationRole` Konfigurationsparameter angeben.
- Die Greengrass-Geräterolle muss die `ssm:RegisterManagedInstance` Aktionen `ssm:AddTagsToResource` und zulassen. Die Geräterolle muss auch die `iam:PassRole` Aktion für die IAM-Servicerolle zulassen, die die vorherige Anforderung erfüllt. Die folgende Beispiel-IAM-Richtlinie gewährt diese Berechtigungen.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "iam:PassRole"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:iam::account-id:role/SSMServiceRole"
      ]
    },
    {
      "Action": [
        "ssm:AddTagsToResource",
        "ssm:RegisterManagedInstance"
```

```
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

Endpunkte und Ports

Diese Komponente muss in der Lage sein, ausgehende Anfragen an die folgenden Endpunkte und Ports auszuführen, zusätzlich zu den Endpunkten und Ports, die für den Basisbetrieb erforderlich sind. Weitere Informationen finden Sie unter Zulassen von Gerätedatenverkehr über einen Proxy oder eine Firewall.

Endpunkt	Port	Erforderlich	Beschreibung
`ec2messages.region.amazonaws.com`	443	Ja	Kommunizieren Sie mit dem Systems Manager Manager-Dienst in der AWS Cloud.
`ssm.region.amazonaws.com`	443	Ja	Registrieren Sie das Kerngerät als von Systems Manager verwalteten Knoten.
`ssmmessages.region.amazonaws.com`	443	Ja	Kommunizieren

Endpunkt	Port	Erforderlich	Beschreibung
			Sie mit Session Manager, einer Funktion von Systems Manager, in der AWS Cloud.

Weitere Informationen finden Sie unter Referenz: ec2messages, ssmmessages und andere API-Aufrufe im Benutzerhandbuch.AWS Systems Manager

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt sie AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und all ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitstellen zu können. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente sowie die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste der Abhängigkeiten.

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 1.0.0 bis 1.2.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Token-Austauschdienst	^2.0.0	Weich

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zu den Komponentenrezepten.

Konfiguration

Diese Komponente stellt die folgenden Konfigurationsparameter bereit, die Sie bei der Bereitstellung der Komponente anpassen können.

`SSMRegistrationRole`

Die IAM-Servicerolle, die Systems Manager übernehmen kann und die die von AmazonSSM ManagedInstanceCore verwaltete Richtlinie oder eine benutzerdefinierte Richtlinie umfasst, die entsprechende Berechtigungen definiert. Weitere Informationen finden Sie im Benutzerhandbuch unter Erstellen einer IAM-Servicerolle für Edge-Geräte.AWS Systems Manager

`SSMOverrideExistingRegistration`

(Optional) Wenn auf dem Kerngerät bereits der mit einer Hybrid-Aktivierung registrierte Systems Manager Agent ausgeführt wird, können Sie die bestehende Systems Manager Agent-Registrierung des Geräts überschreiben. Stellen Sie diese Option auf ein`true`, um das Kerngerät mithilfe des Systems Manager Agents, den diese Komponente bereitstellt, als verwalteten Knoten zu registrieren.

> **Note**
>
> Diese Option gilt nur für Geräte, die mit einer Hybridaktivierung registriert wurden. Wenn das Kerngerät auf einer Amazon EC2 EC2-Instance läuft, auf der der Systems Manager Agent installiert und eine Instance-Profilrolle konfiguriert ist, beginnt die bestehende verwaltete Knoten-ID der Amazon EC2 EC2-Instance mit. `i-` Wenn Sie die Systems Manager Agent-Komponente installieren, registriert der Systems Manager Agent einen neuen verwalteten Knoten, dessen ID mit `mi-` statt mit beginnt`i-`. Anschließend können Sie den verwalteten Knoten, dessen ID mit beginnt, verwenden`mi-`, um das Kerngerät mit Systems Manager zu verwalten.

Standard: `false`

`SSMResourceTags`

(Optional) Die Tags, die dem verwalteten Systems Manager Manager-Knoten hinzugefügt werden sollen, den diese Komponente für das Kerngerät erstellt. Sie können diese Tags verwenden, um Gruppen von Kerngeräten mit Systems Manager zu verwalten. Sie können beispielsweise einen Befehl auf allen Geräten ausführen, die über ein von Ihnen angegebenes Tag verfügen.

Geben Sie eine Liste an, in der jedes Tag ein Objekt mit einem `Key` und a ist`Value`. Der folgende Wert für `SSMResourceTags` weist diese Komponente beispielsweise an, das **`Owner`** Tag auf dem verwalteten Knoten des Kerngeräts **`richard-roe`** auf zu setzen.

```
[
  {
    "Key": "Owner",
    "Value": "richard-roe"
  }
]
```

Diese Komponente ignoriert diese Tags, wenn der verwaltete Knoten bereits existiert und `SSMOverrideExistingRegistration` existiert. `false`

Example Beispiel: Aktualisierung der Konfigurationszusammenführung

In der folgenden Beispielkonfiguration wird die Verwendung einer Servicerolle mit `SSMServiceRole` dem Namen so angegeben, dass sich das Kerngerät registrieren und mit Systems Manager kommunizieren kann.

```
{
  "SSMRegistrationRole": "SSMServiceRole",
  "SSMOverrideExistingRegistration": false,
  "SSMResourceTags": [
    {
      "Key": "Owner",
      "Value": "richard-roe"
    },
    {
      "Key": "Team",
      "Value": "solar"
    }
  ]
}
```

Lokale Protokolldatei

Die Systems Manager Agent-Software schreibt Protokolle in einen Ordner außerhalb des Greengrass-Stammordners. Weitere Informationen finden Sie im AWS Systems Manager Benutzerhandbuch unter Anzeigen von Systems Manager Agent-Protokollen.

Die Systems Manager Agent-Komponente verwendet Shell-Skripts, um den Systems Manager Agent zu installieren, zu starten und zu beenden. Die Ausgabe dieser Skripts finden Sie in der folgenden Protokolldatei.

```
/greengrass/v2/logs/aws.greengrass.SystemsManagerAgent.log
```

Um die Protokolle dieser Komponente einzusehen

- Führen Sie den folgenden Befehl auf dem Kerngerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. */greengrass/v2*Ersetzen Sie durch den Pfad zum AWS IoT Greengrass Stammordner.

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

Weitere Informationen finden Sie auch unter

- Verwalten von Greengrass-Core-Geräten mit AWS Systems Manager
- Was ist AWS Systems Manager? im AWS Systems Manager Benutzerhandbuch
- Informationen zum Systems Manager Agent im AWS Systems Manager Benutzerhandbuch

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
1.2.4	Fehlerkorrekturen und Verbesserungen Aktualisiert diese Komponente auf die Agent-Version 3.2.2303.0.
1.2.3	Fehlerkorrekturen und Verbesserungen • Fügt Wiederholungsversuche für die Installation der Agent-Komponente mit Snap on Greengrass hinzu. • Aktualisiert die Konfiguration der Agent-Komponente, sodass sie nur die Onprem Identity in Greengrass verwendet.

Version	Änderungen
	• Aktualisiert diese Komponente, sodass der Agent nur aktualisiert wird, wenn die installierte Agent-Version nicht mit der Version der Greengrass SSM Agent-Komponente übereinstimmt.
1.1.0	Diese Version enthält Fehlerkorrekturen und Verbesserungen.
1.0.0	Erste Version

Token-Exchange-Service

Die Token-Exchange-Servicekomponente (`aws.greengrass.TokenExchangeService`) stellt AWS Anmeldeinformationen bereit, mit denen Sie mit AWS Services in Ihren benutzerdefinierten Komponenten interagieren können.

Der Token-Exchange-Service führt eine Amazon Elastic Container Service (Amazon ECS)-Container-Instance als lokalen Server aus. Dieser lokale Server stellt mithilfe des AWS IoT Rollenalias, den Sie in der Greengrass-Kernkernkomponente konfigurieren, eine Verbindung zum AWS IoT Anmeldeinformationsanbieter her. Die Komponente bietet zwei Umgebungsvariablen, `AWS_CONTAINER_CREDENTIALS_FULL_URI` und `AWS_CONTAINER_AUTHORIZATION_TOKEN`. `AWS_CONTAINER_CREDENTIALS_FULL_URI` definiert den URI für diesen lokalen Server. Wenn eine Komponente einen AWS SDK-Client erstellt, erkennt der Client diese URI-Umgebungsvariable und verwendet das Token in der `AWS_CONTAINER_AUTHORIZATION_TOKEN`, um eine Verbindung zum Token-Exchange-Service herzustellen und AWS Anmeldeinformationen abzurufen. Auf diese Weise können Greengrass-Core-Geräte -AWSService-Operationen aufrufen. Weitere Informationen zur Verwendung dieser Komponente in benutzerdefinierten Komponenten finden Sie unter Interagieren mit -AWSServices.

> **Important**
>
> Unterstützung für den Erhalt von AWS Anmeldeinformationen auf diese Weise wurde den AWS -SDKs am 13. Juli 2016 hinzugefügt. Ihre Komponente muss eine AWS SDK-Version verwenden, die an oder nach diesem Datum erstellt wurde. Weitere Informationen finden Sie unter Verwenden eines unterstützten AWS SDK im Amazon Elastic Container Service-Entwicklerhandbuch.

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.0.x

Typ

Diese Komponente ist eine generische Komponente (`aws.greengrass.generic`). Der Greengrass-Kern führt die Lebenszyklusskripte der Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf -Core-Geräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Abhängigkeiten

Diese Komponente hat keine Abhängigkeiten.

Konfiguration

Diese Komponente hat keine Konfigurationsparameter.

Lokale Protokolldatei

Diese Komponente verwendet dieselbe Protokolldatei wie die Greengrass-Kernkomponente.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

So zeigen Sie die Protokolle dieser Komponente an

- Führen Sie den folgenden Befehl auf dem Core-Gerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

  Linux

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

  Windows (PowerShell)

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

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in jeder Version der Komponente beschrieben.

Version	Änderungen
2.0.3	Erste Version

IoT SiteWise -OPC-UA-Kollektor

Die IoT SiteWise -OPC-UA-Kollektorkomponente (`aws.iot.SiteWiseEdgeCollectorOpcua`) ermöglicht es AWS IoT SiteWise Gateways, Daten von lokalen OPC-UA-Servern zu sammeln.

Mit dieser Komponente können AWS IoT SiteWise Gateways eine Verbindung zu mehreren OPC-UA-Servern herstellen. Weitere Informationen zu AWS IoT SiteWise Gateways finden Sie unter Verwenden von AWS IoT SiteWise am Edge im AWS IoT SiteWise -Benutzerhandbuch.

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.4.x
- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Typ

Diese Komponente ist eine generische Komponente (`aws.greengrass.generic`). Der Greengrass-Kern führt die Lebenszyklusskripte der Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf -Core-Geräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Das Greengrass-Core-Gerät muss auf einer der folgenden Plattformen ausgeführt werden:
  - Betriebssystem: Ubuntu 18.04 oder höher

    Architektur: x86_64 (AMD64) oder ARMv8 (Aarch64)
  - Betriebssystem: Red Hat Enterprise Linux (RHEL) 8

    Architektur: x86_64 (AMD64) oder ARMv8 (Aarch64)
  - Betriebssystem: Amazon Linux 2

    Architektur: x86_64 (AMD64) oder ARMv8 (Aarch64)
  - Betriebssystem: Debian 11

    Architektur: x86_64 (AMD64) oder ARMv8 (Aarch64)
  - Betriebssystem: Windows Server 2019 oder höher

    Architektur: x86_64 (AMD64)
- Das Greengrass-Core-Gerät muss ausgehende Netzwerkkonnektivität zu OPC-UA-Servern zulassen.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und alle ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitzustellen. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente und die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste Abhängigkeiten.

In der folgenden Tabelle sind die Abhängigkeiten für alle Versionen dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.3.0 <3.0.0	Hart
Stream-Manager	>2.0.10<3.0.0	Hart
Geheimer Manager	>=2.0.8 <3.0.0	Hart

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zum Komponentenrezept .

Konfiguration

Diese Komponente hat keine Konfigurationsparameter.

Sie können die AWS IoT SiteWise Konsole oder API verwenden, um die IoT SiteWise -OPC-UA-Kollektorkomponente zu konfigurieren. Weitere Informationen finden Sie unter Schritt 4: Hinzufügen von Datenquellen – optional im AWS IoT SiteWise -Benutzerhandbuch.

Eingabedaten

Diese Komponente akzeptiert nur Daten in den folgenden Formaten, alle anderen werden ignoriert und verworfen. Die folgende Tabelle ordnet die OPC-UA-Datentypen ihrem SiteWise Äquivalent zu.

SiteWise Datentyp	OPC-UA-Datentyp	Beschreibung

`STRING`	`String`
	`Guid`
	`XmlElement`
		Eine Zeichenfolge mit einer maximalen Länge von 1024 Byte.

`INTEGER`	`SByte`
	`Byte`
	`Int16`
	`UInt16`
	`Int32`
	`UInt32*`
	`Int64*`
		Eine signierte 32-Bit-Ganzzahl mit einem Bereich von `-2,147,483,648 to 2,147,483,647` .

`DOUBLE`	`UInt32*`
	`Int64*`
	`Float`
	`Double`
		Eine Gleitkommazahl mit einem Bereich von `-10^100 to 10^100` und `IEEE 754` doppelter Genauigkeit.

`BOOLEAN`	`Boolean`	`true` oder `false`.

* Für OPC-UA-Datentypen `UInt32` und lautet der SiteWise Datentyp `Int64`, `INTEGER` wenn seinen Wert darstellen SiteWise kann, andernfalls ist er `DOUBLE`.

Ausgabedaten

Diese Komponente schreibt `BatchPutAssetPropertyValue` Nachrichten in den AWS IoT Greengrass Stream-Manager. Weitere Informationen finden Sie unter BatchPutAssetPropertyValue in der AWS IoT SiteWise -API-Referenz.

Lokale Protokolldatei

Diese Komponente verwendet die folgende Protokolldatei.

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgeCollectorOpcua.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeCollectorOpcua.log
```

So zeigen Sie die Protokolle dieser Komponente an

- Führen Sie den folgenden Befehl auf dem Core-Gerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.iot.SiteWiseEdgeCollectorOpcua.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeCollectorOpcua.log -Tail
   10 -Wait
  ```

Fehlerbehebung und Debugging

Diese Komponente enthält ein neues Ereignisprotokoll, das Kunden hilft, Probleme zu identifizieren und zu beheben. Die Protokolldatei ist von der lokalen Protokolldatei getrennt und befindet sich am folgenden Speicherort. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

Linux

```
/greengrass/v2/work/aws.iot.SiteWiseEdgeCollectorOpcua/logs/
IotSiteWiseOpcUaCollectorEvents.log
```

Windows

```
C:\greengrass\v2\work\aws.iot.SiteWiseEdgeCollectorOpcua\logs
\IotSiteWiseOpcUaCollectorEvents.log
```

Dieses Protokoll enthält detaillierte Informationen und Anweisungen zur Fehlerbehebung. Informationen zur Fehlerbehebung werden zusammen mit der Diagnose bereitgestellt, mit einer Beschreibung, wie das Problem behoben werden kann, und manchmal mit Links zu weiteren Informationen. Zu den Diagnoseinformationen gehören:

- Schweregrad
- Zeitstempel
- Zusätzliche ereignisspezifische Informationen

Example Beispielprotokoll

```
dataSourceConnectionSuccess:
  Summary: Successfully connected to OpcUa server
  Level: INFO
  Timestamp: '2023-06-15T21:04:16.303Z'
  Description: Successfully connected to the data source.
  AssociatedMetrics:
  - Name: FetchedDataStreams
    Description: The number of fetched data streams for this data source
    Value: 1.0
    Namespace: IoTSiteWise
    Dimensions:
    - Name: SourceName
      Value: SourceName{value=OPC-UA Server}
    - Name: ThingName
      Value: test-core
  AssociatedData:
  - Name: DataSourceTrace
    Description: Name of the data source
    Data:
    - OPC-UA Server
  - Name: EndpointUri
    Description: The endpoint to which the connection was attempted.
    Data:
```

```
    - '"opc.tcp://10.0.0.1:1234"'
```

Lizenzen

Diese Komponente wird gemäß dem Greengrass Core Software License Agreement veröffentlicht.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in jeder Version der Komponente beschrieben.

Version	Änderungen
2.4.2	Fehlerbehebungen und Verbesserungen • Behebt Probleme bei der Erkennung des OPC-UA-Servers, bei denen ein Knoten mehrmals erkannt werden kann. • Korrigiert die Snapshot-Funktion, um sicherzustellen, dass der Zeitstempel für jeden Snapshot-Datenpunkt neu ist.
2.4.1	Fehlerbehebungen und Verbesserungen • Behebt Probleme im Zusammenhang mit der Proxy-Unterstützung. • Behebt ein Problem, bei dem die Thread-Bereinigung fehlgeschlagen ist und eine Datenblockierung verursacht hat.
2.4.0	Neue Features • Fügt ein Ereignisprotokoll hinzu, um das Identifizieren und Beheben von Problemen zu erleichtern. Fehlerbehebungen und Verbesserungen • Behebt ein Problem mit dem OPC-UA-Client, das beim Herstellen einer Verbindung mit einem OPC-UA-Server, der Version 1.05 der OPC-UA-Spezifikation verwendet, Zertifikatsfehler verursacht hat.
2.3.0	Neue Features • Fügt Unterstützung für die Greengrass-Kern-HTTP-Proxy-Konfiguration unter Linux hinzu. Fehlerbehebungen und Verbesserungen • Behebt ein Sicherheitsproblem (CVE-2019-19135).

Version	Änderungen

2.2.0	Neue Features
	• Fügt Unterstützung für die Installation von Data Collection Pack auf der Linux ARMv8-Architektur hinzu.
	• Mindestanforderungen für Linux ARMv8:
	  • Arbeitsspeicher: 4 GB
	  • CPU: ARM Cortex-A72 oder gleichwertige Spezifikation
	
	Fehlerbehebungen und Verbesserungen
	• Verbessert die Protokollierung von Metriken im Knotenerkennungspr ozess.
	• Verbessert den Umgang mit nicht unterstützten Datentypen.
	• Verbessert die Protokollierung von Datenstromfehlern.

2.1.3	Neue Features • Fügt Unterstützung für Windows Server 2019 oder höher hinzu. Fehlerbehebungen und Verbesserungen • Verbessert Fehlermeldungen, wenn Sie diese Komponente auf nicht unterstützten Geräten bereitstellen.

Version	Änderungen

2.1.1	Neue Features
	• Fügt Unterstützung für die Konfiguration der folgenden Abonnemen teigenschaften hinzu:
	  • DataChangeTrigger – Sie können die Bedingung definieren, die eine Warnung über Datenänderungen auslöst.
	  • QueueSize – Die Tiefe der Warteschlange auf einem OPC-UA-Server für eine bestimmte Metrik, bei der Benachrichtigungen für überwachte Elemente in die Warteschlange gestellt werden.
	  • PublishingIntervalMilliseconds – Das Intervall (in Millisekunden) eines Veröffentlichungszyklus, das bei der Erstellung eines Abonnements angegeben wird.
	  • SnapshotFrequencyMilliseconds – Sie können die Einstellung für das Snapshot-Frequenz-Timeout konfigurieren, um sicherzustellen, dass AWS IoT SiteWise Edge einen konstanten Datenstrom aufnimmt.
	• Diese Version unterstützt die Aufnahme von `BAD` Qualitätsdaten und filtert Daten basierend auf den folgenden Datenqualitäten:
	  • `UNCERTAIN` Qualitätsdaten
	  • `BAD` Qualitätsdaten
	Fehlerbehebungen und Verbesserungen
	• Verbesserungen der Kundenmetriken.
	• Behebt die Sicherheitskodierung, die manchmal Probleme verursach te, wenn eine Verbindung zu Servern mit aktivierter Verschlüsselung hergestellt wurde.
	• Behebt ein Problem, bei dem die Eigenschaftsgruppe nicht aktualisiert werden konnte.

2.0.3	Fehlerbehebungen und Verbesserungen.
2.0.2	Fehlerbehebungen und Verbesserungen der Synchronisierung von Komponentenprioritäten mit Edge.
2.0.1	Erste Version

Weitere Informationen finden Sie auch unter

- Was ist AWS IoT SiteWise? im AWS IoT SiteWise -Benutzerhandbuch.

IoT SiteWise OPC-UA-Datenquellensimulator

Die IoT SiteWise OPC-UA-Datenquellensimulatorkomponente (`aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator`) startet einen lokalen OPC-UA-Server, der Beispieldaten generiert. Verwenden Sie diesen OPC-UA-Server, um eine Datenquelle zu simulieren, die von der IoT SiteWise -OPC-UA-Kollektorkomponente auf einem -AWS IoT SiteWiseGateway gelesen wird. Anschließend können Sie anhand dieser Beispieldaten AWS IoT SiteWise Funktionen erkunden. Weitere Informationen zu AWS IoT SiteWise Gateways finden Sie unter Verwenden von AWS IoT SiteWise am Edge im AWS IoT SiteWise -Benutzerhandbuch.

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 1.0.x

Typ

Diese Komponente ist eine generische Komponente (`aws.greengrass.generic`). Der Greengrass-Kern führt die Lebenszyklusskripte der Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf -Core-Geräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Das Greengrass-Core-Gerät muss Port 4840 auf dem lokalen Host verwenden können. Der lokale OPC-UA-Server dieser Komponente wird an diesem Port ausgeführt.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und alle ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitzustellen. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente und die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste Abhängigkeiten.

In der folgenden Tabelle sind die Abhängigkeiten für alle Versionen dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Abhängigkeitstyp
Greengrass-Kern	>=2.3.0 <3.0.0	Hart

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zum Komponentenrezept .

Konfiguration

Diese Komponente hat keine Konfigurationsparameter.

Lokale Protokolldatei

Diese Komponente verwendet die folgende Protokolldatei.

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.log
```

So zeigen Sie die Protokolle dieser Komponente an

- Führen Sie den folgenden Befehl auf dem Core-Gerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/
  aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs
  \aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.log -Tail 10 -Wait
  ```

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in jeder Version der Komponente beschrieben.

Version	Änderungen
1.0.0	Erste Version

Version	Änderungen
	Fügt Unterstützung für Windows Server 2016 oder höher hinzu.

Weitere Informationen finden Sie auch unter

- Was ist AWS IoT SiteWise? im AWS IoT SiteWise -Benutzerhandbuch.

SiteWise IoT-Herausgeber

Die SiteWise IoT-Publisher-Komponente (`aws.iot.SiteWiseEdgePublisher`) ermöglicht es AWS IoT SiteWise Gateways, Daten vom Edge zum AWS Cloud zu exportieren.

Weitere Informationen zu AWS IoT SiteWise Gateways finden Sie im AWS IoT SiteWise Benutzerhandbuch unter AWS IoT SiteWise Using at the Edge.

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 3.1.x

- 3.0.x
- 2.4.x
- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Typ

Diese Komponente ist eine generische Komponente (`aws.greengrass.generic`). Der Greengrass-Kern führt die Lebenszyklusskripte der Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf Kerngeräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Das Greengrass-Core-Gerät muss auf einer der folgenden Plattformen laufen:
  - Betriebssystem: Ubuntu 18.04 oder höher

    Architektur: x86_64 (AMD64) oder ARMv8 (Aarch64)
  - Betriebssystem: Red Hat Enterprise Linux (RHEL) 8

    Architektur: x86_64 (AMD64) oder ARMv8 (Aarch64)
  - Betriebssystem: Amazon Linux 2

    Architektur: x86_64 (AMD64) oder ARMv8 (Aarch64)
  - Betriebssystem: Debian 11

Architektur: x86_64 (AMD64) oder ARMv8 (Aarch64)

- Betriebssystem: Windows Server 2019 oder höher

  Architektur: x86_64 (AMD64)

- Das Greengrass Core-Gerät muss eine Verbindung zum Internet herstellen.
- Das Greengrass Core-Gerät muss für die Ausführung der `iotsitewise:BatchPutAssetPropertyValue` Aktion autorisiert sein. Weitere Informationen finden Sie unter Autorisieren von Kerngeräten für die Interaktion mit AWS Diensten.

  Example Berechtigungsrichtlinie

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

Endpunkte und Ports

Diese Komponente muss in der Lage sein, ausgehende Anfragen an die folgenden Endpunkte und Ports auszuführen, zusätzlich zu den Endpunkten und Ports, die für den Basisbetrieb erforderlich sind. Weitere Informationen finden Sie unter Zulassen von Gerätedatenverkehr über einen Proxy oder eine Firewall.

Endpunkt	Port	Erforderlich	Beschreibung
`data.iotsitewise.` *`region`*`.amazonaws.com`	443	Ja	Daten veröffentlichen in. AWS IoT SiteWise

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt sie AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und all ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitstellen zu können. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente sowie die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können auch die Abhängigkeiten für jede Version der Komponente in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste der Abhängigkeiten.

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.0.x bis 2.2.x dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.3.0<3.0.0	Hart
Stream-Manager	>=2.3.0 =2.0.10<3.0.0	Hart

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zu den Komponentenrezepten.

Konfiguration

Diese Komponente hat keine Konfigurationsparameter.

Sie können die AWS IoT SiteWise Konsole oder API verwenden, um die SiteWise IoT-Publisher-Komponente zu konfigurieren. Weitere Informationen finden Sie unter Schritt 3: Publisher konfigurieren — optional im AWS IoT SiteWise Benutzerhandbuch.

Eingabedaten

Diese Komponente liest `PutAssetPropertyValueEntry` Nachrichten aus dem AWS IoT Greengrass Stream-Manager. Weitere Informationen finden Sie PutAssetPropertyValueEntryin der AWS IoT SiteWise API-Referenz.

Lokale Protokolldatei

Diese Komponente verwendet die folgende Protokolldatei.

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgePublisher.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgePublisher.log
```

Um die Protokolle dieser Komponente einzusehen

- Führen Sie den folgenden Befehl auf dem Kerngerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.iot.SiteWiseEdgePublisher.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.iot.SiteWiseEdgePublisher.log -Tail 10 -
  Wait
  ```

Problembehandlung und Debugging

Diese Komponente enthält ein neues Ereignisprotokoll, mit dem Kunden Probleme identifizieren und beheben können. Die Protokolldatei ist von der lokalen Protokolldatei getrennt und befindet sich am folgenden Speicherort. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

Linux

```
/greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/logs/
IotSiteWisePublisherEvents.log
```

Windows

```
C:\greengrass\v2\work\aws.iot.SiteWiseEdgePublisher\logs
\IotSiteWisePublisherEvents.log
```

Dieses Protokoll enthält detaillierte Informationen und Anweisungen zur Fehlerbehebung. Informationen zur Fehlerbehebung werden zusammen mit der Diagnose bereitgestellt. Sie enthalten eine Beschreibung, wie das Problem behoben werden kann, und manchmal auch Links zu weiteren Informationen. Zu den Diagnoseinformationen gehören die folgenden:

- Schweregrad
- Zeitstempel
- Zusätzliche ereignisspezifische Informationen

Example Beispielprotokoll

```
accountBeingThrottled:
  Summary: Data upload speed slowed due to quota limits
  Level: WARN
  Timestamp: '2023-06-09T21:30:24.654Z'
  Description: The IoT SiteWise Publisher is limited to the "Rate of data points
 ingested"
    quota for a customers account. See the associated documentation and associated
    metric for the number of requests that were limited for more information. Note
    that this may be temporary and not require any change, although if the issue
 continues
    you may need to request an increase for the mentioned quota.
  FurtherInformation:
  - https://docs.aws.amazon.com/iot-sitewise/latest/userguide/quotas.html
  - https://docs.aws.amazon.com/iot-sitewise/latest/userguide/troubleshooting-
gateway.html#gateway-issue-data-streams
  AssociatedMetrics:
  - Name: TotalErrorCount
    Description: The total number of errors of this type that occurred.
    Value: 327724.0
  AssociatedData:
  - Name: AggregatePropertyAliases
    Description: The aggregated property aliases of the throttled data.
```

```
    FileLocation: /greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/./logs/data/
AggregatePropertyAliases_1686346224654.log
```

Lizenzen

Diese Komponente wird im Rahmen der Greengrass Core Software-Lizenzvereinbarung veröffentlicht.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
3.1.3	Fehlerkorrekturen und Verbesserungen • Es wurde ein Problem behoben, bei dem die Ereignisprotokolldatei unter erstellt `/greengrass/v2/work/aws.iot.SiteWise EdgePublisher/logs/IoTSiteWisePublisherEvents .log` wurde, aber keine Ereignisse protokolliert wurden. • Die folgenden CloudWatch Metriken zur Überwachung der Verbindung mit dem MQTT-Broker wurden hinzugefügt: • `IoTSiteWisePublisher.IsConnectedToMqttBroker` • `IoTSiteWisePublisher.NumberOfSubscriptionsToM qttBroker` • `IoTSiteWisePublisher.NumberOfUniqueMqttTopics Received` • `IoTSiteWisePublisher.MqttMessageReceivedSucce ssCount` • `IoTSiteWisePublisher.MqttReceivedSuccessBytes` Weitere Informationen zu diesen Metriken finden Sie unter AWS IoT Greengrass Version 2 Gateway-Metriken. • Es wurde ein Problem behoben, bei dem die `BatchCreateJob` API auch dann weiterhin aufgerufen wurde, wenn das Hochladen einer Parquet-Datei auf S3 fehlschlug.

Version	Änderungen
3.1.2	Fehlerkorrekturen und Verbesserungen • Das in Version 3.1.1 eingeführte Problem der hohen CPU-Auslastung wurde behoben.
3.1.1	Fehlerkorrekturen und Verbesserungen • Fügt zusätzliche Protokollierung hinzu, die die betroffenen Datenaliase identifiziert, wenn ein Fehler auftritt. • Fügt die lokale Durchsetzung von AWS IoT SiteWise API-Grenzwerten für das Alter der aufgenommenen Daten hinzu. • Behebt das Problem, dass Publisher die Checkpoints der StreamMan ager Streams durcheinander bringt, wenn es mehrere Amazon S3 S3-Ziele gibt. • Behebt Leistungsengpässe bei der Art und Weise, wie der Publisher aus den Streams liest. StreamManager
3.1.0	Neue Features • Integriert die Unterstützung für die Veröffentlichung von Daten als Parquet-Dateien in Amazon S3. • Fügt Unterstützung für AWS IoT SiteWise gepufferte Aufnahme hinzu.
3.0.0	Fehlerkorrekturen und Verbesserungen • Behebt Probleme im Zusammenhang mit der Proxyunterstützung. Neue Features • Aktiviert die Unterstützung der Datenaufnahme durch einen MQTT-Brok er.
2.4.1	Fehlerkorrekturen und Verbesserungen • Aktivieren Sie die Komponente so, dass sie mit Java Corretto 11 Versionen 11.0.20.8.1 und höher funktioniert. Die Komponentenversion en 2.4.0 und 2.3.3 zeigen die `"Could not find or load main class"` Fehlermeldung an, wenn sie mit Java Corretto Version 11.0.20.8.1 verwendet werden.

Version	Änderungen
2.4.0	Neue Features • Fügt ein neues Ereignisprotokoll hinzu, um Probleme leichter identifiz ieren und beheben zu können. Fehlerkorrekturen und Verbesserungen • Verbessert die Publisher-Checkpoint-Wiederherstellung.
2.3.3	Fehlerkorrekturen und Verbesserungen • Verbessert die Fähigkeit, hohen Durchsatz zu unterstützen.
2.3.2	Fehlerkorrekturen und Verbesserungen • Behebt die HTTP-Proxyunterstützung beim Herunterladen der Publisher -Konfiguration.
2.3.1	Neue Features • Fügt Unterstützung für die Installation von Data Collection Pack auf der Linux ARMv8-Architektur hinzu. • Mindestanforderungen für Linux ARMv8: • Arbeitsspeicher: 4 GB • CPU: ARM Cortex-A72 oder gleichwertige Spezifikation
2.2.3	Fehlerkorrekturen und Verbesserungen • Entfernt den Wiederholungsversuch für eine generische Ausnahme, die nicht in der Liste der abrufbaren Ausnahmen enthalten war.
2.2.2	Fehlerkorrekturen und Verbesserungen • Führt die Unterstützung für das Hochladen von Daten AWS IoT SiteWise über einen HTTP-Proxyserver wieder ein.

Version	Änderungen
2.2.1	**Note** Diese Version unterstützt keine HTTP-Proxykonfiguration. Version 2.2.2 und höher führt die Unterstützung für diese Funktion wieder ein. Neue Features • Fügt dieser Komponente Unterstützung hinzu, um die Komprimierung beim Hochladen von Daten umzuschalten. AWS IoT SiteWise

2.2.0	**Note**
	Diese Version unterstützt keine HTTP-Proxykonfiguration. Version 2.2.2 und höher führt die Unterstützung für diese Funktion wieder ein.
	
	Neue Features
	• Aktualisiert diese Komponente, um Daten zu komprimieren, bevor sie an den Dienst gesendet werden. AWS IoT SiteWise
	  • In den meisten Fällen reduziert diese Änderung die Bandbreit ennutzung im Vergleich zu früheren Versionen dieser Komponente um 75 Prozent.
	  • In den meisten Fällen erhöht diese Änderung die CPU-Auslastung um bis zu 5 Prozent. Auf Gateways, die große Datenmengen verarbeit en, kann diese Änderung die CPU-Auslastung um bis zu 15 Prozent erhöhen.
	  • Diese Änderung wirkt sich nicht auf die AWS IoT SiteWise Servicege bühren oder die Nutzung der Servicekontingenten aus.
	  • Fügt Unterstützung für Windows Server 2019 oder höher hinzu.
	Fehlerkorrekturen und Verbesserungen
	• Behebt ein Problem, das verhindert, dass diese Komponente gestartet wird, wenn die Checkpoint-Datei beschädigt ist.

Version	Änderungen
2.1.4	Fehlerkorrekturen und Verbesserungen • Behebt die Kompatibilität mit Java-Version 8.
2.1.3	⚠ Warning Diese Version ist nicht mehr verfügbar, außer in den Regionen USA Ost (Ohio), Kanada (Mitte) und AWS GovCloud (USA Ost). Für die Ausführung dieser Komponentenversion ist Java-Version 11 oder höher erforderlich. Die Verbesserungen in dieser Version sind in späteren Versionen dieser Komponente verfügbar. Fehlerkorrekturen und Verbesserungen • Verbessert die Fehlermeldungen, wenn Sie diese Komponente auf nicht unterstützten Geräten bereitstellen. • Aktualisierungen zur Protokollierung von Fehlern, wenn Datenuploads fehlschlagen.
2.1.2	Fehlerkorrekturen und Verbesserungen • Updates, um die Exportfunktion für abgelaufene Daten aufzurufen, sobald Daten ablaufen.
2.1.1	Fehlerkorrekturen und Verbesserungen.
2.1.0	Neue Features • Fügt die Unterstützung hinzu, die neuesten Daten zuerst in der Cloud zu veröffentlichen. • Integriert die Unterstützung dafür, abgelaufene Daten nicht in der Cloud zu veröffentlichen. • Fügt Unterstützung für das lokale Speichern abgelaufener Daten hinzu. Fehlerkorrekturen und Verbesserungen • Reduziert Festplatten-I/O und die entsprechende Latenz.
2.0.2	Fehlerkorrekturen und Verbesserungen.

Version	Änderungen
2.0.1	Erste Version

Weitere Informationen finden Sie auch unter

- Was ist AWS IoT SiteWise? im AWS IoT SiteWise Benutzerhandbuch.

SiteWise IoT-Prozessor

Die SiteWise IoT-Prozessorkomponente (`aws.iot.SiteWiseEdgeProcessor`) ermöglicht es AWS IoT SiteWise Gateways, Daten am Edge zu verarbeiten.

Mit dieser Komponente können AWS IoT SiteWise Gateways Asset-Modelle und Assets verwenden, um Daten auf Gateway-Geräten zu verarbeiten. Weitere Informationen zu AWS IoT SiteWise Gateways finden Sie unter Using AWS IoT SiteWise at the Edge im AWS IoT SiteWise Benutzerhandbuch.

Themen

Versionen

Diese Komponente hat die folgenden Versionen:

- 3.2.x
- 3.1.x
- 3.0.x
- 2.2.x
- 2.1.x
- 2.0.x

Typ

Diese Komponente ist eine generische Komponente (`aws.greengrass.generic`). Der Greengrass-Kern führt die Lebenszyklusskripte der Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf Kerngeräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Das Greengrass-Core-Gerät muss auf einer der folgenden Plattformen laufen:
  - Betriebssystem: Ubuntu 20.04 oder 18.04

    Architektur: x86_64 (AMD64)
  - Betriebssystem: Red Hat Enterprise Linux (RHEL) 8

    Architektur: x86_64 (AMD64)
  - Betriebssystem: Amazon Linux 2

    Architektur: x86_64 (AMD64)
  - Betriebssystem: Windows Server 2019 oder höher

Architektur: x86_64 (AMD64)

- Das Greengrass-Core-Gerät muss eingehenden Verkehr auf Port 443 zulassen.
- Das Greengrass-Core-Gerät muss ausgehenden Verkehr auf den Ports 443 und 8883 zulassen.
- Die folgenden Ports sind für die Verwendung durch reserviert AWS IoT SiteWise: 80, 443, 3001, 4569, 4572, 8000, 8081, 8082, 8084, 8085, 8086, 8445, 9000, 9500, 11080 und 50010. Die Verwendung eines reservierten Ports für den Datenverkehr kann zu einem Verbindungsabbruch führen.

> **Note**
>
> Port 8087 ist nur für Version 2.0.15 und höher dieser Komponente erforderlich.

- Die Greengrass-Geräterolle muss über Berechtigungen verfügen, die es Ihnen ermöglichen, AWS IoT SiteWise Gateways auf Ihren AWS IoT Greengrass V2 Geräten zu verwenden. Weitere Informationen finden Sie unter Anforderungen im AWS IoT SiteWise Benutzerhandbuch.

Endpunkte und Anschlüsse

Diese Komponente muss in der Lage sein, ausgehende Anfragen an die folgenden Endpunkte und Ports auszuführen, zusätzlich zu den Endpunkten und Ports, die für den Basisbetrieb erforderlich sind. Weitere Informationen finden Sie unter Zulassen von Gerätedatenverkehr über einen Proxy oder eine Firewall.

Endpunkt	Port	Erforderlich	Beschreibung
`model.iotsitewise.region.amazonaws.com`	443	Ja	Informieren Sie sich über Ihre AWS IoT SiteWise Anlagen und Anlagemodelle.

Endpunkt	Port	Erforderlich	Beschreibung
`edge.iotsitewise.region.amazonaws.com`	443	Ja	Informieren Sie sich über die AWS IoT SiteWise Gateway-Konfiguration des Kerngeräts.
`ecr.region.amazonaws.com`	443	Ja	Laden Sie AWS IoT SiteWise Edge-Gateway-Docker-Images von Amazon Elastic Container Registry herunter.
`iot.region.amazonaws.com`	443	Ja	Holen Sie sich Geräteendpunkte für Ihre. AWS-Konto

Endpunkt	Port	Erforderlich	Beschreibung
`sts.region.amazonaws.com`	443	Ja	Holen Sie sich die ID von Ihrem AWS-Konto.
`monitor.iotsitewise. region.amazonaws.com`	443	Nein	Erforderlich, wenn Sie auf dem Kerngerät auf AWS IoT SiteWise Monitor Portale zugreifen.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt AWS IoT Greengrass auch kompatible Versionen ihrer Abhängigkeiten bereit. Das bedeutet, dass Sie die Anforderungen für die Komponente und all ihre Abhängigkeiten erfüllen müssen, um die Komponente erfolgreich bereitstellen zu können. In diesem Abschnitt werden die Abhängigkeiten für die veröffentlichten Versionen dieser Komponente sowie die semantischen Versionseinschränkungen aufgeführt, die die Komponentenversionen für jede Abhängigkeit definieren. Sie können die Abhängigkeiten für jede Version der Komponente auch in der AWS IoT Greengrass Konsole anzeigen. Suchen Sie auf der Seite mit den Komponentendetails nach der Liste der Abhängigkeiten.

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.0.x bis 2.1.x dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Token-Austauschdienst	>=2.0.3 <3.0.0	Hart
Stream-Manager	>=2,0.3 =2.0.10 <3.0.0	Hart
Greengrass CLI	>=2.3.0 <3.0.0	Hart

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zu den Komponentenrezepten.

Konfiguration

Diese Komponente hat keine Konfigurationsparameter.

Lokale Protokolldatei

Diese Komponente verwendet die folgende Protokolldatei.

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgeProcessor.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeProcessor.log
```

Um die Protokolle dieser Komponente einzusehen

- Führen Sie den folgenden Befehl auf dem Kerngerät aus, um die Protokolldatei dieser Komponente in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.iot.SiteWiseEdgeProcessor.log
  ```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeProcessor.log -Tail 10 -
Wait
```

Lizenzen

Diese Komponente umfasst die folgende Software/Lizenzierung von Drittanbietern:

- Apache-2.0
- MIT
- BSD-2-Klausel
- BSD-3-Klausel
- CDDL-1.0
- CDDL-1,1
- DISC
- Zlib
- GPL-3.0 mit GCC-Ausnahme
- Öffentliche Domäne
- Python-2.0
- Unicode-DFS-2015
- BSD-1-Klausel
- OpenSSL
- EPL-1.0
- EPL-2,0
- GPL-2.0-mit-ClassPath-Ausnahme
- MPL-2.0
- CC0-1,0
- JSON

Diese Komponente wird im Rahmen der Greengrass Core Software-Lizenzvereinbarung veröffentlicht.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
3.2.1	Fehlerkorrekturen und Verbesserungen • Behebung eines Problems, bei dem die AWS IoT SiteWise API-Aufrufe nicht synchron mit SiteWise Edge paginiert wurden. • Problem behoben, bei dem die `MessageRemaining.SiteWise_Edge_Stream` Metrik nicht mehr veröffentlicht wurde. • Die folgenden CloudWatch Metriken wurden hinzugefügt, um die Verbindung mit dem MQTT-Broker zu überwachen. • `IoTSiteWiseProcessor.IsConnectedToMqttBroker` • `IoTSiteWiseProcessor.NumberOfSubscriptionsToMqttBroker` • `IoTSiteWiseProcessor.NumberOfUniqueMqttTopicsReceived` • `IoTSiteWiseProcessor.MqttMessageReceivedSuccessCount` • `IoTSiteWiseProcessor.MqttReceivedSuccessBytes` Weitere Informationen zu diesen Metriken finden Sie unter AWS IoT Greengrass Version 2 Gateway-Metriken.
3.2.0	Leistungsverbesserungen • Optimieren Sie API-Dienste so, dass sie einen geringeren Speicherbedarf haben und weniger Festplattenspeicher für die Installation benötigen • Dadurch wird der anfängliche Speicherverbrauch um 2 GB reduziert (verwendet jetzt 7,5 GB Arbeitsspeicher beim Start, 16 GB werden jedoch weiterhin empfohlen) und die Download-Größe um 500 MB reduziert (erfordert jetzt einen Download von 1,4 GB) für die gesamte Komponente.

Version	Änderungen
	Neue Features • `GetAssetPropertyValueAggregates` Die API unterstützt jetzt 15-minütige Aggregationsfenster am Edge. • Die Ports 8081 und 8082 müssen nicht mehr verfügbar sein, damit diese Komponente ordnungsgemäß ausgeführt werden kann. **Note** Der lokale Endpunkt für AWS IoT SiteWise Datenebenen-APIs, z. B.`get-asset-property-value` , wird von `http://localhost:8081` zu geändert. `http://localhost:11080/data` Der lokale Endpunkt für AWS IoT SiteWise Steuerungsebenen-APIs`list-asset-models` , z. B., wird von `http://localhost:11080` zu geänderthttp://localhost:11080/control . AWS empfiehlt immer, die HTTPS-Endpunkte des SiteWise Edge-Gateways zu verwenden. Diese Endpunkte haben sich nicht geändert. Fehlerkorrekturen und Verbesserungen • Bei der Synchronisierung von AWS IoT SiteWise werden Ressourcen nun in einen gültigen Status versetzt, falls die vorherige Synchronisierung unterbrochen wurde. Dadurch werden Probleme behoben, bei denen einige Ressourcen nach einem erzwungenen Neustart beschädigt wurden. • Behebt ein seltenes Problem, bei dem eine Ressource am Edge beschädigt werden kann, wenn sie während der Synchronisierung geändert wird. Die Synchronisierung schlägt jetzt fehl, wenn dieser Zustand erkannt wird, und die Ressource wird bei der nächsten Synchronisierung erneut versucht. • Behebt ein Problem, durch das der HTTP-Endpunkt für APIs extern aufgerufen werden konnte. Nur HTTPS kann jetzt verwendet werden, um APIs außerhalb der lokalen Loopback-Adresse aufzurufen. • `ListAssets` Die API zeigt jetzt die Asset-Hierarchien für Assets, die am Edge gespeichert sind.

Version	Änderungen
	• Behebt ein Problem, bei dem das Data Processing Pack unter Windows nicht neu gestartet, aktualisiert oder herabgestuft werden konnte. • Behebt einen Fehler im Data Processing Pack für Windows OS, der Kunden daran hinderte, Anmeldeinformationen zu verwenden, um eine Verbindung mit einem MQTT-Broker herzustellen.
3.1.3	Fehlerkorrekturen und Verbesserungen • Behebung eines Problems, bei dem das Data Processing Pack fälschlic herweise eine erfolgreiche Synchronisierung meldete, obwohl einige Ressourcen tatsächlich ausgefallen sind. • Erlauben Sie, dass mehrere Objekte denselben Namen haben, solange sie nicht dasselbe übergeordnete Objekt haben.
3.1.1	Fehlerkorrekturen und Verbesserungen • Behebung eines Problems, bei dem die SigV4-Anfrage aufgrund einer nicht übereinstimmenden Zeitzone fehlschlägt. • Problem behoben, bei dem Transformations- und Metrikeigenschaften nicht mehr berechnet werden, wenn sie nach dem Neustart auf Attribute n basieren. • Aktiviert die Unterstützung der benutzerdefinierten Stream Manager-P ort-Konfiguration. • Behebt ein Problem, bei dem Eigenschaften, die mit dem Edge synchronisiert wurden, möglicherweise nicht mehr aktualisiert werden.
3.1.0	Fehlerkorrekturen und Verbesserungen • Problem behoben, bei dem die `ListAssetModels` API das nächste Token nicht generieren konnte.
3.0.0	Neue Features • Aktiviert die Unterstützung der Datenaufnahme durch einen MQTT-Brok er.

Version	Änderungen
2.2.1	Fehlerkorrekturen und Verbesserungen • Passen Sie den Synchronisierungsprozess an, um die Datenspei cherung auf der Steuerungsebene besser an die Funktionsweise der Cloud anzupassen. Dies wirkt sich geringfügig auf das Upgrade aus. **Note** Daten der Steuerungsebene, die mit Version 2.2.1 oder höher synchronisiert wurden, sind nicht mit früheren Versionen kompatibel. Um ein Downgrade auf frühere Versionen durchzufü hren, müssen Sie eine Neuinstallation durchführen. Dies hat keine Auswirkungen auf Upgrades. Daten, die auf früheren Versionen synchronisiert wurden, funktionieren mit Version 2.2.1. • Zusätzliche Änderungen an der AWS Anmeldeinformationskette zur Priorisierung AWS IoT Greengrass V2 von Anmeldeinformationen.
2.1.37	Fehlerkorrekturen und Verbesserungen • Verwerfen Sie den dependency-routing-service Prozess und übertrage n Sie seine Funktionalität in den property-state-service Prozess, um den Ressourcenverbrauch durch die Kommunikation zwischen Prozessen zu reduzieren. • Erhöhen Sie das maximale Ergebnislimit für die `get-asset-property-value-history` API auf 20.000, um dem von verwendeten Limit zu entsprechen. AWS IoT SiteWise • Behebt ein Problem, bei dem das nächste Token nicht in paginierten Ergebnissen für die `get-asset-property-value-history` API bereitgestellt wurde, obwohl kein maximales Ergebnislimit angegeben wurde.

Version	Änderungen
2.1.35	Fehlerkorrekturen und Verbesserungen • Ändert die AWS Anmeldeinformationskette, um Anmeldeinformationen zu priorisieren AWS IoT Greengrass . • Behebt ein Problem mit der Kontoerkennung bei der Bereitstellung als Teil einer Dinggruppe AWS IoT .
2.1.34	Fehlerkorrekturen und Verbesserungen • Passt Metrik-/Transformationsberechnungen an, um Multithreading unter Linux zu verwenden. Windows führt aus Kompatibilitätsgründen weiterhin Single-Thread-Berechnungen durch. • Behebt ein Problem, bei dem metrische Berechnungen in einigen Berechnungsfenstern fehlten.
2.1.33	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem mit der Meldung von Fehlerzuständen an die Greengrass-Konsole.
2.1.32	Fehlerkorrekturen und Verbesserungen • Integriert die Unterstützung für benutzerdefinierte Benutzernamen und Gruppen.
2.1.31	Fehlerkorrekturen und Verbesserungen • Integriert die Unterstützung für die Berechnung des zeitgewichteten Durchschnitts und der zeitgewichteten Standardabweichung für Daten, die modelliert wurden. AWS IoT SiteWise
2.1.29	Fehlerkorrekturen und Verbesserungen • Fügt Unterstützung für das Filtern von Assets in der Edge-Funktionalität hinzu.
2.1.28	Fehlerkorrekturen und Verbesserungen • Optimiert die Ressourcensynchronisierung, sodass eine große Anzahl von Assets vom Netzwerkrand bis zum Netzwerkrand synchronisiert AWS Cloud werden kann.

Version	Änderungen
2.1.24	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, das dazu führte, dass das Dashboard verschwand, wenn eine Ressource zum zweiten Mal synchronisiert wurde.
2.1.23	Fehlerkorrekturen und Verbesserungen • Es wurde ein Timeout für den `aws.iot.SiteWiseEdgeProcessor` Installationsvorgang hinzugefügt, um Installationsfehler zu vermeiden, wenn die Internetverbindung langsam ist. • Optimierte Ressourcensynchronisierung zur Verbesserung der Synchronisierungseffizienz zwischen Cloud und Edge.
2.1.21	**Warning** Ein Upgrade von 2.0.x auf 2.1.x führt zum Verlust lokaler Daten. Neue Features • Fügt Unterstützung für Windows Server 2019 oder höher hinzu. • Entfernt Docker für Linux-basierte Betriebssysteme.
2.0.16	Diese Version enthält Fehlerkorrekturen und Verbesserungen.
2.0.15	Fehlerkorrekturen und Verbesserungen • Ändert den Port, den diese Komponente für API-Operationen zur Ressourcensynchronisierung verwendet, von 8085 auf 8087. Aus diesem Grund muss für diese Komponente jetzt Port 8087 verfügbar sein. Für diese Komponente muss weiterhin Port 8085 verfügbar sein. • Aktualisiert die AWS OpsHub Authentifizierung, sodass nicht autorisierte Benutzer sich nicht anmelden können, anstatt wenn ein Benutzer versucht, API-Operationen aufzurufen.
2.0.14	Diese Version enthält Fehlerkorrekturen und Verbesserungen.

Version	Änderungen
2.0.13	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, sodass diese Komponente, wenn sie Daten an Amazon CloudWatch Metrics meldet, jetzt korrekt angibt, welche Daten nicht modelliert sind.
2.0.9	Fehlerkorrekturen und Verbesserungen • Verbessert die Zuverlässigkeit beim Erstellen und Aktualisieren von AWS IoT SiteWise Ressourcen auf dem Kerngerät. • Fügt zusätzliche lokale API-Operationen hinzu, mit denen Sie überwachen können, welche Komponenten auf dem Kerngerät installie rt sind, welche Version die einzelnen Komponenten haben und welchen Status die einzelnen Komponenten haben. Sie können diese Informati onen auf der Registerkarte Einstellungen in der AWS IoT SiteWise Anwendung AWS OpsHub for auf dem Kerngerät einsehen. • Fügt einen Integritätsstatus für die Docker-Container hinzu, die diese Komponente ausführt. Sie können den `docker ps` Befehl ausführen, um den Gesundheitsstatus der Container anzuzeigen.
2.0.7	Fehlerkorrekturen und Verbesserungen • Behebt die Unterstützung für die Anzeige von AWS IoT SiteWise Monitor Portalen auf dem Kerngerät.
2.0.6	Fehlerkorrekturen und Verbesserungen • Behebt die AWS IoT SiteWise `statetime() earliest()` , und `latest()` -Funktionen, die diese Komponente auf dem Kerngerät berechnet.
2.0.5	Fehlerkorrekturen und Verbesserungen • Fügt Unterstützung für die AWS IoT SiteWise `pretrigger()` Funktion in Transformationen hinzu, die diese Komponente auf dem Kerngerät berechnet. • Ändert den Pfad, in dem diese Komponente die LDAP-Konfiguration (Lightweight Directory Access Protocol) für die Authentifizierung speichert.

Version	Änderungen
2.0.2	Erste Version

Weitere Informationen finden Sie auch unter

- Was ist AWS IoT SiteWise? im AWS IoT SiteWise Benutzerhandbuch.

Vom Publisher unterstützte Komponenten

Vom Publisher unterstützte Komponenten befinden sich in einer Vorschauversion für AWS IoT Greengrass und können sich ändern. Diese Komponenten werden von nicht unterstütztAWS. Sie müssen sich bei Problemen mit den einzelnen Komponenten an den Herausgeber wenden.

Die von Greengrass Publisher unterstützten Komponenten werden von Drittanbietern von Komponenten entwickelt, angeboten und bedient. Drittanbieter von Komponenten stammen entweder aus dem AWS Partner Gerätekatalog, AWS aus Boles oder aus der Community. Sie können die Komponenten in diesem Katalog kaufen, indem Sie sich direkt an den Drittanbieter von Komponenten wenden.

Zu den von Greengrass Publisher unterstützten Komponenten gehören die folgenden:

Themen

- AIShield.Edge
- KI- EdgeLabs Sensor
- Greengrass-S3-Ingestor

AIShield.Edge

Diese Komponente wurde von AIShield entwickelt und wird von AIShield unterstützt, das von unterstützt wird. Verbessern Sie Ihre KI-Sicherheit mit AIShield.Edge. Diese Komponente ist so konzipiert, dass sie bedrohungsgestützte, maßgeschneiderte Schutzmaßnahmen nahtlos auf Edge-Geräten bereitstellt, die Ihre Geräte vor KI-Angriffen schützen.

Diese Komponente bietet die folgenden Vorteile:

- Nahtloser Übergang von der Schwachstellenanalyse mit AIShield AI Security zu gesicherten Edge-Schutzmaßnahmen innerhalb von AWS
- Einfaches Bereitstellen maßgeschneiderter Schutzmaßnahmen auf mehreren Edge-Geräten
- Breiter Schutz, der auf verschiedene KI-Einrichtungen zugeschnitten ist und verschiedene Modelltypen und Frameworks unterstützt
- Mit nahtloser Integration in Amazon SageMaker und Greengrass-Workflows auf dem neuesten Stand bleiben
- Erhalten Sie sofortige Einblicke in potenzielle Bedrohungen, wobei Daten direkt an weitergeleitet werden AWS IoT Core
- Ein zusammenhängender KI-Sicherheitspfad für die Verteidigungsbereitstellung am Edge von AIShield AI Security auf dem AWS Marketplace

Diese Komponente muss auf der folgenden Plattform ausgeführt werden:

- Betriebssystem: Linux

Wenn Sie diese Komponente erwerben möchten, wenden Sie sich an Software and Digital Solutions: `<AIShield.Contact@bosch.com>` .

KI- EdgeLabs Sensor

Diese Komponente wurde entwickelt und wird von KI unterstützt EdgeLabs. AI EdgeLabs Sensor ist eine containerbasierte Anwendung, die KI-basierte Funktionen zur Erkennung und Verhinderung von Bedrohungen enthält. AI Sensor wird in eine Greengrass-Komponente verpackt und zusammen mit anderen Greengrass-Komponenten als eigenständiger Container auf dem Core-Gerät bereitgestellt.

Diese aktuelle Komponente ist ein containerbasierter Agent, der die Netzwerkkommunikation kontinuierlich überprüft und nach Bedrohungsmustern in Software sucht, die auf dem Edge-Host oder IoT-Gateway ausgeführt wird. Diese Komponente verwendet eBPF, die Verhaltensüberprüfung der Prozessbandbreite und die hostbasierte Konfiguration. Die Hauptfunktionalität dieser Komponente basiert auf NDR/IPS- und EDR-Funktionen.

Diese Komponente bietet die folgenden Vorteile:

- KI-basierte Bedrohungserkennung gegen Netzwerkangriffe und Malware (EDR/NDR)
- Automatisierte KI-basierte Reaktion auf Vorfälle (IPS)
- Host-lokale Bedrohungsinformationen mit minimaler Datenübertragung außerhalb von

- Einfache Bereitstellung mit Docker und Greengrass

Diese Komponente muss auf einer der folgenden Plattformen ausgeführt werden:

- Betriebssystem: Linux

Wenn Sie diese Komponente erwerben möchten, wenden Sie sich an KI EdgeLabs: `<contact@edgelabs.ai>`.

Greengrass-S3-Ingestor

Diese Komponente wurde entwickelt und wird von Nathan Bolr unterstützt. Die Greengrass-S3-Ingestor-Komponente ist für die Verwendung mit der Stream-Manager-Komponente konzipiert. Diese Komponente nimmt einen durch Zeilen getrennten Stream von JSON-Nachrichten aus dem Stream-Manager und stapelt sie in eine GZIP-Datei. Diese Komponente ermöglicht eine effiziente Aufnahme von Daten in Amazon S3 zur Weiterverarbeitung oder Speicherung. Diese Komponente unterstützt nicht das Senden von Daten an in AWS Cloud Echtzeit.

Diese Komponente muss auf einer der folgenden Plattformen ausgeführt werden:

- Betriebssystem: Linux
- Betriebssystem: Windows

Wenn Sie diese Komponente kaufen möchten, wenden Sie sich an Nathan Bolr: `<nathan@glovers.id.au>`.

Komponenten der Gemeinschaft

Der Greengrass Software Catalog ist ein Index von Greengrass-Komponenten, die von der Greengrass-Community entwickelt wurden. Aus diesem Katalog können Sie Komponenten herunterladen, ändern und bereitstellen, um Ihre Greengrass-Anwendungen zu erstellen. Sie können den Katalog unter dem folgenden Link einsehen: https://github.com/aws-greengrass/ aws-greengrass-software-catalog.

Jede Komponente hat ein öffentliches GitHub Repository, das Sie erkunden können. Die vollständige Liste der Community-Komponenten finden GitHub Sie im Greengrass-Softwarekatalog. Dieser Katalog umfasst beispielsweise die folgenden Komponenten:

- Amazon Kinesis Video Streams

  Diese Komponente nimmt Audio- und Videostreams von lokalen Kameras auf, die das Real Time Streaming Protocol (RTSP) verwenden. Die Komponente lädt dann die Audio- und Videostreams in Amazon Kinesis Video Streams hoch.

- Bluetooth-IoT-Gateway

  Diese Komponente verwendet die BluePyBibliothek, die die Kommunikation mit Bluetooth Low Energy (LE) -Geräten ermöglicht, um Bluetooth LE-Clientschnittstellen zu erstellen.

- Rotator für Zertifikate

  Diese Komponente bietet die Möglichkeit, das Zertifikat und den privaten Schlüssel für das AWS IoT Greengrass Kerngerät in Ihrer gesamten Flotte nach Bedarf zu rotieren.

- Sicheres Tunneling in Containern

  Diese Komponente bietet einen Docker-Container für sicheres Tunneling mit allen Abhängigkeiten und passenden Bibliotheken in einem wiederverwendbaren Rezept, das nicht auf ein bestimmtes Host-Betriebssystem angewiesen ist.

- Grafana

  Mit dieser Komponente können Sie einen Grafana-Server auf einem Greengrass-Core-Gerät hosten. Sie können Grafana-Dashboards verwenden, um Daten auf dem Kerngerät zu visualisieren und zu verwalten.

- GStreamer für Amazon Lookout for Vision

  Diese Komponente stellt ein GStreamer-Plugin bereit, mit dem Sie Lookout for Vision Vision-Anomalieerkennung in Ihren benutzerdefinierten GStreamer-Pipelines durchführen können.

- Assistent für zu Hause

  Diese Komponente ermöglicht es dem Kunden, Home Assistant zur lokalen Steuerung von Smart-Home-Geräten zu verwenden. Es ermöglicht die Integration mit AWS Diensten am Netzwerkrand und in der Cloud, um Hausautomationslösungen bereitzustellen, die Home Assistant erweitern.

- InfluxDB und ein Fana-Dashboard

  Diese Komponente bietet eine Ein-Klick-Erfahrung zum Einrichten der InfluxDB- und Grafana-Komponenten. Es verbindet InfluxDB mit Grafana und automatisiert die Einrichtung eines lokalen Grafana-Dashboards, das Telemetrie in Echtzeit rendert AWS IoT Greengrass .

- InfluxDB

Diese Komponente stellt eine InfluxDB-Zeitreihendatenbank auf einem Greengrass-Core-Gerät bereit. Sie können diese Komponente verwenden, um Daten von IoT-Sensoren zu verarbeiten, Daten in Echtzeit zu analysieren und den Betrieb am Edge zu überwachen.

- InfluxDB-Herausgeber

  Diese Komponente leitet die AWS IoT Greengrass Systemintegritätstelemetrie vom Nucleus-Emitter-Plugin an InfluxDB weiter. Diese Komponente kann auch benutzerdefinierte Telemetrie an InfluxDB weiterleiten.

- IoT-Pubsub-Framework

  Dieses Framework bietet eine Anwendungsarchitektur, Vorlagencode und bereitstellbare Beispiele, die dazu beitragen, die Codequalität für verteilte ereignisgesteuerte IoT-Pubsub-Anwendungen mit AWS IoT Greengrass benutzerdefinierten v2-Komponenten zu verbessern. Weitere Informationen finden Sie unter Erstellen von AWS IoT Greengrass Komponenten.

- Jupyter Labs

  Diese Komponente wird auf einem Kerngerät bereitgestellt JupyterLab . AWS IoT Greengrass Die Jupyter-Umgebung hat Zugriff auf die von festgelegten Prozess- und Umgebungsvariablen AWS IoT Greengrass, was das Testen und Entwickeln von in Python geschriebenen Komponenten vereinfacht.

- Lokaler Webserver

  Mit dieser Komponente können Sie eine lokale Webbenutzeroberfläche auf einem Greengrass-Core-Gerät erstellen. Sie können eine lokale Webbenutzeroberfläche erstellen, mit der Sie beispielsweise Geräte- und Anwendungseinstellungen konfigurieren oder das Gerät überwachen können.

- LoRaWaKein Protokolladapter

  Diese Komponente nimmt Daten von lokalen drahtlosen Geräten auf, die das LoRaWa N-Protokoll verwenden, bei dem es sich um ein LPWAN-Protokoll (Low-Power Wide Area Network) handelt. Die Komponente ermöglicht es Ihnen, Daten lokal zu analysieren und darauf zu reagieren, ohne mit der Cloud zu kommunizieren.

- Modbus TCP

  Diese Komponente sammelt Daten von lokalen Geräten mithilfe des ModbusTCP-Protokolls und veröffentlicht sie in ausgewählten Datenströmen.

- Node-RED

  Diese Komponente installiert Node-RED mithilfe von NPM auf einem AWS IoT Greengrass Kerngerät. Die Komponente hängt von der Node-RED Auth-Komponente ab, die explizit bereitgestellt und konfiguriert werden muss. Sie können die Node-RED-CLI für Greengrass verwenden, um Node-RED-Flows auf Geräten bereitzustellen. AWS IoT Greengrass

- Node-RED-Docker

  Diese Komponente installiert Node-RED mithilfe des offiziellen Node-RED-Docker-Containers auf dem AWS IoT Greengrass Kerngerät. Die Komponente hängt von der Node-RED Auth-Komponente ab, die explizit bereitgestellt und konfiguriert werden muss. Sie können die Node-RED-CLI für Greengrass verwenden, um Node-RED-Flows auf Geräten bereitzustellen. AWS IoT Greengrass

- Node-RED-Authentifizierung

  Diese Komponente konfiguriert einen Benutzernamen und ein Passwort, um die Node-RED-Instanz zu sichern, die auf einem Kerngerät ausgeführt wird. AWS IoT Greengrass

- OpenThreadGrenzrouter

  Diese Komponente stellt den OpenThread Border Router Docker-Container bereit. Die Komponente hilft bei der Zusammenstellung eines Matter-Geräts, das einen Thread-Border-Router enthält.

- OSI Pi-Anschluss für Streaming-Daten

  Diese Komponente ermöglicht das Streaming von Daten in Echtzeit aus dem OSI Pi-Datenarchiv in eine moderne Datenarchitektur auf. AWS Sie ist in das OSI Pi Asset Framework integriert, das zentral über AWS IoT PubSub Messaging verwaltet wird.

- Parsec-Anbieter

  Diese Komponente ermöglicht es AWS IoT Greengrass Geräten, Hardware-Sicherheitslösungen mithilfe des Open-Source-Projekts Parsec der Cloud Native Computing Foundation (CNCF) zu integrieren.

- PostgreSQL-Datenbank

  Diese Komponente bietet Unterstützung für die relationale PostgreSQL-Datenbank am Edge. Kunden können diese Komponente verwenden, um eine lokale PostgreSQL-Instanz in einem Docker-Container bereitzustellen und zu verwalten.

- S3-Datei-Uploader

  Diese Komponente überwacht ein Verzeichnis auf neue Dateien, lädt sie auf Amazon Simple Storage Service (Amazon S3) hoch und löscht sie dann nach einem erfolgreichen Upload.

- Secrets Manager Manager-Kunde

  Diese Komponente stellt ein CLI-Tool bereit, das von anderen Komponenten verwendet werden kann, die Geheimnisse aus der Secrets Manager-Komponente in einem Rezeptlebenszyklus-Skript abrufen müssen.

- TES-Routing zum Container

  Diese Komponente konfiguriert nftables oder iptables auf einem AWS IoT Greengrass Gerät, sodass sie die Komponente mit Containern verwenden kann. Token-Exchange-Service

- WebRTC

  Diese Komponente nimmt Audio- und Videostreams von RTSP-Kameras auf, die an das Kerngerät angeschlossen sind. AWS IoT Greengrass Anschließend wandelt die Komponente die Audio- und Videostreams in peer-to-peer Kommunikation oder Weiterleitung über Amazon Kinesis Video Streams um.

Um eine Funktion anzufordern oder einen Fehler zu melden, öffnen Sie ein GitHub Problem im Repository für diese Komponente. AWS bietet keine Unterstützung für Community-Komponenten. Weitere Informationen finden Sie in der CONTRIBUTING.mdDatei im Repository der einzelnen Komponenten.

Einige AWS der bereitgestellten Komponenten sind ebenfalls Open Source. Weitere Informationen finden Sie unter Open-SourceAWS IoT Greengrass-Core-Software.

AWS IoT Greengrass -Entwicklungstools

Verwenden Sie AWS IoT Greengrass Entwicklungstools, um benutzerdefinierte Greengrass-Komponenten zu erstellen, zu testen, zu erstellen, zu veröffentlichen und bereitzustellen.

- CLI des Greengrass Development Kit

  Verwenden Sie die AWS IoT Greengrass Development Kit Command-Line Interface (GDK CLI) in Ihrer lokalen Entwicklungsumgebung, um Komponenten aus Vorlagen und Community-Komponenten im Greengrass Software Catalog zu erstellen. Sie können die GDK-CLI verwenden,

um die Komponente zu erstellen und die Komponente als private Komponente in Ihrem im AWS IoT Greengrass Service zu veröffentlichenAWS-Konto.

- Greengrass-Befehlszeilenschnittstelle

  Verwenden Sie die Greengrass-Befehlszeilenschnittstelle (Greengrass CLI) auf Greengrass-Core-Geräten, um Greengrass-Komponenten bereitzustellen und zu debuggen. Die Greengrass-CLI ist eine Komponente, die Sie auf Ihren -Core-Geräten bereitstellen können, um lokale Bereitstellungen zu erstellen, Details zu installierten Komponenten anzuzeigen und Protokolldateien zu untersuchen.

- Lokale Debug-Konsole

  Verwenden Sie die lokale Debug-Konsole auf Greengrass-Core-Geräten, um Greengrass-Komponenten über eine lokale Dashboard-Webschnittstelle bereitzustellen und zu debuggen. Die lokale Debug-Konsole ist eine Komponente, die Sie auf Ihren -Core-Geräten bereitstellen können, um lokale Bereitstellungen zu erstellen und Details zu installierten Komponenten anzuzeigen.

AWS IoT Greengrass bietet auch die folgenden SDKs, die Sie in benutzerdefinierten Greengrass-Komponenten verwenden können:

- Die AWS IoT Device SDK, die die IPC-Bibliothek (Interprocess Communication) enthält. Weitere Informationen finden Sie unter Verwenden Sie den AWS IoT Device SDK , um mit dem Greengrass-Kern und anderen Komponenten zu kommunizieren und AWS IoT Core.
- Das Stream-Manager-SDK, mit dem Sie Datenströme an die übertragen könnenAWS Cloud. Weitere Informationen finden Sie unter Verwalten von Datenströmen auf Greengrass-Core-Geräten.

Themen

- AWS IoT Greengrass Befehlszeilenschnittstelle des Development Kit
- Greengrass-Befehlszeilenschnittstelle
- Verwenden Sie das AWS IoT Greengrass Testing Framework

AWS IoT Greengrass Befehlszeilenschnittstelle des Development Kit

Das AWS IoT Greengrass Development Kit Command-Line Interface (GDK CLI) bietet Funktionen, mit denen Sie benutzerdefinierte Greengrass-Komponenten entwickeln können. Sie können die GDK-

CLI verwenden, um benutzerdefinierte Komponenten zu erstellen, zu erstellen und zu veröffentlichen. Wenn Sie ein Komponenten-Repository mit der GDK-CLI erstellen, können Sie mit einer Vorlage oder einer Community-Komponente aus dem Greengrass Software Catalog beginnen. Anschließend können Sie ein Build-System auswählen, das Dateien als ZIP-Archive verpackt, ein Maven- oder Gradle-Build-Skript verwendet oder einen benutzerdefinierten Build-Befehl ausführt. Nachdem Sie eine Komponente erstellt haben, können Sie die GDK-CLI verwenden, um sie im AWS IoT Greengrass Service zu veröffentlichen, sodass Sie die AWS IoT Greengrass Konsole oder API verwenden können, um die Komponente auf Ihren Greengrass-Core-Geräten bereitzustellen.

Wenn Sie Greengrass-Komponenten ohne die GDK-CLI entwickeln, müssen Sie die Versions- und Artefakt-URIs in der Komponentenrezeptdatei jedes Mal aktualisieren, wenn Sie eine neue Version der Komponente erstellen. Wenn Sie die GDK-CLI verwenden, können die Versions- und Artefakt-URIs jedes Mal, wenn Sie eine neue Version der Komponente veröffentlichen, automatisch aktualisiert werden.

Die GDK-CLI ist Open Source und auf verfügbar GitHub. Sie können die GDK-CLI an Ihre Anforderungen an die Komponentenentwicklung anpassen und erweitern. Wir laden Sie ein, Probleme zu öffnen und Anfragen im GitHub Repository abzurufen. Die GDK-CLI-Quelle finden Sie unter folgendem Link: https://github.com/aws-greengrass/aws-greengrass-gdk-cli.

Voraussetzungen

Um die Greengrass Development Kit CLI zu installieren und zu verwenden, benötigen Sie Folgendes:

- Ein(e) AWS-Konto. Falls Sie noch keines haben, beachten Sie die Informationen unter Richten Sie eine ein AWS-Konto.
- Ein Windows-, macOS- oder Unix-ähnlicher Entwicklungscomputer mit einer Internetverbindung.
- Für GDK-CLI-Version 1.1.0 oder höher ist Python 3.6 oder höher auf Ihrem Entwicklungscomputer installiert.

  Für GDK-CLI-Version 1.0.0 ist Python 3.8 oder höher auf Ihrem Entwicklungscomputer installiert.
- Git ist auf Ihrem Entwicklungscomputer installiert.
- AWS Command Line Interface (AWS CLI) installiert und mit Anmeldeinformationen auf Ihrem Entwicklungscomputer konfiguriert. Weitere Informationen finden Sie unter Installieren, Aktualisieren und Deinstallieren der AWS CLI und Konfigurieren der AWS CLI im AWS Command Line Interface -Benutzerhandbuch.

> **Note**
>
> Wenn Sie einen Raspberry Pi oder ein anderes 32-Bit-ARM-Gerät verwenden, installieren Sie AWS CLI V1. AWS CLI V2 ist für 32-Bit-ARM-Geräte nicht verfügbar. Weitere Informationen finden Sie unter Installieren, Aktualisieren und Deinstallieren der AWS CLI Version 1.

- Um die GDK-CLI zum Veröffentlichen von Komponenten im AWS IoT Greengrass Service zu verwenden, benötigen Sie die folgenden Berechtigungen:
  - `s3:CreateBucket`
  - `s3:GetBucketLocation`
  - `s3:PutObject`
  - `greengrass:CreateComponentVersion`
  - `greengrass:ListComponentVersions`
- Um die GDK-CLI zum Erstellen einer Komponente zu verwenden, deren Artefakte in einem S3-Bucket und nicht im lokalen Dateisystem vorhanden sind, müssen Sie über die folgenden Berechtigungen verfügen:
  - `s3:ListBucket`

  Diese Funktion ist für GDK CLI v1.1.0 und höher verfügbar.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in jeder Version der GDK-CLI beschrieben. Weitere Informationen finden Sie auf der Seite GDK-CLI-Versionen auf GitHub.

Version	Änderungen
1.6.2	Fehlerbehebungen und Verbesserungen • Behebt ein Problem, bei dem Windows gradlew.bat aufgrund des relativen Pfads nicht funktioniert. • Kleinere Verbesserungen an Protokollierung, Tests und Paketierung.

Version	Änderungen
1.6.1	Fehlerbehebungen und Verbesserungen • Fügt einen Sicherheitsfix für das Analysieren von CLI-Argumenten hinzu. • Ermöglicht dem GDK, den neuesten Greengrass Testing Framework (GTF)-Versionsnamen als Standard-GTF-Version abzurufen. • Ermöglicht GDK, Kunden, die eine ältere Version von GTF verwenden, zu empfehlen, dass sie auf die neueste Version aktualisieren.
1.6.0	Neue Features • Fügt während der `component publish` Befehle `component build` und eine Überprüfung der Rezeptvalidierung anhand des Greengras s-Rezeptschemas hinzu. Dieses Update hilft Entwicklern, verwertba re Probleme innerhalb ihrer Komponentenrezepte zu einem früheren Zeitpunkt im Prozess der Komponentenerstellung zu identifizieren. • Fügt der Vorlage eine Konfidenztestsuite hinzu, die mit dem `test-e2e init` Befehl nach unten abgerufen werden kann. Diese Konfidenz testsuite umfasst acht generische Tests, die verwendet und erweitert werden können, um grundlegende Anforderungen an Komponententests zu erfüllen. Fehlerbehebungen und Verbesserungen • Aktualisiert die vom `test-e2e` Befehl verwendete Standardversion des Greengrass Testing Framework (GTF) auf Version 1.2.0.
1.5.0	Fehlerbehebungen und Verbesserungen Aktualisiert die Muster, die von der `excludes` Build-Option erkannt werden, wenn `build_system` `zip` ist. Diese Version erkennt jetzt globalen Muster, die Pfadnamen basierend auf ihren Platzhalterzeichen abgleichen. Dies ermöglicht eine benutzerdefinierte Angabe, aus welchen Verzeichnissen ausgeschlossen werden sollen.

Version	Änderungen
1.4.0	Neue Features • Fügt einen neuen `config` Befehl hinzu, der eine interaktive Aufforder ung zum Ändern von Feldern innerhalb einer vorhandenen GDK-Konfi gurationsdatei startet. • Ändert die `gdk component publish` Befehle `gdk component build` und , um zu überprüfen, ob die Rezeptgröße den Greengrass-Anforderungen (<=16 000 Bytes) entspricht, bevor fortgefahren wird. Fehlerbehebungen und Verbesserungen • Fügt zusätzliche Protokollierung in der Ausgabe des `gdk component build` Befehls hinzu, wenn ein Rezeptsyntaxfehler verhindert, dass der Build aus Gründen der Kenntnis abgeschlossen wird. • Benennt `otf-options` und `otf-version` in `gtf-options` `gtf-version` bzw. um, da Open Test Framework in Greengrass Testing Framework umbenannt wurde.
1.3.0	Neue Features • Fügt einen neuen `test-e2e` Befehl hinzu, um das end-to-end Testen von Komponenten mit Open Test Framework zu unterstützen. • Fügt eine neue Konfigurationsoption hinzu, `zip_name`um konfiguri erbare ZIP-Dateinamen mit dem ZIP-Build-System zu unterstützen. • Legt die `region` Eigenschaft in der GDK-Konfigurationsdatei als optional fest. Fehlerbehebungen und Verbesserungen • Behebt ein Problem, bei dem ein neues Verzeichnis erstellt wird, auch wenn die angegebene Vorlage oder das angegebene Repository nicht vorhanden ist, wenn ein GDK-Projekt mit dem `--name` Argument initialis iert wird.

Version	Änderungen
1.2.3	Fehlerbehebungen und Verbesserungen • Behebt ein Problem, bei dem die Bucket-Erstellung aufgrund einer falschen Fehlerbehandlung fehlschlägt. • Behebt ein Problem, bei dem Listenstrukturen im Komponentenrezept entfernt werden.
1.2.2	Fehlerbehebungen und Verbesserungen • Bei Rezeptschlüsseln wird die Groß- und Kleinschreibung nicht mehr beachtet. • Fügt eine Prüfung hinzu, um festzustellen, ob ein Bucket in einem vorhanden ist AWS-Region und für den Benutzer zugänglich ist, bevor ein neuer Bucket erstellt wird. Erfordert, dass der Benutzer über die `-GetBucketLocation` Berechtigung verfügt. • Behebt ein Problem mit dem `excludes` Schlüsselwort in der GDK-CLI-Konfigurationsdatei.
1.2.1	Fehlerbehebungen und Verbesserungen • Akzeptiert das Kanada (Zentral) (`ca-central-1`) AWS-Region im Konfigurationseintrag der Region in der `-gdk-config.json` Datei. • Behebt Probleme mit dem `--region` GDK-CLI-Argument für den `-publish`Befehl.

Version	Änderungen
1.2.0	Neue Features • Fügt der `build` Konfiguration in der GDK-CLI-Konfigurationsdatei den `options` Eintrag hinzu. Unterstützt `excludes` unter `options`, um bestimmte Dateien aus dem ZIP-Artefakt auszuschließen, wenn das `zip` Build-System verwendet wird. • Fügt das `gradlew` Build-System hinzu, um Gradle Wrapper zum Erstellen von Komponenten zu verwenden. • Fügt Unterstützung für Kotlin-DSL-Build-Dateien für die `gradle` Build-Option hinzu. • Fügt der `publish` Konfiguration in der GDK-CLI-Konfigurationsdatei einen `options` Eintrag hinzu. Unterstützt das `file_upload_args` unter , `options` um zusätzliche Argumente beim Hochladen von Dateien in Amazon S3 bereitzustellen. Fehlerbehebungen und Verbesserungen • Behebt ein Problem, bei dem Gradle-Builds nicht bereinigt wurden, bevor ein Build-Befehl ausgeführt wurde. • Behebt ein Problem, bei dem der Build nicht beendet wurde, wenn der Build-Befehl fehlschlägt. • Verbessert das Ausgabeformat des `gdk component list` Befehls.

Version	Änderungen
1.1.0	Neue Features • Fügt Unterstützung für das Gradle-Build-System hinzu. • Fügt Unterstützung für das Maven-Build-System auf Windows-Geräten hinzu. • Fügt das `--bucket` Argument dem Befehl zum Veröffentlichen von Komponenten hinzu. Sie können dieses Argument verwenden, um den genauen Bucket anzugeben, in den die GDK-CLI Komponentenartefakte hochlädt. • Fügt das Argument dem init`--name`-Befehl der Komponente hinzu. ??? Sie können diese Option verwenden, um den Ordner anzugeben, in dem die GDK-CLI die Komponente initialisiert. • Fügt Unterstützung für Komponentenartefakte hinzu, die in einem S3-Bucket vorhanden sind, aber nicht im Build-Ordner der lokalen Komponente. Sie können diese Funktion verwenden, um die Bandbreit enkosten für große Komponentenartefakte wie Machine-Learning-M odelle zu senken. Fehlerbehebungen und Verbesserungen • Aktualisiert den Befehl zum Veröffentlichen von Komponenten, um zu überprüfen, ob die Komponente erstellt wurde, bevor sie die Komponent e veröffentlicht. Wenn die Komponente nicht erstellt wird, erstellt dieser Befehl die Komponente jetzt für Sie. • Behebt ein Problem, bei dem das ZIP-Build-System nicht auf Windows-Geräten erstellt werden kann, wenn der ZIP-Dateiname Großbuchs taben enthält. • Verbessert das Protokollnachrichtenformat und ändert die Standardp rotokollebene auf Geräten, `INFO` auf denen Python-Versionen vor 3.8 ausgeführt werden. • Ändert die Mindestanforderung für Python-Versionen in Python 3.6.
1.0.0	Erste Version

Installieren oder Aktualisieren der AWS IoT Greengrass Development-Kit-Befehlszeilenschnittstelle

Das AWS IoT Greengrass Development Kit Command-Line Interface (GDK CLI) basiert auf Python, sodass Sie es `pip` mit auf Ihrem Entwicklungscomputer installieren können.

> **Tip**
>
> Sie können die GDK-CLI auch in virtuellen Python-Umgebungen wie venv installieren. Weitere Informationen finden Sie unter Virtuelle Umgebungen und Pakete in der Python-3-Dokumentation.

So installieren oder aktualisieren Sie die GDK-CLI

1. Führen Sie den folgenden Befehl aus, um die neueste Version der GDK-CLI aus ihrem GitHub Repository zu installieren.

   ```
   python3 -m pip install -U git+https://github.com/aws-greengrass/aws-greengrass-gdk-cli.git@v1.6.2
   ```

   > **Note**
   >
   > Um eine bestimmte Version der GDK-CLI zu installieren, ersetzen Sie *`versionTag`* durch das zu installierende Versions-Tag. Sie können Versions-Tags für die GDK-CLI in ihrem GitHub Repository anzeigen.
   >
   > ```
   > python3 -m pip install -U git+https://github.com/aws-greengrass/aws-greengrass-gdk-cli.git@versionTag
   > ```

2. Führen Sie den folgenden Befehl aus, um zu überprüfen, ob die GDK-CLI erfolgreich installiert wurde.

   ```
   gdk --help
   ```

   Wenn der `gdk` Befehl nicht gefunden wird, fügen Sie seinen Ordner zu PATH hinzu.

- Fügen Sie auf Linux-Geräten `/home/MyUser/.local/bin` zu PATH hinzu und ersetzen Sie durch *MyUser* den Namen Ihres Benutzers.
- Fügen Sie auf Windows-Geräten *PythonPath*`\\Scripts` zu PATH hinzu und ersetzen Sie durch *PythonPath* den Pfad zum Python-Ordner auf Ihrem Gerät.

Sie können jetzt die GDK-CLI verwenden, um Greengrass-Komponenten zu erstellen, zu erstellen und zu veröffentlichen. Weitere Informationen zur Verwendung der GDK-CLI finden Sie unter AWS IoT Greengrass Befehle der Befehlszeilenschnittstelle des Development Kit.

AWS IoT Greengrass Befehle der Befehlszeilenschnittstelle des Development Kit

Die AWS IoT Greengrass Development Kit Command-Line Interface (GDK CLI) bietet eine Befehlszeilenschnittstelle, mit der Sie Greengrass-Komponenten auf Ihrem Entwicklungscomputer erstellen, erstellen und veröffentlichen können. GDK-CLI-Befehle verwenden das folgende Format.

```
gdk <command> <subcommand> [arguments]
```

Wenn Sie die GDK-CLI installieren, fügt das Installationsprogramm `gdk` dem PATH hinzu, sodass Sie die GDK-CLI über die Befehlszeile ausführen können.

Sie können die folgenden Argumente mit jedem Befehl verwenden:

- Verwenden Sie `-h` oder `--help` für Informationen zu einem GDK-CLI-Befehl.
- Verwenden Sie `-v` oder , `--version` um zu sehen, welche Version der GDK-CLI installiert ist.
- Verwenden Sie `-d` oder , `--debug` um ausführliche Protokolle auszugeben, die Sie zum Debuggen der GDK-CLI verwenden können.

In diesem Abschnitt werden die GDK-CLI-Befehle beschrieben und Beispiele für jeden Befehl bereitgestellt. Die Synopsis für jeden Befehl zeigt seine Argumente und ihre Verwendung. Optionale Argumente werden in eckigen Klammern angezeigt.

Verfügbare Befehle

- Komponente
- config
- test-e2e

Komponente

Verwenden Sie den `-component`Befehl in der AWS IoT Greengrass Development Kit Command-Line Interface (GDK CLI), um benutzerdefinierte Greengrass-Komponenten zu erstellen, zu erstellen und zu veröffentlichen.

Unterbefehle

- init
- build
- publish
- auflisten

init

Initialisieren Sie einen Greengrass-Komponentenordner aus einer Komponentenvorlage oder Community-Komponente.

Die GDK-CLI ruft Community-Komponenten aus dem Greengrass Software Catalog und Komponentenvorlagen aus dem AWS IoT Greengrass Komponentenvorlagen-Repository auf ab GitHub.

> **Note**
>
> Wenn Sie GDK CLI v1.0.0 verwenden, müssen Sie diesen Befehl in einem leeren Ordner ausführen. Die GDK-CLI lädt die Vorlage oder die Community-Komponente in den aktuellen Ordner herunter.
> Wenn Sie GDK CLI v1.1.0 oder höher verwenden, können Sie das `--name` Argument angeben, um den Ordner anzugeben, in den die GDK CLI die Vorlage oder Community-Komponente herunterlädt. Wenn Sie dieses Argument verwenden, geben Sie einen Ordner an, der nicht vorhanden ist. Die GDK-CLI erstellt den Ordner für Sie. Wenn Sie dieses Argument nicht angeben, verwendet die GDK-CLI den aktuellen Ordner, der leer sein muss.
> Wenn die Komponente das ZIP-Build-System verwendet, komprimiert die GDK-CLI bestimmte Dateien im Ordner der Komponente in eine ZIP-Datei mit demselben Namen wie der Komponentenordner. Wenn der Name des Komponentenordners beispielsweise lautet`HelloWorld`, erstellt die GDK-CLI eine ZIP-Datei mit dem Namen `HelloWorld.zip`. Im Komponentenrezept muss der ZIP-Artefaktname mit dem Namen des Komponentenordners übereinstimmen. Wenn Sie GDK CLI Version 1.0.0 auf einem

Windows-Gerät verwenden, dürfen der Komponentenordner und die ZIP-Dateinamen nur Kleinbuchstaben enthalten.
Wenn Sie eine Vorlage oder Community-Komponente, die das ZIP-Build-System verwendet, in einen Ordner mit einem anderen Namen als die Vorlage oder Komponente initialisieren, müssen Sie den Namen des ZIP-Artefakts im Komponentenrezept ändern. Aktualisieren Sie die `Lifecycle` Definitionen `Artifacts` und so, dass der ZIP-Dateiname mit dem Namen des Komponentenordners übereinstimmt. Im folgenden Beispiel wird der ZIP-Dateiname in den `Lifecycle` Definitionen `Artifacts` und hervorgehoben.

JSON

```
{
  ...
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Artifacts": [
        {
          "URI": "s3://BUCKET_NAME/COMPONENT_NAME/
COMPONENT_VERSION/HelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ],
      "Lifecycle": {
        "run": "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
      }
    }
  ]
}
```

YAML

```
---
...
Manifests:
  - Platform:
      os: all
    Artifacts:
```

```
      - URI: "s3://BUCKET_NAME/COMPONENT_NAME/
COMPONENT_VERSION/HelloWorld.zip"
        Unarchive: ZIP
    Lifecycle:
      run: "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
```

Syntax

```
$ gdk component init
    [--language]
    [--template]
    [--repository]
    [--name]
```

Argumente (initialisieren aus Komponentenvorlage)

- `-l`, `--language` – Die Programmiersprache, die für die von Ihnen angegebene Vorlage verwendet werden soll.

  Sie müssen entweder `--repository` oder `--language` und angeben`--template`.
- `-t`, `--template` – Die Komponentenvorlage, die für ein lokales Komponentenprojekt verwendet werden soll. Um verfügbare Vorlagen anzuzeigen, verwenden Sie den Befehl list.

  Sie müssen entweder `--repository` oder `--language` und angeben`--template`.
- `-n`, `--name` – (Optional) Der Name des lokalen Ordners, in dem die GDK-CLI die Komponente initialisiert. Geben Sie einen Ordner an, der nicht vorhanden ist. Die GDK-CLI erstellt den Ordner für Sie.

  Diese Funktion ist für GDK CLI v1.1.0 und höher verfügbar.

Argumente (von der Community-Komponente initialisieren)

- `-r`, `--repository` – Die Community-Komponente, die Sie im lokalen Ordner auschecken möchten. Um verfügbare Community-Komponenten anzuzeigen, verwenden Sie den Befehl list.

  Sie müssen entweder `--repository` oder `--language` und angeben`--template`.
- `-n`, `--name` – (Optional) Der Name des lokalen Ordners, in dem die GDK-CLI die Komponente initialisiert. Geben Sie einen Ordner an, der nicht vorhanden ist. Die GDK-CLI erstellt den Ordner für Sie.

Diese Funktion ist für GDK CLI v1.1.0 und höher verfügbar.

Ausgabe

Das folgende Beispiel zeigt die Ausgabe, die erzeugt wird, wenn Sie diesen Befehl ausführen, um einen Komponentenordner aus der Python Hello World-Vorlage zu initialisieren.

```
$ gdk component init -l python -t HelloWorld
[2021-11-29 12:51:40] INFO - Initializing the project directory with a python
 component template - 'HelloWorld'.
[2021-11-29 12:51:40] INFO - Fetching the component template 'HelloWorld-python'
 from Greengrass Software Catalog.
```

Das folgende Beispiel zeigt die Ausgabe, die erzeugt wird, wenn Sie diesen Befehl ausführen, um einen Komponentenordner aus einer Community-Komponente zu initialisieren.

```
$ gdk component init -r aws-greengrass-labs-database-influxdb
[2022-01-24 15:44:33] INFO - Initializing the project directory with a component
 from repository catalog - 'aws-greengrass-labs-database-influxdb'.
[2022-01-24 15:44:33] INFO - Fetching the component repository 'aws-greengrass-labs-
database-influxdb' from Greengrass Software Catalog.
```

build

Bauen Sie die Quelle einer Komponente in ein Rezept und Artefakte ein, die Sie im AWS IoT Greengrass Service veröffentlichen können. Die GDK-CLI führt das Build-System aus, das Sie in der GDK-CLI-Konfigurationsdatei angebengdk-config.json. Sie müssen diesen Befehl in demselben Ordner ausführen, in dem sich die gdk-config.json Datei befindet.

Wenn Sie diesen Befehl ausführen, erstellt die GDK-CLI ein Rezept und Artefakte im greengrass-build Ordner im Komponentenordner. Die GDK-CLI speichert das Rezept im greengrass-build/recipes Ordner und die Artefakte im greengrass-build/artifacts/*componentName*/*componentVersion* Ordner .

Wenn Sie GDK CLI v1.1.0 oder höher verwenden, kann das Komponentenrezept Artefakte angeben, die in einem S3-Bucket vorhanden sind, aber nicht im Build-Ordner der lokalen Komponente. Sie können diese Funktion verwenden, um die Bandbreitennutzung zu reduzieren, wenn Sie Komponenten mit großen Artefakten wie Machine-Learning-Modellen entwickeln.

Nachdem Sie eine Komponente erstellt haben, können Sie sie auf einem Greengrass-Kerngerät wie folgt testen:

- Wenn Sie auf einem anderen Gerät als dem entwickeln, auf dem Sie die AWS IoT Greengrass -Core-Software ausführen, müssen Sie die Komponente veröffentlichen, um sie auf einem Greengrass-Core-Gerät bereitzustellen. Veröffentlichen Sie die Komponente im AWS IoT Greengrass Service und stellen Sie sie auf dem Greengrass-Kerngerät bereit. Weitere Informationen finden Sie unter dem Befehl publish und Erstellen von Bereitstellungen.
- Wenn Sie auf demselben Gerät entwickeln, auf dem Sie die AWS IoT Greengrass -Core-Software ausführen, können Sie die Komponente im AWS IoT Greengrass Service veröffentlichen, um sie bereitzustellen, oder Sie können eine lokale Bereitstellung erstellen, um die Komponente zu installieren und auszuführen. Verwenden Sie die Greengrass-CLI, um eine lokale Bereitstellung zu erstellen. Weitere Informationen finden Sie unter Greengrass-Befehlszeilenschnittstelle und Testen von AWS IoT Greengrass Komponenten mit lokalen Bereitstellungen. Wenn Sie die lokale Bereitstellung erstellen, geben Sie `greengrass-build/recipes` als Rezeptordner und `greengrass-build/artifacts` als Artefaktordner an.

Syntax

```
$ gdk component build
```

Argumente

None

Ausgabe

Das folgende Beispiel zeigt die Ausgabe, die erzeugt wird, wenn Sie diesen Befehl ausführen.

```
$ gdk component build
[2021-11-29 13:18:49] INFO - Getting project configuration from gdk-config.json
[2021-11-29 13:18:49] INFO - Found component recipe file 'recipe.yaml' in the
 project directory.
[2021-11-29 13:18:49] INFO - Building the component 'com.example.PythonHelloWorld'
 with the given project configuration.
[2021-11-29 13:18:49] INFO - Using 'zip' build system to build the component.
[2021-11-29 13:18:49] WARNING - This component is identified as using 'zip' build
 system. If this is incorrect, please exit and specify custom build command in the
 'gdk-config.json'.
[2021-11-29 13:18:49] INFO - Zipping source code files of the component.
```

```
[2021-11-29 13:18:49] INFO - Copying over the build artifacts to the greengrass
 component artifacts build folder.
[2021-11-29 13:18:49] INFO - Updating artifact URIs in the recipe.
[2021-11-29 13:18:49] INFO - Creating component recipe in 'C:\Users\MyUser\Documents
\greengrass-components\python\HelloWorld\greengrass-build\recipes'.
```

publish

Veröffentlichen Sie diese Komponente im AWS IoT Greengrass Service. Dieser Befehl lädt Build-Artefakte in einen S3-Bucket hoch, aktualisiert den Artefakt-URI im Rezept und erstellt eine neue Version der Komponente aus dem Rezept. Die GDK-CLI verwendet den S3-Bucket und die AWS Region, die Sie in der GDK-CLI-Konfigurationsdatei angebengdk-config.json. Sie müssen diesen Befehl in demselben Ordner ausführen, in dem sich die gdk-config.json Datei befindet.

Wenn Sie GDK CLI v1.1.0 oder höher verwenden, können Sie das Argument angeben, um den S3--bucket-Bucket anzugeben, in den die GDK CLI die Artefakte der Komponente hochlädt. Wenn Sie dieses Argument nicht angeben, lädt die GDK-CLI in den S3-Bucket hoch, dessen Name lautet*bucket*-*region*-*accountId*, wobei *Bucket* und *Region* die Werte sind, die Sie in angebengdk-config.json, und *accountId* Ihre AWS-Konto -ID ist. Die GDK-CLI erstellt den Bucket, wenn er nicht vorhanden ist.

Wenn Sie GDK CLI v1.2.0 oder höher verwenden, können Sie die in der GDK-CLI-Konfigurationsdatei AWS-Region angegebene mit dem --region Parameter überschreiben. Sie können auch zusätzliche Optionen mit dem ---optionsParameter angeben. Eine Liste der verfügbaren Optionen finden Sie unter CLI-Konfigurationsdatei des Greengrass Development Kit.

Wenn Sie diesen Befehl ausführen, veröffentlicht die GDK-CLI die Komponente mit der Version, die Sie im Rezept angeben. Wenn Sie angebenNEXT_PATCH, verwendet die GDK-CLI die nächste Patch-Version, die noch nicht vorhanden ist. Semantische Versionen verwenden ein wichtiges .minor .patch-Nummerierungssystem. Weitere Informationen finden Sie in der semantischen Versionsspezifikation .

> **Note**
>
> Wenn Sie GDK CLI v1.1.0 oder höher verwenden und diesen Befehl ausführen, prüft die GDK CLI, ob die Komponente erstellt wurde. Wenn die Komponente nicht erstellt wird, erstellt die GDK-CLI die Komponente, bevor sie die Komponente veröffentlicht.

Syntax

```
$ gdk component publish
    [--bucket] [--region] [--options]
```

Argumente

- `-b`, `--bucket` – (Optional) Geben Sie den Namen des S3-Buckets an, in dem die GDK-CLI Komponentenartefakte veröffentlicht.

  Wenn Sie dieses Argument nicht angeben, lädt die GDK-CLI in den S3-Bucket hoch, dessen Name lautet*`bucket`*-*`region`*-*`accountId`*, wobei *`Bucket`* und *`Region`* die Werte sind, die Sie in angeben`gdk-config.json`, und *`accountId`* Ihre AWS-Konto -ID ist. Die GDK-CLI erstellt den Bucket, wenn er nicht vorhanden ist.

  Die GDK-CLI erstellt den Bucket, wenn er nicht vorhanden ist.

  Diese Funktion ist für GDK CLI v1.1.0 und höher verfügbar.

- `-r`, `--region` – (Optional) Geben Sie den Namen des AWS-Region an, wenn die Komponente erstellt wird. Dieses Argument überschreibt den Regionsnamen in der GDK-CLI-Konfiguration.

  Diese Funktion ist für GDK CLI v1.2.0 und höher verfügbar.

- `-o`, `--options` (Optional) Geben Sie eine Liste von Optionen zum Veröffentlichen einer Komponente an. Das Argument muss eine gültige JSON-Zeichenfolge oder ein Dateipfad zu einer JSON-Datei sein, die die Veröffentlichungsoptionen enthält. Dieses Argument überschreibt die Optionen in der GDK-CLI-Konfiguration.

  Diese Funktion ist für GDK CLI v1.2.0 und höher verfügbar.

Ausgabe

Das folgende Beispiel zeigt die Ausgabe, die erzeugt wird, wenn Sie diesen Befehl ausführen.

```
$ gdk component publish
[2021-11-29 13:45:29] INFO - Getting project configuration from gdk-config.json
[2021-11-29 13:45:29] INFO - Found component recipe file 'recipe.yaml' in the
 project directory.
[2021-11-29 13:45:29] INFO - Found credentials in shared credentials file: ~/.aws/
credentials
[2021-11-29 13:45:30] INFO - Publishing the component 'com.example.PythonHelloWorld'
 with the given project configuration.
```

```
[2021-11-29 13:45:30] INFO - No private version of the component
 'com.example.PythonHelloWorld' exist in the account. Using '1.0.0' as the next
 version to create.
[2021-11-29 13:45:30] INFO - Uploading the component built artifacts to s3 bucket.
[2021-11-29 13:45:30] INFO - Uploading component artifacts to S3 bucket: {bucket}.
 If this is your first time using this bucket, add the 's3:GetObject' permission
 to each core device's token exchange role to allow it to download the component
 artifacts. For more information, see https://docs.aws.amazon.com/greengrass/v2/
developerguide/device-service-role.html.
[2021-11-29 13:45:30] INFO - Not creating an artifacts bucket as it already exists.
[2021-11-29 13:45:30] INFO - Updating the component recipe
 com.example.PythonHelloWorld-1.0.0.
[2021-11-29 13:45:30] INFO - Creating a new greengrass component
 com.example.PythonHelloWorld-1.0.0
[2021-11-29 13:45:30] INFO - Created private version '1.0.0' of the component in the
 account.'com.example.PythonHelloWorld'.
```

auflisten

Rufen Sie die Liste der verfügbaren Komponentenvorlagen und Community-Komponenten ab.

Die GDK-CLI ruft Community-Komponenten aus dem Greengrass Software Catalog und Komponentenvorlagen aus dem AWS IoT Greengrass Komponentenvorlagen-Repository auf ab GitHub.

Sie können die Ausgabe dieses Befehls an den init-Befehl übergeben, um Komponenten-Repositorys aus Vorlagen und Community-Komponenten zu initialisieren.

Syntax

```
$ gdk component list
    [--template]
    [--repository]
```

Argumente

- `-t`, `--template` – (Optional) Geben Sie dieses Argument an, um verfügbare Komponentenvorlagen aufzulisten. Dieser Befehl gibt den Namen und die Sprache jeder Vorlage im Format aus*`name`*`-`*`language`*. In lautet `HelloWorld-python`der Vorlagenname beispielsweise `HelloWorld` und die Sprache ist `python`.

- `-r`, `--repository` – (Optional) Geben Sie dieses Argument an, um verfügbare Community-Komponenten-Repositorys aufzulisten.

Ausgabe

Das folgende Beispiel zeigt die Ausgabe, die erzeugt wird, wenn Sie diesen Befehl ausführen.

```
$ gdk component list --template
[2021-11-29 12:29:04] INFO - Listing all the available component templates from
 Greengrass Software Catalog.
[2021-11-29 12:29:04] INFO - Found '2' component templates to display.
1. HelloWorld-python
2. HelloWorld-java
```

config

Verwenden Sie den `config` Befehl im AWS IoT Greengrass Development Kit Command-Line Interface (GDK CLI), um die Konfiguration für das GDK in der Konfigurationsdatei zu ändern. `gdk-config.json`

Unterbefehle

- update

update

Startet eine interaktive Eingabeaufforderung, um Felder in einer vorhandenen GDK-Konfigurationsdatei zu ändern.

Syntax

```
$ gdk config update
    [--component]
```

Argumente

- `-c`, `--component` — Um komponentenbezogene Felder in der Datei zu aktualisieren. `gdk-config.json` Dieses Argument ist erforderlich, da es die einzige Option ist.

Ausgabe

Das folgende Beispiel zeigt die Ausgabe, die erzeugt wird, wenn Sie diesen Befehl ausführen, um eine Komponente zu konfigurieren.

```
$ gdk config update --component
Current value of the REQUIRED component_name is (default:
 com.example.PythonHelloWorld):
Current value of the REQUIRED author is (default: author):
Current value of the REQUIRED version is (default: NEXT_PATCH):
Do you want to change the build configurations? (y/n)
Do you want to change the publish configurations? (y/n)
[2023-09-26 10:19:48] INFO - Config file has been updated. Exiting...
```

test-e2e

Verwenden Sie den `-test-e2e`Befehl in der AWS IoT Greengrass Development Kit Command-Line Interface (GDK CLI), um end-to-end Testmodule im GDK-Projekt zu initialisieren, zu erstellen und auszuführen.

Unterbefehle

- init
- build
- run

init

Initialisieren Sie ein vorhandenes GDK-CLI-Projekt mit einem Testmodul, das Greengrass Testing Framework (GTF) verwendet.

Standardmäßig ruft die GDK-CLI die Maven-Modulvorlage aus dem AWS IoT Greengrass Komponentenvorlagen-Repository auf ab GitHub. Dieses Maven-Modul ist von der `aws-greengrass-testing-standalone` JAR-Datei abhängig.

Dieser Befehl erstellt ein neues Verzeichnis namens `gg-e2e-tests` innerhalb des GDK-Projekts. Wenn das Testmodulverzeichnis bereits vorhanden ist und nicht leer ist, wird der Befehl beendet, ohne etwas zu tun. Dieser `gg-e2e-tests` Ordner enthält die Cucumber-Funktion und Schrittdefinitionen, die in einem Maven-Projekt strukturiert sind.

Standardmäßig versucht dieser Befehl, die neueste Version von GTF zu verwenden.

Syntax

```
$ gdk test-e2e init
```

```
    [--gtf-version]
```

Argumente

- `-ov`, `--gtf-version` – (Optional) Die Version der GTF, die mit dem end-to-end Testmodul im GDK-Projekt verwendet werden soll. Dieser Wert muss eine der GTF-Versionen aus Versionen sein. Dieses Argument überschreibt die `gtf_version` in der GDK-CLI-Konfiguration.

Ausgabe

Das folgende Beispiel zeigt die Ausgabe, die erzeugt wird, wenn Sie diesen Befehl ausführen, um das GDK-Projekt mit dem Testmodul zu initialisieren.

```
$ gdk test-e2e init
[2023-12-06 12:20:28] INFO - Using the GTF version provided in the GDK test config
 1.2.0
[2023-12-06 12:20:28] INFO - Downloading the E2E testing template from GitHub into
 gg-e2e-tests directory...
```

build

> **Note**
>
> Sie müssen die Komponente erstellen, indem Sie ausführen, gdk component build bevor Sie das end-to-end Testmodul erstellen.

Erstellen Sie das end-to-end Testmodul. Die GDK-CLI erstellt das Testmodul mit dem Build-System, das Sie in der GDK-CLI-Konfigurationsdatei , `gdk-config.json`, unter der `-test-e2e`Eigenschaft angeben. Sie müssen diesen Befehl in demselben Ordner ausführen, in dem sich die `gdk-config.json` Datei befindet.

Standardmäßig verwendet die GDK-CLI ein Maven-Build-System, um das Testmodul zu erstellen. Maven ist erforderlich, um den `gdk test-e2e build` Befehl auszuführen.

Sie müssen die Komponente erstellen, indem Sie ausführen, gdk-component-build bevor Sie das Testmodul erstellen, wenn die Testfunktionsdateien Variablen wie `GDK_COMPONENT_NAME` und `GDK_COMPONENT_RECIPE_FILE` zum Interpolieren enthalten.

Wenn Sie diesen Befehl ausführen, interpoliert die GDK-CLI alle Variablen aus der GDK-Projektkonfiguration und erstellt das `gg-e2e-tests` Modul, um die endgültige Test-JAR-Datei zu generieren.

Syntax

```
$ gdk test-e2e build
```

Argumente

None

Ausgabe

Das folgende Beispiel zeigt die Ausgabe, die erzeugt wird, wenn Sie diesen Befehl ausführen.

```
$ gdk test-e2e build
[2023-07-20 15:36:48] INFO - Updating feature file: file:///path/to//
HelloWorld/greengrass-build/gg-e2e-tests/src/main/resources/greengrass/features/
component.feature
[2023-07-20 15:36:48] INFO - Creating the E2E testing recipe file:///path/to/
HelloWorld/greengrass-build/recipes/e2e_test_recipe.yaml
[2023-07-20 15:36:48] INFO - Building the E2E testing module
[2023-07-20 15:36:48] INFO - Running the build command 'mvn package'
.........
```

run

Führen Sie das Testmodul mit den Testoptionen in der GDK-Konfigurationsdatei aus.

> **Note**
>
> Sie müssen das Testmodul erstellen, indem Sie ausführen, gdk test-e2e build bevor Sie die end-to-end Tests ausführen.

Syntax

```
$ gdk test-e2e run
    [--gtf-options]
```

Argumente

- `-oo`, `--gtf-options` – (Optional) Geben Sie eine Liste von Optionen zum Ausführen der end-to-end Tests an. Das Argument muss eine gültige JSON-Zeichenfolge oder ein Dateipfad zu einer JSON-Datei sein, die die GTF-Optionen enthält. Die in der Konfigurationsdatei bereitgestellten Optionen werden mit denen in den Befehlsargumenten zusammengeführt. Wenn an beiden Stellen eine Option vorhanden ist, hat die im Argument angegebene Vorrang vor der Option aus der Konfigurationsdatei.

  Wenn die `tags` Option in diesem Befehl nicht angegeben ist, verwendet GDK `Sample` für Tags. Wenn nicht angegeben `ggc-archive` ist, lädt GDK die neueste Version des Greengrass-Kernarchivs herunter.

Ausgabe

Das folgende Beispiel zeigt die Ausgabe, die erzeugt wird, wenn Sie diesen Befehl ausführen.

```
$ gdk test-e2e run
[2023-07-20 16:35:53] INFO - Downloading latest nucleus archive from url https://
d2s8p88vqu9w66.cloudfront.net/releases/greengrass-latest.zip
[2023-07-20 16:35:57] INFO - Running test jar with command java -jar /path/to/
greengrass-build/gg-e2e-tests/target/uat-features-1.0.0.jar –ggc-archive=/path/to/
aws-greengrass-gdk-cli/HelloWorld/greengrass-build/greengrass-nucleus-latest.zip –
tags=Sample

16:35:59.693 [] [] [] [INFO]
 com.aws.greengrass.testing.modules.GreengrassContextModule - Extracting /path/
to/workplace/aws-greengrass-gdk-cli/HelloWorld/greengrass-build/greengrass-
nucleus-latest.zip into /var/folders/7g/ltzcb_3s77nbtmkzfb6brwv40000gr/T/gg-
testing-7718418114158172636/greengrass
16:36:00.534 [gtf-1.1.0-SNAPSHOT] [] [] [INFO]
 com.aws.greengrass.testing.features.LoggerSteps - GTF Version is gtf-1.1.0-SNAPSHOT
.......
```

CLI-Konfigurationsdatei des Greengrass Development Kit

Das AWS IoT Greengrass Development Kit Command-Line Interface (GDK CLI) liest aus einer Konfigurationsdatei mit dem Namen `gdk-config.json`, um Komponenten zu erstellen und zu veröffentlichen. Diese Konfigurationsdatei muss im Stammverzeichnis des Komponenten-Repositorys vorhanden sein. Sie können den Befehl GDK CLI init verwenden, um Komponenten-Repositorys mit dieser Konfigurationsdatei zu initialisieren.

Themen

Format der GDK-CLI-Konfigurationsdatei

Wenn Sie eine GDK-CLI-Konfigurationsdatei für eine Komponente definieren, geben Sie die folgenden Informationen im JSON-Format an.

`gdk_version`

Die Mindestversion der GDK-CLI, die mit dieser Komponente kompatibel ist. Dieser Wert muss eine der GDK-CLI-Versionen aus Versionen sein.

`component`

Die Konfiguration für diese Komponente.

`componentName`

`author`

Der Autor oder Herausgeber der Komponente.

`version`

Die Version der Komponente. Geben Sie eines der folgenden Elemente an:

- `NEXT_PATCH` – Wenn Sie diese Option wählen, legt die GDK-CLI die Version fest, wenn Sie die Komponente veröffentlichen. Die GDK-CLI fragt den AWS IoT Greengrass Service ab, um die neueste veröffentlichte Version der Komponente zu identifizieren. Anschließend wird die Version auf die nächste Patch-Version nach dieser Version festgelegt. Wenn Sie die Komponente noch nicht veröffentlicht haben, verwendet die GDK-CLI Version `1.0.0`.

  Wenn Sie diese Option wählen, können Sie die Greengrass-CLI nicht verwenden, um die Komponente lokal auf Ihrem lokalen Entwicklungscomputer bereitzustellen und zu testen, auf dem die AWS IoT Greengrass Core-Software ausgeführt wird. Um lokale Bereitstellungen zu aktivieren, müssen Sie stattdessen eine semantische Version angeben.

- Eine semantische Version, z. B. **`1.0.0`**. Semantische Versionen verwenden ein Hauptversions-.Nebenversions-.Patch-Nummerierungssystem. Weitere Informationen finden Sie in der semantischen Versionsspezifikation .

  Wenn Sie Komponenten auf einem Greengrass-Core-Gerät entwickeln, auf dem Sie die Komponente bereitstellen und testen möchten, wählen Sie diese Option. Sie müssen die Komponente mit einer bestimmten Version erstellen, um lokale Bereitstellungen mit der Greengrass-CLI zu erstellen.

`build`

Die Konfiguration, die verwendet werden soll, um die Quelle dieser Komponente in Artefakte zu integrieren. Dieses Objekt enthält die folgenden Informationen:

`build_system`

Das zu verwendende Build-System. Wählen Sie aus den folgenden Optionen aus:

- `zip` – Verpackt den Ordner der Komponente in eine ZIP-Datei, um als einziges Artefakt der Komponente zu definieren. Wählen Sie diese Option für die folgenden Komponententypen aus:
  - Komponenten, die interpretierte Programmiersprachen wie Python oder verwenden JavaScript.
  - Komponenten, die andere Dateien als Code verpacken, z. B. Modelle für maschinelles Lernen oder andere Ressourcen.

  Die GDK-CLI komprimiert den Ordner der Komponente in eine ZIP-Datei mit demselben Namen wie der Komponentenordner. Wenn der Name des Komponentenordners beispielsweise lautet`HelloWorld`, erstellt die GDK-CLI eine ZIP-Datei mit dem Namen `HelloWorld.zip`.

  > **Note**
  >
  > Wenn Sie GDK CLI Version 1.0.0 auf einem Windows-Gerät verwenden, dürfen der Komponentenordner und die ZIP-Dateinamen nur Kleinbuchstaben enthalten.

  Wenn die GDK-CLI den Ordner der Komponente in eine ZIP-Datei komprimiert, überspringt sie die folgenden Dateien:

- Die Datei `gdk-config.json`
- Die Rezeptdatei (`recipe.json` oder `recipe.yaml`)
- Erstellen von Ordnern, z. B. `greengrass-build`

- `maven` – Führt den `mvn clean package` Befehl aus, um die Quelle der Komponente in Artefakte zu erstellen. Wählen Sie diese Option für Komponenten aus, die Maven verwenden, z. B. Java-Komponenten.

  Auf Windows-Geräten ist diese Funktion für GDK CLI v1.1.0 und höher verfügbar.
- `gradle` – Führt den `gradle build` Befehl aus, um die Quelle der Komponente in Artefakte zu erstellen. Wählen Sie diese Option für Komponenten aus, die Gradle verwenden. Diese Funktion ist für GDK CLI v1.1.0 und höher verfügbar.

  Das `gradle` Build-System unterstützt Kotlin DSL als Build-Datei. Diese Funktion ist für GDK CLI v1.2.0 und höher verfügbar.
- `gradlew` – Führt den `gradlew` Befehl aus, um die Quelle der Komponente in Artefakte zu erstellen. Wählen Sie diese Option für Komponenten aus, die den Gradle Wrapper verwenden.

  Diese Funktion ist für GDK CLI v1.2.0 und höher verfügbar.
- `custom` – Führt einen benutzerdefinierten Befehl aus, um die Quelle der Komponente in ein Rezept und Artefakte zu integrieren. Geben Sie den benutzerdefinierten Befehl im `custom_build_command` Parameter an.

`custom_build_command`

(Optional) Der benutzerdefinierte Build-Befehl, der für ein benutzerdefiniertes Build-System ausgeführt werden soll. Sie müssen diesen Parameter angeben, wenn Sie `custom` für angeben`build_system`.

> **Important**
>
> Dieser Befehl muss ein Rezept und Artefakte in den folgenden Ordnern im Komponentenordner erstellen. Die GDK-CLI erstellt diese Ordner für Sie, wenn Sie den Komponentenerstellungsbefehl ausführen.
>
> - Rezeptordner: `greengrass-build/recipes`
> - Artefakte-Ordner: `greengrass-build/artifacts/componentName/componentVersion`

Ersetzen Sie *componentName* durch den Komponentennamen und *componentVersion* durch die Komponentenversion oder NEXT_PATCH.

Sie können eine einzelne Zeichenfolge oder eine Liste von Zeichenfolgen angeben, wobei jede Zeichenfolge ein Wort im Befehl ist. Um beispielsweise einen benutzerdefinierten Build-Befehl für eine C++-Komponente auszuführen, können Sie **`cmake --build build --config Release`** oder angeben**`["cmake", "--build", "build", "--config", "Release"]`**.

Ein Beispiel für ein benutzerdefiniertes Build-System finden Sie unter aws.greengrass.labs.LocalWebServer community component auf GitHub.

`options`

(Optional) Zusätzliche Konfigurationsoptionen, die während des Komponentenerstellungsprozesses verwendet werden.

Diese Funktion ist für GDK CLI v1.2.0 und höher verfügbar.

`excludes`

Eine Liste von globalen Mustern, die definieren, welche Dateien beim Erstellen der ZIP-Datei aus dem Komponentenverzeichnis ausgeschlossen werden sollen. Nur gültig, wenn `build_system` ist`zip`.

> **Note**
>
> In den GDK-CLI-Versionen 1.4.0 und früher wird jede Datei, die mit einem Eintrag in der Ausschlussliste übereinstimmt, aus allen Unterverzeichnissen der Komponente ausgeschlossen. Um dasselbe Verhalten in GDK-CLI-Versionen 1.5.0 und höher `**/` zu erreichen, stellen Sie den vorhandenen Einträgen in der Ausschlussliste voran. Beispielsweise `*.txt` schließt Textdateien nur aus dem Verzeichnis aus; `**/*.txt` schließt Textdateien aus allen Verzeichnissen und Unterverzeichnissen aus.
> In GDK-CLI-Versionen 1.5.0 und höher wird möglicherweise eine Warnung während des Komponentenaufbaus angezeigt, wenn in der GDK-Konfigurationsdatei definiert `excludes` ist. Um diese

Warnung zu deaktivieren, setzen Sie die Umgebungsvariable `GDK_EXCLUDES_WARN_IGNORE` auf `true`.

Die GDK-CLI schließt immer die folgenden Dateien aus der ZIP-Datei aus:

- Die Datei `gdk-config.json`
- Die Rezeptdatei (`recipe.json` oder `recipe.yaml`)
- Erstellen von Ordnern, z. B. `greengrass-build`

Die folgenden Dateien sind standardmäßig ausgeschlossen. Mit der `excludes` Option können Sie jedoch steuern, welche dieser Dateien ausgeschlossen werden.

- Jeder Ordner, der mit dem Präfix „test“ (`test*`) beginnt
- Alle ausgeblendeten Dateien
- Den Ordner `node_modules`

Wenn Sie die `excludes` Option angeben, schließt die GDK-CLI nur die Dateien aus, die Sie mit der `excludes` Option festgelegt haben. Wenn Sie die `excludes` Option nicht angeben, schließt die GDK-CLI die zuvor genannten Standarddateien und Ordner aus.

`zip_name`

Der ZIP-Dateiname, der beim Erstellen eines ZIP-Artefakts während des Erstellungsprozesses verwendet werden soll. Nur gültig, wenn `build_system` ist`zip`. Wenn leer `build_system` ist, wird der Komponentenname für den ZIP-Dateinamen verwendet.

`publish`

Die Konfiguration, die zum Veröffentlichen dieser Komponente im AWS IoT Greengrass Service verwendet werden soll.

Wenn Sie GDK CLI v1.1.0 oder höher verwenden, können Sie das Argument angeben, um den S3`--bucket`-Bucket anzugeben, in den die GDK CLI die Artefakte der Komponente hochlädt. Wenn Sie dieses Argument nicht angeben, lädt die GDK-CLI in den S3-Bucket hoch, dessen Name lautet*`bucket-region-accountId`*, wobei *`Bucket`* und *`Region`* die Werte sind, die Sie in angebengdk`-config.json`, und *`accountId`* Ihre AWS-Konto -ID ist. Die GDK-CLI erstellt den Bucket, wenn er nicht vorhanden ist.

Dieses Objekt enthält die folgenden Informationen:

`bucket`

Der S3-Bucket-Name, der zum Hosten von Komponentenartefakten verwendet werden soll.

`region`

Die AWS-Region, in der die GDK-CLI diese Komponente veröffentlicht.

Diese Eigenschaft ist optional, wenn Sie GDK CLI v1.3.0 oder höher verwenden.

`options`

(Optional) Zusätzliche Konfigurationsoptionen, die bei der Erstellung der Komponentenversion verwendet werden.

Diese Funktion ist für GDK CLI v1.2.0 und höher verfügbar.

`file_upload_args`

Eine JSON-Struktur, die Argumente enthält, die beim Hochladen von Dateien in einen Bucket an Amazon S3 gesendet werden, z. B. Metadaten und Verschlüsselungsmechanismen. Eine Liste der zulässigen Argumente finden Sie in der -`S3Transfer`Klasse in der Boto3-Dokumentation.

`test-e2e`

(Optional) Die Konfiguration, die beim end-to-end Testen der Komponente verwendet werden soll. Diese Funktion ist für GDK CLI v1.3.0 und höher verfügbar.

`build`

`build_system` – Das zu verwendende Build-System. Die Standardoption ist `maven`. Wählen Sie aus den folgenden Optionen aus:

- `maven` – Führt den `mvn package` Befehl aus, um das Testmodul zu erstellen. Wählen Sie diese Option zum Erstellen des Testmoduls, das Maven verwendet.
- `gradle` – Führt den `gradle build` Befehl aus, um das Testmodul zu erstellen. Wählen Sie diese Option für das Testmodul aus, das Gradle verwendet.

`gtf_version`

(Optional) Die Version des Greengrass Testing Framework (GTF), die als Abhängigkeit des end-to-end Testmoduls verwendet werden soll, wenn Sie das GDK-Projekt mit

GTF initialisieren. Dieser Wert muss eine der GTF-Versionen aus Versionen sein. Der Standardwert ist GTF Version 1.1.0.

`gtf_options`

(Optional) Zusätzliche Konfigurationsoptionen, die beim end-to-end Testen der Komponente verwendet werden.

Die folgende Liste enthält die Optionen, die Sie mit GTF Version 1.1.0 verwenden können.

- `additional-plugins` – (Optional) Zusätzliche Cucumber-Plugins
- `aws-region` – Zielt auf bestimmte regionale Endpunkte für -AWSServices ab. Standardmäßig ist das, was das AWS SDK erkennt.
- `credentials-path` – Optionaler Pfad für AWS Profilanmeldeinformationen. Standardmäßig werden Anmeldeinformationen verwendet, die in der Hostumgebung erkannt wurden.
- `credentials-path-rotation` – Optionale Rotationsdauer für AWS Anmeldeinformationen. Der Standardwert ist 15 Minuten oder `PT15M`.
- `csr-path` – Der Pfad für die CSR, mit der das Gerätezertifikat generiert wird.
- `device-mode` – Das zu testende Zielgerät. Standardmäßig ist das lokale Gerät.
- `env-stage` – Zielt auf die Bereitstellungsumgebung von Greengrass ab. Der Standardwert ist Produktion.
- `existing-device-cert-arn` – Der ARN eines vorhandenen Zertifikats, das Sie als Gerätezertifikat für Greengrass verwenden möchten.
- `feature-path` – Datei oder Verzeichnis, das zusätzliche Feature-Dateien enthält. Standardmäßig werden keine zusätzlichen Feature-Dateien verwendet.
- `gg-cli-version` – Überschreibt die Version der Greengrass-CLI. Der Standardwert ist der Wert in `ggc.version`.
- `gg-component-bucket` – Der Name eines vorhandenen Amazon S3-Buckets, der Greengrass-Komponenten enthält.
- `gg-component-overrides` – Eine Liste von Greengrass-Komponentenüberschreibungen.
- `gg-persist` – Eine Liste von Testelementen, die nach einem Testlauf bestehen bleiben sollen. Das Standardverhalten besteht darin, nichts beizubehalten. Zulässige Werte sind: `aws.resources`, `installed.software`, und `generated.files`.
- `gg-runtime` – Eine Liste von Werten, die beeinflussen, wie der Test mit Testressourcen interagiert. Diese Werte ersetzen den `gg.persist` Parameter . Wenn der Standardwert

leer ist, wird davon ausgegangen, dass alle Testressourcen von Testfällen verwaltet werden, einschließlich der installierten Greengrass-Laufzeit. Akzeptierte Werte sind: `aws.resourcesinstalled.software`, und `generated.files`.

- `ggc-archive` – Der Pfad zur archivierten Greengrass-Kernkomponente.
- `ggc-install-root` – Verzeichnis zur Installation der Greengrass-Kernkomponente. Standardmäßig sind test.temp.path und test run folder.
- `ggc-log-level` – Legen Sie die Greengrass-Kernprotokollebene für den Testlauf fest. Der Standardwert ist „INFO“.
- `ggc-tes-rolename` – Die IAM-Rolle, die AWS IoT Greengrass Core für den Zugriff auf -AWSServices übernimmt. Wenn keine Rolle mit dem angegebenen Namen vorhanden ist, wird eine erstellt und es wird eine Standardzugriffsrichtlinie verwendet.
- `ggc-trusted-plugins` – Die durch Komma getrennte Liste der Pfade (auf dem Host) der vertrauenswürdigen Plugins, die Greengrass hinzugefügt werden müssen. Um den Pfad auf dem DUT selbst bereitzustellen, stellen Sie dem Pfad „dut:“ voran.
- `ggc-user-name` – Der Wert user:group posixUser für den Greengrass-Kern. Standardmäßig wird der aktuelle Benutzername verwendet, der angemeldet ist.
- `ggc-version` – Überschreibt die Version der ausgeführten Greengrass-Kernkomponente. Standardmäßig ist der Wert in ggc.archive angegeben.
- `log-level` – Protokollebene des Testlaufs. Der Standardwert ist „INFO“.
- `parallel-config` – Satz von Batch-Index und Anzahl der Batches als JSON-Zeichenfolge. Der Standardwert des Batch-Index ist 0 und die Anzahl der Batches ist 1.
- `proxy-url` – Konfigurieren Sie alle Tests so, dass der Datenverkehr über diese URL geleitet wird.
- `tags` – Führen Sie nur Feature-Tags aus. Kann mit „&“ überlappen
- `test-id-prefix` – Ein gemeinsames Präfix, das auf alle testspezifischen Ressourcen angewendet wird, einschließlich AWS Ressourcennamen und Tags. Der Standardwert ist ein „gg“-Präfix.
- `test-log-path` – Verzeichnis, das die Ergebnisse des gesamten Testlaufs enthält. Standardmäßig ist „testResults“.
- `test-results-json` – Flag, um festzustellen, ob ein resultierender Cucumber-JSON-Bericht generiert wird, der auf die Festplatte geschrieben wird. Standardwert ist „true“.
- `test-results-log` – Flag, um festzustellen, ob die Konsolenausgabe generiert und auf die Festplatte geschrieben wird. Standardwert "false".

- `test-results-xml` – Flag, um festzustellen, ob ein resultierender JUnit-XML-Bericht generiert wird, der auf die Festplatte geschrieben wird. Standardwert ist „true“.
- `test-temp-path` – Verzeichnis zum Generieren lokaler Testartefakte. Standardmäßig ist ein zufälliges temporäres Verzeichnis mit dem Präfix gg-testing.
- `timeout-multiplier` – Multiplikator für alle Test-Timeouts bereitgestellt. Standard = 1.0.

Beispiele für GDK-CLI-Konfigurationsdateien

Sie können auf die folgenden Beispiele für GDK-CLI-Konfigurationsdateien verweisen, um Sie bei der Konfiguration von Greengrass-Komponentenumgebungen zu unterstützen.

Hallo Welt (Python)

Die folgende GDK-CLI-Konfigurationsdatei unterstützt eine Hello World-Komponente, die ein Python-Skript ausführt. Diese Konfigurationsdatei verwendet das `zip` Build-System, um das Python-Skript der Komponente in eine ZIP-Datei zu verpacken, die die GDK-CLI als Artefakt hochlädt.

```
{
  "component": {
    "com.example.PythonHelloWorld": {
      "author": "Amazon",
      "version": "NEXT_PATCH",
      "build": {
        "build_system" : "zip",
        "options": {
           "excludes": [".*"]
        }
      },
      "publish": {
        "bucket": "greengrass-component-artifacts",
        "region": "us-west-2",
        "options": {
           "file_upload_args": {
              "Metadata": {
                 "some-key": "some-value"
              }
           }
        }
      }
    },
  "test-e2e":{
```

```
    "build":{
        "build_system": "maven"
    },
    "gtf_version": "1.1.0",
    "gtf_options": {
         "tags": "Sample"
     }
  },
  "gdk_version": "1.6.1"
  }
}
```

Hallo Welt (Java)

Die folgende GDK-CLI-Konfigurationsdatei unterstützt eine Hello World-Komponente, die eine Java-Anwendung ausführt. Diese Konfigurationsdatei verwendet das `maven` Build-System, um den Java-Quellcode der Komponente in eine JAR-Datei zu verpacken, die die GDK-CLI als Artefakt hochlädt.

```
{
  "component": {
    "com.example.JavaHelloWorld": {
      "author": "Amazon",
      "version": "NEXT_PATCH",
      "build": {
        "build_system" : "maven"
      },
      "publish": {
        "bucket": "greengrass-component-artifacts",
        "region": "us-west-2",
        "options": {
           "file_upload_args": {
              "Metadata": {
                 "some-key": "some-value"
              }
           }
        }
      }
  },
  "test-e2e":{
    "build":{
        "build_system": "maven"
    },
    "gtf_version": "1.1.0",
```

```
    "gtf_options": {
         "tags": "Sample"
     }
  },
  "gdk_version": "1.6.1"
  }
}
```

Community-Komponenten

Mehrere Community-Komponenten im Greengrass Software Catalog verwenden die GDK-CLI. Sie können die GDK-CLI-Konfigurationsdateien in den Repositorys dieser Komponenten untersuchen.

So zeigen Sie die GDK-CLI-Konfigurationsdateien der Community-Komponenten an

1. Führen Sie den folgenden Befehl aus, um die Community-Komponenten aufzulisten, die die GDK-CLI verwenden.

   ```
   gdk component list --repository
   ```

   Die Antwort listet den Namen des GitHub Repositorys für jede Community-Komponente auf, die die GDK-CLI verwendet. Jedes Repository ist in der `awslabs` Organisation vorhanden.

   ```
   [2022-02-22 17:27:31] INFO - Listing all the available component repositories from
    Greengrass Software Catalog.
   [2022-02-22 17:27:31] INFO - Found '6' component repositories to display.
   1. aws-greengrass-labs-database-influxdb
   2. aws-greengrass-labs-telemetry-influxdbpublisher
   3. aws-greengrass-labs-dashboard-grafana
   4. aws-greengrass-labs-dashboard-influxdb-grafana
   5. aws-greengrass-labs-local-web-server
   6. aws-greengrass-labs-lookoutvision-gstreamer
   ```

2. Öffnen Sie das GitHub Repository einer Community-Komponente unter der folgenden URL. Ersetzen Sie durch *`community-component-name`* den Namen einer Community-Komponente aus dem vorherigen Schritt.

   ```
   https://github.com/awslabs/community-component-name
   ```

Greengrass-Befehlszeilenschnittstelle

Mit der Greengrass Command Line Interface (CLI) können Sie mit AWS IoT Greengrass Core auf Ihrem Gerät interagieren, um Komponenten lokal zu entwickeln und Probleme zu debuggen. Sie können beispielsweise die Greengrass-CLI verwenden, um eine lokale Bereitstellung zu erstellen und eine Komponente auf dem Core-Gerät neu zu starten.

Stellen Sie die Greengrass-CLI-Komponente (`aws.greengrass.Cli`) bereit, um die Greengrass-CLI auf Ihrem Core-Gerät zu installieren.

> **Important**
>
> Wir empfehlen, diese Komponente nur in Entwicklungsumgebungen und nicht in Produktionsumgebungen zu verwenden. Diese Komponente bietet Zugriff auf Informationen und Vorgänge, die Sie normalerweise in einer Produktionsumgebung nicht benötigen. Folgen Sie dem Prinzip der geringsten Berechtigung, indem Sie diese Komponente nur auf -Core-Geräten bereitstellen, auf denen Sie sie benötigen.

Themen

Installieren Sie die Greengrass-CLI

Sie können die Greengrass-CLI auf eine der folgenden Arten installieren:

- Verwenden Sie das `--deploy-dev-tools` Argument, wenn Sie die AWS IoT Greengrass Core-Software zum ersten Mal auf Ihrem Gerät einrichten. Sie müssen auch angeben`--provision true`, ob dieses Argument angewendet werden soll.
- Stellen Sie die Greengrass CLI-Komponente (`aws.greengrass.Cli`) auf Ihrem Gerät bereit.

In diesem Abschnitt werden die Schritte zur Bereitstellung der Greengrass-CLI-Komponente beschrieben. Hinweise zur Installation der Greengrass-CLI bei der Ersteinrichtung finden Sie unterTutorial: Erste Schritte mit AWS IoT Greengrass V2.

Voraussetzungen

Um die Greengrass CLI-Komponente bereitzustellen, müssen Sie die folgenden Anforderungen erfüllen:

- AWS IoT Greengrass Kernsoftware, die auf Ihrem Kerngerät installiert und konfiguriert ist. Weitere Informationen finden Sie unter Tutorial: Erste Schritte mit AWS IoT Greengrass V2.
- Um das AWS CLI zur Bereitstellung der Greengrass CLI zu verwenden, müssen Sie das AWS CLI installiert und konfiguriert haben. Weitere Informationen finden Sie unter Konfigurieren der AWS CLI im AWS Command Line Interface -Leitfaden.
- Sie müssen autorisiert sein, die Greengrass-CLI zu verwenden, um mit der AWS IoT Greengrass Core-Software zu interagieren. Führen Sie einen der folgenden Schritte aus, um die Greengrass-CLI zu verwenden:
  - Verwenden Sie den Systembenutzer, der die AWS IoT Greengrass Core-Software ausführt.
  - Verwenden Sie einen Benutzer mit Root- oder Administratorrechten. Auf Linux-Core-Geräten können Sie diese Option verwenden, um `sudo` Root-Rechte zu erhalten.
  - Verwenden Sie einen Systembenutzer, der zu einer Gruppe gehört, die Sie bei der Bereitstellung der Komponente in den `AuthorizedWindowsGroups` Konfigurationsparametern `AuthorizedPosixGroups` oder angeben. Weitere Informationen finden Sie unter Konfiguration der Greengrass-CLI-Komponente.

Stellen Sie die Greengrass-CLI-Komponente bereit

Gehen Sie wie folgt vor, um die Greengrass CLI-Komponente auf Ihrem Kerngerät bereitzustellen:

So stellen Sie die Greengrass-CLI-Komponente (Konsole) bereit

1. Melden Sie sich an der AWS IoT Greengrass -Konsole an.
2. Wählen Sie im Navigationsmenü Komponenten.
3. Wählen Sie auf der Seite Komponenten auf der Registerkarte Öffentliche Komponenten die Option `aws.greengrass.Cli` aus.
4. Wählen Sie auf der aws.greengrass.Cli Seite Bereitstellen aus.
5. Wählen Sie unter Zur Bereitstellung hinzufügen die Option Neue Bereitstellung erstellen aus.
6. Wählen Sie auf der Seite Ziel angeben unter Bereitstellungsziele in der Liste Zielname die Greengrass-Gruppe aus, für die Sie die Bereitstellung durchführen möchten, und klicken Sie auf Weiter.

7. Vergewissern Sie sich auf der Seite „Komponenten auswählen“, dass die aws.greengrass.CliKomponente ausgewählt ist, und klicken Sie auf Weiter.
8. Behalten Sie auf der Seite Komponenten konfigurieren die Standardkonfigurationseinstellungen bei und wählen Sie Weiter aus.
9. Behalten Sie auf der Seite Erweiterte Einstellungen konfigurieren die Standardkonfigurationseinstellungen bei und wählen Sie Weiter.
10. Klicken Sie auf der Seite „Überprüfen“ auf Bereitstellen

So stellen Sie die Greengrass-CLI-Komponente bereit ()AWS CLI

1. Erstellen Sie auf Ihrem Gerät eine `deployment.json` Datei, um die Bereitstellungskonfiguration für die Greengrass CLI-Komponente zu definieren. Diese Datei sollte wie folgt aussehen:

```
{
  "targetArn":"targetArn",
  "components": {
    "aws.greengrass.Cli": {
      "componentVersion": "2.12.6",
      "configurationUpdate": {
        "merge": "{\"AuthorizedPosixGroups\":\"<group1>,<group2>,...,<groupN>\",
\"AuthorizedWindowsGroups\":\"<group1>,<group2>,...,<groupN>\"}"
      }
    }
  }
}
```

   - Ersetzen Sie im `target` Feld *`targetArn`* durch den Amazon-Ressourcennamen (ARN) des Objekts oder der Objektgruppe, auf die die Bereitstellung ausgerichtet werden soll, und zwar im folgenden Format:
     - Objekt: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`thingName`*
     - Objektgruppe: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thinggroup/`*`thingGroupName`*
   - Geben Sie im `aws.greengrass.Cli` Komponentenobjekt die Werte wie folgt an:

     `version`

     Die Version der Greengrass-CLI-Komponente.

`configurationUpdate.AuthorizedPosixGroups`

(Optional) Eine Zeichenfolge, die eine durch Kommas getrennte Liste von Systemgruppen enthält. Sie autorisieren diese Systemgruppen, die Greengrass-CLI für die Interaktion mit der AWS IoT Greengrass Core-Software zu verwenden. Sie können Gruppennamen oder Gruppen-IDs angeben. `group1,1002,group3`Autorisiert beispielsweise drei Systemgruppen (`group1`, und`group3`)`1002`, die Greengrass-CLI zu verwenden.

Wenn Sie keine zu autorisierenden Gruppen angeben, können Sie die Greengrass-CLI als Root-Benutzer (`sudo`) oder als Systembenutzer verwenden, der die AWS IoT Greengrass Core-Software ausführt.

`configurationUpdate.AuthorizedWindowsGroups`

(Optional) Eine Zeichenfolge, die eine durch Kommas getrennte Liste von Systemgruppen enthält. Sie autorisieren diese Systemgruppen, die Greengrass-CLI für die Interaktion mit der AWS IoT Greengrass Core-Software zu verwenden. Sie können Gruppennamen oder Gruppen-IDs angeben. `group1,1002,group3`Autorisiert beispielsweise drei Systemgruppen (`group1`, und`group3`)`1002`, die Greengrass-CLI zu verwenden.

Wenn Sie keine zu autorisierenden Gruppen angeben, können Sie die Greengrass-CLI als Administrator oder als Systembenutzer verwenden, der die AWS IoT Greengrass Core-Software ausführt.

2. Führen Sie den folgenden Befehl aus, um die Greengrass-CLI-Komponente auf dem Gerät bereitzustellen:

```
$ aws greengrassv2 create-deployment --cli-input-json file://path/
to/deployment.json
```

Während der Installation fügt die Komponente einen symbolischen Link zu `greengrass-cli` dem */greengrass/v2*`/bin` Ordner auf Ihrem Gerät hinzu, und Sie führen die Greengrass-CLI von diesem Pfad aus aus. Um die Greengrass-CLI ohne ihren absoluten Pfad auszuführen, fügen Sie Ihren */greengrass/v2*`/bin` Ordner zu Ihrer PATH-Variablen hinzu. Führen Sie den folgenden Befehl aus, um die Greengrass CLI-Installation zu überprüfen:

Linux or Unix

```
/greengrass/v2/bin/greengrass-cli help
```

Windows

```
C:\greengrass\v2\bin\greengrass-cli help
```

Die Ausgabe sollte folgendermaßen aussehen:

```
Usage: greengrass-cli [-hV] [--ggcRootPath=<ggcRootPath>] [COMMAND]
Greengrass command line interface

      --ggcRootPath=<ggcRootPath>
                  The AWS IoT Greengrass V2 root directory.
  -h, --help      Show this help message and exit.
  -V, --version   Print version information and exit.
Commands:
  help                Show help information for a command.
  component           Retrieve component information and stop or restart
                        components.
  deployment          Create local deployments and retrieve deployment status.
  logs                Analyze Greengrass logs.
  get-debug-password  Generate a password for use with the HTTP debug view
                        component.
```

Wenn das `greengrass-cli` nicht gefunden wird, konnte die Greengrass-CLI bei der Bereitstellung möglicherweise nicht installiert werden. Weitere Informationen finden Sie unter Problembehebung AWS IoT Greengrass V2.

Greengrass-CLI-Befehle

Die Greengrass-CLI bietet eine Befehlszeilenschnittstelle für die lokale Interaktion mit Ihrem AWS IoT Greengrass Core-Gerät. Greengrass-CLI-Befehle verwenden das folgende Format.

```
$ greengrass-cli <command> <subcommand> [arguments]
```

Standardmäßig interagiert die `greengrass-cli` ausführbare Datei im */greengrass/v2/bin/* Ordner mit der Version der AWS IoT Greengrass Core-Software, die im */greengrass/v2* Ordner ausgeführt wird. Wenn Sie eine ausführbare Datei aufrufen, die nicht an diesem Speicherort platziert ist, oder wenn Sie mit AWS IoT Greengrass Core-Software an einem anderen Speicherort interagieren möchten, müssen Sie eine der folgenden Methoden verwenden, um explizit den Stammpfad der AWS IoT Greengrass Core-Software anzugeben, mit der Sie interagieren möchten:

- Legen Sie die Umgebungsvariable `GGC_ROOT_PATH` auf */greengrass/v2* fest.
- Fügen Sie Ihrem Befehl das `--ggcRootPath` */greengrass/v2* Argument hinzu, wie im folgenden Beispiel gezeigt.

```
greengrass-cli --ggcRootPath /greengrass/v2 <command> <subcommand> [arguments]
```

Sie können die folgenden Argumente mit jedem Befehl verwenden:

- Verwenden Sie `--help` für Informationen zu einem bestimmten Greengrass-CLI-Befehl.
- Verwenden Sie `--version` für Informationen zur Greengrass-CLI-Version.

Dieser Abschnitt beschreibt die Greengrass-CLI-Befehle und enthält Beispiele für diese Befehle. Die Synopsis für jeden Befehl zeigt seine Argumente und ihre Verwendung. Optionale Argumente werden in eckigen Klammern angezeigt.

Verfügbare Befehle

- Komponente
- Bereitstellung
- Protokolle
- get-debug-password

Komponente

Verwenden Sie den `component` Befehl, um mit lokalen Komponenten auf Ihrem Kerngerät zu interagieren.

Unterbefehle

- details
- auflisten
- Neustart
- stop

details

Rufen Sie die Version, den Status und die Konfiguration einer Komponente ab.

Syntax

```
greengrass-cli component details --name <component-name>
```

Argumente

`--name,-n`. Der Name der Komponente.

Ausgabe

Das folgende Beispiel zeigt die Ausgabe, die erzeugt wird, wenn Sie diesen Befehl ausführen.

```
$ sudo greengrass-cli component details --name MyComponent

Component Name: MyComponent
Version: 1.0.0
State: RUNNING
Configuration: null
```

auflisten

Rufen Sie den Namen, die Version, den Status und die Konfiguration jeder auf dem Gerät installierten Komponente ab.

Syntax

```
greengrass-cli component list
```

Argumente

Keine

Ausgabe

Das folgende Beispiel zeigt die Ausgabe, die erzeugt wird, wenn Sie diesen Befehl ausführen.

```
$ sudo greengrass-cli component list

Components currently running in Greengrass:
```

```
Component Name: FleetStatusService
Version: 0.0.0
State: RUNNING
Configuration: {"periodicUpdateIntervalSec":86400.0}
Component Name: UpdateSystemPolicyService
Version: 0.0.0
State: RUNNING
Configuration: null
Component Name: aws.greengrass.Nucleus
Version: 2.0.0
State: FINISHED
Configuration: {"awsRegion":"region","runWithDefault":
{"posixUser":"ggc_user:ggc_group"},"telemetry":{}}
Component Name: DeploymentService
Version: 0.0.0
State: RUNNING
Configuration: null
Component Name: TelemetryAgent
Version: 0.0.0
State: RUNNING
Configuration: null
Component Name: aws.greengrass.Cli
Version: 2.0.0
State: RUNNING
Configuration: {"AuthorizedPosixGroups":"ggc_user"}
```

Neustart

Starten Sie die Komponenten neu.

Syntax

```
greengrass-cli component restart --names <component-name>,...
```

Argumente

`--names,-n`. Der Name der Komponente. Es ist mindestens ein Komponentenname erforderlich. Sie können zusätzliche Komponentennamen angeben, indem Sie jeden Namen durch ein Komma trennen.

Ausgabe

Keine

stop

Beenden Sie die Ausführung von Komponenten.

Syntax

```
greengrass-cli component stop --names <component-name>,...
```

Argumente

`--names,-n`. Der Name der Komponente. Es ist mindestens ein Komponentenname erforderlich. Sie können bei Bedarf zusätzliche Komponentennamen angeben, indem Sie die einzelnen Namen durch ein Komma trennen.

Ausgabe

Keine

Bereitstellung

Verwenden Sie den `deployment` Befehl , um mit lokalen Komponenten auf Ihrem Core-Gerät zu interagieren.

Verwenden Sie den `status` Unterbefehl , um den Fortschritt einer lokalen Bereitstellung zu überwachen. Sie können den Fortschritt einer lokalen Bereitstellung nicht mithilfe der Konsole überwachen.

Unterbefehle

- create
- Abbrechen
- auflisten
- Status

create

Erstellen oder aktualisieren Sie eine lokale Bereitstellung mit bestimmten Komponentenrezepten, Artefakten und Laufzeitargumenten.

Syntax

```
greengrass-cli deployment create
    --recipeDir path/to/component/recipe
    [--artifactDir path/to/artifact/folder ]
    [--update-config {component-configuration}]
    [--groupId <thing-group>]
    [--merge "<component-name>=<component-version>"]...
    [--runWith "<component-name>:posixUser=<user-name>[:<group-name>]"]...
    [--systemLimits "{component-system-resource-limits}]"]...
    [--remove <component-name>,...]
    [--failure-handling-policy <policy name[ROLLBACK, DO_NOTHING]>]
```

Argumente

- `--recipeDir`, `-r`. Der vollständige Pfad zu dem Ordner, der die Komponentenrezeptdateien enthält.
- `--artifactDir`, `-a`. Der vollständige Pfad zu dem Ordner, der die Artefaktdateien enthält, die Sie in Ihre Bereitstellung aufnehmen möchten. Der Ordner Artefakte muss die folgende Verzeichnisstruktur enthalten:

  ```
  /path/to/artifact/folder/<component-name>/<component-version>/<artifacts>
  ```

- `--update-config`, `-c`. Die Konfigurationsargumente für die Bereitstellung, bereitgestellt als JSON-Zeichenfolge oder JSON-Datei. Die JSON-Zeichenfolge sollte das folgende Format haben:

  ```
  { \
    "componentName": { \
      "MERGE": {"config-key": "config-value"}, \
      "RESET": ["path/to/reset/"] \
    } \
  }
  ```

  `MERGE` Bei und `RESET` wird zwischen Groß- und Kleinschreibung unterschieden und sie müssen in Großbuchstaben geschrieben werden.

- `--groupId`, `-g`. Die Ziel-Objektgruppe für die Bereitstellung.
- `--merge`, `-m`. Der Name und die Version der Zielkomponente, die Sie hinzufügen oder aktualisieren möchten. Sie müssen die Komponenteninformationen im Format angeben`<component>`=`<version>`. Verwenden Sie ein separates Argument für jede

anzugebende zusätzliche Komponente. Verwenden Sie bei Bedarf das Argument `--runWith` , um die `windowsUser` Informationen `posixUserposixGroup`, und für die Ausführung der Komponente bereitzustellen.

- `--runWith`. Die `posixUser`-, - `posixGroup`und -`windowsUser`Informationen zum Ausführen einer generischen oder Lambda-Komponente. Sie müssen diese Informationen im Format angeben`<component>`:{`posixUser`|`windowsUser`}=`<user>`[:`<=posixGroup>`]. Sie können beispielsweise **`HelloWorld:posixUser=ggc_user:ggc_group`** oder angeben**`HelloWorld:windowsUser=ggc_user`**. Verwenden Sie für jede anzugebende zusätzliche Option ein separates Argument.

  Weitere Informationen finden Sie unter Konfigurieren Sie den Benutzer, der die Komponenten ausführt.

- `--systemLimits`. Die Systemressourcenlimits, die für generische und nicht containerisierte Prozesse von Lambda-Komponenten auf dem Core-Gerät gelten sollen. Sie können die maximale CPU- und RAM-Auslastung konfigurieren, die die Prozesse jeder Komponente verwenden können. Geben Sie ein serialisiertes JSON-Objekt oder einen Dateipfad zu einer JSON-Datei an. Das JSON-Objekt muss das folgende Format haben.

  ```
  {  \
    "componentName": { \
      "cpus": cpuTimeLimit, \
      "memory": memoryLimitInKb \
    } \
  }
  ```

  Sie können die folgenden Systemressourcenlimits für jede Komponente konfigurieren:

  - `cpus` – Die maximale CPU-Zeit, die die Prozesse dieser Komponente auf dem Core-Gerät verwenden können. Die gesamte CPU-Zeit eines Core-Geräts entspricht der Anzahl der CPU-Kerne des Geräts. Auf einem Core-Gerät mit 4 CPU-Kernen können Sie diesen Wert beispielsweise auf setzen, 2 um die Prozesse dieser Komponente auf eine Auslastung von 50 Prozent jedes CPU-Kerns zu beschränken. Auf einem Gerät mit 1 CPU-Kern können Sie diesen Wert auf setzen, `0.25` um die Prozesse dieser Komponente auf eine CPU-Auslastung von 25 Prozent zu beschränken. Wenn Sie diesen Wert auf eine Zahl festlegen, die größer als die Anzahl der CPU-Kerne ist, begrenzt die AWS IoT Greengrass Core-Software die CPU-Auslastung der Komponente nicht.
  - `memory` – Die maximale Menge an RAM (in Kilobyte), die die Prozesse dieser Komponente auf dem Core-Gerät verwenden können.

Weitere Informationen finden Sie unter Konfigurieren Sie die Systemressourcenlimits für Komponenten.

Diese Funktion ist für v2.4.0 und höher der Greengrass-Kernkomponente und der Greengrass-CLI auf Linux-Kerngeräten verfügbar. unterstützt diese Funktion derzeit AWS IoT Greengrass nicht auf Windows-Kerngeräten.

- `--remove`. Der Name der Zielkomponente, die Sie aus einer lokalen Bereitstellung entfernen möchten. Um eine Komponente zu entfernen, die aus einer Cloud-Bereitstellung zusammengeführt wurde, müssen Sie die Gruppen-ID der Zielgruppe im folgenden Format angeben:

  Greengrass nucleus v2.4.0 and later

  ```
  --remove <component-name> --groupId <group-name>
  ```

  Earlier than v2.4.0

  ```
  --remove <component-name> --groupId thinggroup/<group-name>
  ```

- `--failure-handling-policy`. Definiert die Aktion, die ausgeführt wird, wenn eine Bereitstellung fehlschlägt. Es gibt zwei Aktionen, die Sie angeben können:
  - `ROLLBACK` –
  - `DO_NOTHING` –

  Diese Funktion ist für v2.11.0 und höher der verfügbarGrüngraskern.

Ausgabe

Das folgende Beispiel zeigt die Ausgabe, die erzeugt wird, wenn Sie diesen Befehl ausführen.

```
$ sudo greengrass-cli deployment create \
    --merge MyApp1=1.0.0 \
    --merge MyApp2=1.0.0 --runWith MyApp2:posixUser=ggc_user \
    --remove MyApp3 \
    --recipeDir recipes/ \
    --artifactDir artifacts/

Local deployment has been submitted! Deployment Id: 44d89f46-1a29-4044-
ad89-5151213dfcbc
```

Abbrechen

Bricht die angegebene Bereitstellung ab.

Syntax

```
greengrass-cli deployment cancel
    -i <deployment-id>
```

Argumente

`-i`. Die eindeutige Kennung der abzubrechenden Bereitstellung. Die Bereitstellungs-ID wird in der Ausgabe des `create` Befehls zurückgegeben.

Output

- None

auflisten

Rufen Sie den Status der letzten 10 lokalen Bereitstellungen ab.

Syntax

```
greengrass-cli deployment list
```

Argumente

None

Ausgabe

Das folgende Beispiel zeigt die Ausgabe, die erzeugt wird, wenn Sie diesen Befehl ausführen. Abhängig vom Status Ihrer Bereitstellung zeigt die Ausgabe einen der folgenden Statuswerte an: `IN_PROGRESS`SUCCEEDED, oder `FAILED`.

```
$ sudo greengrass-cli deployment list

44d89f46-1a29-4044-ad89-5151213dfcbc: SUCCEEDED
Created on: 6/27/23 11:05 AM
```

Status

Rufen Sie den Status einer bestimmten Bereitstellung ab.

Syntax

```
greengrass-cli deployment status -i <deployment-id>
```

Argumente

`-i`. Die ID der Bereitstellung.

Ausgabe

Das folgende Beispiel zeigt die Ausgabe, die erzeugt wird, wenn Sie diesen Befehl ausführen. Abhängig vom Status Ihrer Bereitstellung zeigt die Ausgabe einen der folgenden Statuswerte an: `IN_PROGRESS`, `SUCCEEDED`, oder `FAILED`.

```
$ sudo greengrass-cli deployment status -i 44d89f46-1a29-4044-ad89-5151213dfcbc

44d89f46-1a29-4044-ad89-5151213dfcbc: FAILED
Created on: 6/27/23 11:05 AM
Detailed Status: <Detailed deployment status>
Deployment Error Stack: List of error codes
Deployment Error Types: List of error types
Failure Cause: Cause
```

Protokolle

Verwenden Sie den `logs` Befehl , um Greengrass-Protokolle auf Ihrem Core-Gerät zu analysieren.

Unterbefehle

- get
- list-keywords
- list-log-files

get

Sammeln, filtern und visualisieren Sie Greengrass-Protokolldateien. Dieser Befehl unterstützt nur JSON-formatierte Protokolldateien. Sie können das Protokollierungsformat in der -Kernkonfiguration angeben.

Syntax

```
greengrass-cli logs get
    [--log-dir path/to/a/log/folder]
    [--log-file path/to/a/log/file]
    [--follow true | false ]
    [--filter <filter> ]
    [--time-window <start-time>,<end-time> ]
    [--verbose ]
    [--no-color ]
    [--before <value> ]
    [--after <value> ]
    [--syslog ]
    [--max-long-queue-size <value> ]
```

Argumente

- `--log-dir`, `-ld`. Der Pfad zum Verzeichnis, das auf Protokolldateien geprüft werden soll, z. B. ***/greengrass/v2*/logs**. Verwenden Sie nicht mit `--syslog`. Verwenden Sie ein separates Argument für jedes anzugebende zusätzliche Verzeichnis. Sie müssen mindestens einen von `--log-dir` oder verwenden`--log-file`. Sie können auch beide Argumente in einem einzigen Befehl verwenden.
- `--log-file`, `-lf`. Die Pfade zu den Protokollverzeichnissen, die Sie verwenden möchten. Verwenden Sie ein separates Argument für jedes anzugebende zusätzliche Verzeichnis. Sie müssen mindestens einen von `--log-dir` oder verwenden`--log-file`. Sie können auch beide Argumente in einem einzigen Befehl verwenden.
- `--follow`, `-fol`. Zeigen Sie Protokollaktualisierungen an, sobald sie auftreten. Die Greengrass-CLI wird weiterhin ausgeführt und liest aus den angegebenen Protokollen. Wenn Sie ein Zeitfenster angeben, stoppt die Greengrass-CLI die Überwachung von Protokollen, nachdem alle Zeitfenster beendet wurden.
- `--filter`, `-f`. Das Schlüsselwort, reguläre Ausdrücke oder Schlüssel-Wert-Paar, das als Filter verwendet werden soll. Geben Sie diesen Wert als Zeichenfolge, als regulären Ausdruck oder als Schlüssel-Wert-Paar an. Verwenden Sie für jeden anzugebenden zusätzlichen Filter ein separates Argument.

Bei der Auswertung werden mehrere Filter, die in einem einzigen Argument angegeben sind, durch ODER-Operatoren getrennt, und Filter, die in zusätzlichen Argumenten angegeben sind, werden mit AND-Operatoren kombiniert. Wenn Ihr Befehl beispielsweise enthält`--filter "installed" --filter "name=alpha,name=beta"`, filtert und zeigt Greengrass CLI Protokollmeldungen an, die sowohl das Schlüsselwort als auch `installed` einen `name` Schlüssel mit den Werten `alpha` oder enthaltenbeta.

- `--time-window`, `-t`. Das Zeitfenster, für das Protokollinformationen angezeigt werden sollen. Sie können sowohl exakte Zeitstempel als auch relative Offsets verwenden. Sie müssen diese Informationen im Format angeben*<begin-time>*,*<end-time>*. Wenn Sie weder die Start- noch die Endzeit angeben, wird der Wert für diese Option standardmäßig auf das aktuelle Systemdatum und die aktuelle Systemzeit festgelegt. Verwenden Sie ein separates Argument für jedes zusätzliche Zeitfenster, um anzugeben.

  Greengrass CLI unterstützt die folgenden Formate für Zeitstempel:

  - `yyyy-MM-DD`, z. B. `2020-06-30`. Die Standardzeit ist 00:00:00, wenn Sie dieses Format verwenden.

    `yyyyMMDD`, z. B. `20200630`. Die Standardzeit ist 00:00:00, wenn Sie dieses Format verwenden.

    `HH:mm:ss`, z. B. `15:30:45`. Das Datum ist standardmäßig das aktuelle Systemdatum, wenn Sie dieses Format verwenden.

    `HH:mm:ssSSS`, z. B. `15:30:45`. Das Datum ist standardmäßig das aktuelle Systemdatum, wenn Sie dieses Format verwenden.

    `YYYY-MM-DD'T'HH:mm:ss'Z'`, z. B. `2020-06-30T15:30:45Z`.

    `YYYY-MM-DD'T'HH:mm:ss`, z. B. `2020-06-30T15:30:45`.

    `yyyy-MM-dd'T'HH:mm:ss.SSS`, z. B. `2020-06-30T15:30:45.250`.

  Relative Offsets geben einen Offset des Zeitraums von der aktuellen Systemzeit an. Die Greengrass-CLI unterstützt das folgende Format für relative Offsets: `+|-[<`*value*`>h|hr|hours][`*value*`m|min|minutes][`*value*`]s|sec|seconds`.

  Zum Beispiel das folgende Argument zur Angabe eines Zeitfensters zwischen 1 Stunde und 2 Stunden und 15 Minuten, bevor die aktuelle Zeit ist`--time-window -2h15min,-1hr`.

- `--verbose`. Zeigen Sie alle Felder aus den Protokollmeldungen an. Verwenden Sie nicht mit `--syslog`.
- `--no-color`, `-nc`. Entfernen Sie die Farbkodierung. Die Standard-Farbkodierung für Protokollmeldungen verwendet fettroten Text. Unterstützt nur UNIX-ähnliche Terminals, da ANSI-Escape-Sequenzen verwendet werden.
- `--before`, `-b`. Die Anzahl der Zeilen, die vor einem übereinstimmenden Protokolleintrag angezeigt werden sollen. Standard = 0.
- `--after`, `-a`. Die Anzahl der Zeilen, die nach einem übereinstimmenden Protokolleintrag angezeigt werden sollen. Standard = 0.
- `--syslog`. Verarbeiten Sie alle Protokolldateien mit dem von RFC3164 definierten Syslog-Protokoll. Verwenden Sie nicht mit `--log-dir` und `--verbose`. Das Syslog-Protokoll verwendet das folgende Format: `"<$Priority>$Timestamp $Host $Logger ($Class): $Message"`. Wenn Sie keine Protokolldatei angeben, liest die Greengrass-CLI Protokollmeldungen von den folgenden Speicherorten: `/var/log/messages`/`var/log/syslog`, oder `/var/log/system.log`.

  AWS IoT Greengrass unterstützt diese Funktion derzeit nicht auf Windows-Core-Geräten.
- `--max-log-queue-size`, `-m`. Die maximale Anzahl von Protokolleinträgen, die dem Speicher zugewiesen werden sollen. Verwenden Sie diese Option, um die Speichernutzung zu optimieren. Der Standardwert ist 100.

Ausgabe

Das folgende Beispiel zeigt die Ausgabe, die erzeugt wird, wenn Sie diesen Befehl ausführen.

```
$ sudo greengrass-cli logs get --verbose \
    --log-file /greengrass/v2/logs/greengrass.log \
    --filter deployment,serviceName=DeploymentService \
    --filter level=INFO \
    --time-window 2020-12-08T01:11:17,2020-12-08T01:11:22

2020-12-08T01:11:17.615Z [INFO] (pool-2-thread-14)
 com.aws.greengrass.deployment.DeploymentService: Current deployment finished.
 {DeploymentId=44d89f46-1a29-4044-ad89-5151213dfcbc, serviceName=DeploymentService,
 currentState=RUNNING}
2020-12-08T01:11:17.675Z [INFO] (pool-2-thread-14)
 com.aws.greengrass.deployment.IotJobsHelper: Updating status of persisted
 deployment. {Status=SUCCEEDED, StatusDetails={detailed-deployment-
status=SUCCESSFUL}, ThingName=MyThing, JobId=22d89f46-1a29-4044-ad89-5151213dfcbc
```

list-keywords

Zeigen Sie vorgeschlagene Schlüsselwörter an, mit denen Sie Protokolldateien filtern können.

Syntax

```
greengrass-cli logs list-keywords [arguments]
```

Argumente

None

Ausgabe

Die folgenden Beispiele zeigen die Ausgabe, die erzeugt wird, wenn Sie diesen Befehl ausführen.

```
$ sudo greengrass-cli logs list-keywords

Here is a list of suggested keywords for Greengrass log:
level=$str
thread=$str
loggerName=$str
eventType=$str
serviceName=$str
error=$str
```

```
$ sudo greengrass-cli logs list-keywords --syslog

Here is a list of suggested keywords for syslog:
priority=$int
host=$str
logger=$str
class=$str
```

list-log-files

Zeigen Sie Protokolldateien an, die sich in einem angegebenen Verzeichnis befinden.

Syntax

```
greengrass-cli logs list-log-files [arguments]
```

Argumente

`--log-dir`, `-ld`. Der Pfad zum Verzeichnis, das auf Protokolldateien geprüft werden soll.

Ausgabe

Das folgende Beispiel zeigt die Ausgabe, die erzeugt wird, wenn Sie diesen Befehl ausführen.

```
$ sudo greengrass-cli logs list-log-files -ld /greengrass/v2/logs/

/greengrass/v2/logs/aws.greengrass.Nucleus.log
/greengrass/v2/logs/main.log
/greengrass/v2/logs/greengrass.log
Total 3 files found.
```

get-debug-password

Verwenden der`get-debug-password`Befehl, um ein zufällig generiertes Passwort für den[Komponente der lokalen Debug-Konsole](`aws.greengrass.LocalDebugConsole`) enthalten. Das Passwort läuft 8 Stunden nach seiner Generierung ab.

Syntax

```
greengrass-cli get-debug-password
```

Argumente

Keine

Ausgabe

Das folgende Beispiel zeigt die Ausgabe, die beim Ausführen dieses Befehls erzeugt wird.

```
$ sudo greengrass-cli get-debug-password

Username: debug
Password: bEDp3MOHdj8ou2w5de_sCBI2XAaguy3a8XxREXAMPLE
Password expires at: 2021-04-01T17:01:43.921999931-07:00
The local debug console is configured to use TLS security. The certificate is self-
signed so you will need to bypass your web browser's security warnings to open the
 console.
```

```
Before you bypass the security warning, verify that the certificate fingerprint
 matches the following fingerprints.
SHA-256: 15 0B 2C E2 54 8B 22 DE 08 46 54 8A B1 2B 25 DE FB 02 7D 01 4E 4A 56 67 96
 DA A6 CC B1 D2 C4 1B
SHA-1: BC 3E 16 04 D3 80 70 DA E0 47 25 F9 90 FA D6 02 80 3E B5 C1
```

Verwenden Sie das AWS IoT Greengrass Testing Framework

Das Greengrass Testing Framework (GTF) ist eine Sammlung von Bausteinen, die die end-to-end Automatisierung aus Kundensicht unterstützen. GTF verwendet Cucumber als Feature-Treiber. AWS IoT Greengrass verwendet dieselben Bausteine, um Softwareänderungen auf verschiedenen Geräten zu qualifizieren. Weitere Informationen finden Sie unter Greengrass Testing Framework auf Github.

GTF wird mithilfe von Cucumber implementiert, einem Tool zur Durchführung automatisierter Tests, um eine verhaltensgesteuerte Entwicklung (BDD) der Komponenten zu fördern. In Cucumber werden die Funktionen dieses Systems in einem speziellen Dateityp namens beschrieben. `feature` Jede Funktion wird in einem für Menschen lesbaren Format beschrieben, das als Szenarien bezeichnet wird. Dabei handelt es sich um Spezifikationen, die in automatisierte Tests umgewandelt werden können. Jedes Szenario besteht aus einer Reihe von Schritten, die die Interaktionen und Ergebnisse des zu testenden Systems mithilfe einer domänenspezifischen Sprache namens Gherkin definieren. Ein Gherkin-Schritt ist mit dem Programmiercode verknüpft. Dabei wird eine Methode verwendet, die als Schrittdefinition bezeichnet wird und die Spezifikation fest mit dem Testablauf verknüpft. Schrittdefinitionen in GTF werden mit Java implementiert.

Themen

Funktionsweise

AWS IoT Greengrass verteilt das GTF als eigenständiges JAR, das aus mehreren Java-Modulen besteht. Um GTF zum end-to-end Testen von Komponenten zu verwenden, müssen Sie die Tests

in einem Java-Projekt implementieren. Wenn Sie das Test-Standable-JAR als Abhängigkeit in Ihrem Java-Projekt hinzufügen, können Sie die bestehende Funktionalität der GTF nutzen und sie erweitern, indem Sie Ihre eigenen benutzerdefinierten Testfälle schreiben. Um die benutzerdefinierten Testfälle auszuführen, können Sie Ihr Java-Projekt erstellen und das Ziel-JAR mit den unter beschriebenen Konfigurationsoptionen ausführen. Konfigurationsoptionen für das Greengrass Testing Framework

Eigenständiges GTF-JAR

Greengrass verwendet Cloudfront als Maven-Repository, um verschiedene Versionen des GTF-Standalone-JAR zu hosten. Eine vollständige Liste der GTF-Versionen finden Sie unter GTF-Versionen.

GTF Standalone JAR umfasst die folgenden Module. Es ist nicht nur auf diese Module beschränkt. Sie können jede dieser Abhängigkeiten separat in Ihrem Projekt auswählen oder sie alle gleichzeitig in die eigenständige JAR-Test-JAR-Datei aufnehmen.

- `aws-greengrass-testing-resources`: Dieses Modul bietet Abstraktion für die Verwaltung des Lebenszyklus einer AWS Ressource während eines Tests. Sie können dies verwenden, um Ihre benutzerdefinierten AWS Ressourcen mithilfe von `ResourceSpec` Abstraktion zu definieren, sodass GTF sich für Sie um die Erstellung und Entfernung dieser Ressourcen kümmern kann.
- `aws-greengrass-testing-platform`: Dieses Modul bietet Abstraktion auf Plattformebene für das zu testende Gerät während des Testlebenszyklus. Es enthält APIs, die für die plattformunabhängige Interaktion mit dem Betriebssystem verwendet werden, und kann verwendet werden, um die Befehle zu simulieren, die in der Geräte-Shell ausgeführt werden.
- `aws-greengrass-testing-components`: Dieses Modul besteht aus Beispielkomponenten, die zum Testen der Greengrass-Kernfunktionen wie Bereitstellungen, IPC und anderen Funktionen verwendet werden.
- `aws-greengrass-testing-features`: Dieses Modul besteht aus wiederverwendbaren gemeinsamen Schritten und ihren Definitionen, die für Tests in der Greengrass-Umgebung verwendet werden.

Themen

- Tutorial: Verwenden Sie einen Konfidenztest aus der Konfidenztestsuite

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen des GTF beschrieben. Weitere Informationen finden Sie auf der Seite GTF-Veröffentlichungen unter. GitHub

Version	Änderungen
1.2.0	Neue Features • Fügt netzwerkbezogene Schritte zur Konfiguration von MQTT und Internet-Netzwerkkonnektivität während Tests hinzu. • Fügt Schritte zur Systemmetrik hinzu, um die RAM- und CPU-Ausla stung des Geräts zu überwachen. Fehlerkorrekturen und Verbesserungen • Der lokale Bereitstellungsschritt von Greengrass CLI wird wiederholt, bis er erfolgreich ist. • Tests stoppen den Greengrass-Kern auf elegante Weise, anstatt ihn abzutöten. • Fügt eine Verbesserung hinzu, bei der GTF den Endpunkt AWS IoT Credentials abfragt, bis Anmeldeinformationen für das Ding und den Rollen-Alias abgerufen werden können. • Behebt fehlende Artefakte und Rezeptverzeichnisse. Diese Version behebt auch fehlende Komponentenversionen. • Behebt ein Problem, bei dem GTF während der Docker-Image-Berei nigung fehlschlägt, wenn das Docker-Image nicht existiert. • Fügt das Schlüsselwort CURRENT als Version der Komponente hinzu.
1.1.0	Neue Features • Fügt die Möglichkeit hinzu, eine benutzerdefinierte Komponente mit Konfiguration zu installieren. Dies erfordert ein Rezept für die benutzerd efinierte Komponente. • Fügt die Möglichkeit hinzu, eine lokale Bereitstellung mit einer benutzerd efinierten Konfiguration zu aktualisieren.

Version	Änderungen
	Fehlerkorrekturen und Verbesserungen • Behebt das Problem mit der Inkonsistenz der GTF-Version im Logkontext.
1.0.0	Erste Version

Konfigurationsoptionen für das Greengrass Testing Framework

GTF-Konfigurationsoptionen

Mit dem Greengrass Testing Framework (GTF) können Sie bestimmte Parameter während des Starts des konfigurieren end-to-end Testprozess zur Orchestrierung des Testflusses. Sie können diese Konfigurationsoptionen als CLI-Argumente für das eigenständige GTF-JAR angeben.

GTF Version 1.1.0 und höher bietet die folgenden Konfigurationsoptionen.

- `additional-plugins`— (Optional) Zusätzliche Cucumber-Plugins
- `aws-region`— Zielt auf spezifische regionale Endpunkte ab fürAWSDienstleistungen. Standardmäßig ist dasAWSDas SDK entdeckt.
- `credentials-path`— FakultativAWSPfad der Profilanmeldeinformationen. Standardmäßig werden Anmeldeinformationen verwendet, die in der Host-Umgebung gefunden wurden.
- `credentials-path-rotation`— Optionale Rotationsdauer fürAWSAnmeldeinformationen. Die Standardeinstellung ist 15 Minuten oder`PT15M`.
- `csr-path`— Der Pfad für die CSR, mit der das Gerätezertifikat generiert wird.
- `device-mode`— Das zu testende Zielgerät. Standardmäßig ist das lokale Gerät aktiviert.
- `env-stage`— Zielt auf die Bereitstellungsumgebung von Greengrass ab. Standardmäßig ist die Produktion eingestellt.
- `existing-device-cert-arn`— Der ARN eines vorhandenen Zertifikats, das Sie als Gerätezertifikat für Greengrass verwenden möchten.
- `feature-path`— Datei oder Verzeichnis mit zusätzlichen Funktionsdateien. Standardmäßig werden keine zusätzlichen Featuredateien verwendet.
- `gg-cli-version`— Setzt die Version der Greengrass-CLI außer Kraft. Standardmäßig wird der Wert verwendet, der in gefunden wurde`ggc.version`.

- `gg-component-bucket`— Der Name eines vorhandenen Amazon S3-Buckets, der Greengrass-Komponenten enthält.
- `gg-component-overrides`— Eine Liste der Überschreibungen von Greengrass-Komponenten.
- `gg-persist`— Eine Liste von Testelementen, die nach einem Testlauf beibehalten werden sollen. Das Standardverhalten besteht darin, nichts beizubehalten. Zulässige Werte sind:`aws.resources`,`installed.software`, und`generated.files`.
- `gg-runtime`— Eine Liste von Werten, die beeinflussen, wie der Test mit den Testressourcen interagiert. Diese Werte haben Vorrang vor dem`gg.persist`Parameter. Wenn der Standardwert leer ist, wird davon ausgegangen, dass alle Testressourcen nach Testfällen verwaltet werden, einschließlich der installierten Greengrass-Runtime. Zulässige Werte sind:`aws.resources`,`installed.software`, und`generated.files`.
- `ggc-archive`— Der Pfad zur archivierten Greengrass-Nukleus-Komponente.
- `ggc-install-root`— Verzeichnis zur Installation der Greengrass Nucleus-Komponente. Standardmäßig ist dies der Ordner test.temp.path und der Testrun-Ordner.
- `ggc-log-level`— Legt die Greengrass Nucleus-Protokollebene für den Testlauf fest. Die Standardeinstellung ist „INFO“.
- `ggc-tes-rolename`— Die IAM-Rolle, dieAWS IoT GreengrassCore geht davon aus, darauf zuzugreifenAWSDienstleistungen. Wenn eine Rolle mit dem angegebenen Namen nicht existiert, wird eine erstellt und es wird eine Standardzugriffsrichtlinie festgelegt.
- `ggc-trusted-plugins`— Die durch Kommas getrennte Liste der Pfade (auf dem Host) der vertrauenswürdigen Plugins, die zu Greengrass hinzugefügt werden müssen. Um den Pfad auf dem DUT selbst anzugeben, stellen Sie dem Pfad „dut:“ voran
- `ggc-user-name`— Der posixUser-Wert user:group für den Greengrass-Kern. Standardmäßig wird der aktuelle Benutzername verwendet, der angemeldet ist.
- `ggc-version`— Überschreibt die Version der laufenden Greengrass Nucleus-Komponente. Standardmäßig wird der in ggc.archive gefundene Wert verwendet.
- `log-level`— Protokollebene des Testlaufs. Der Standardwert ist „INFO“.
- `parallel-config`— Satz von Batch-Index und Anzahl der Batches als JSON-Zeichenfolge. Der Standardwert des Batch-Index ist 0 und die Anzahl der Batches ist 1.
- `proxy-url`— Konfigurieren Sie alle Tests so, dass der Verkehr über diese URL weitergeleitet wird.
- `tags`— Nur Feature-Tags ausführen. Kann mit '&' überschnitten werden

- `test-id-prefix`— Ein gemeinsames Präfix, das auf alle testspezifischen Ressourcen angewendet wird, einschließlichAWSRessourcennamen und Tags. Die Standardeinstellung ist ein „gg“ -Präfix.
- `test-log-path`— Verzeichnis, das die Ergebnisse des gesamten Testlaufs enthalten wird. Der Standardwert ist „TestResults“.
- `test-results-json`— Markierung, um festzustellen, ob ein resultierender Cucumber-JSON-Bericht generiert und auf die Festplatte geschrieben wird. Standardwert ist „true“.
- `test-results-log`— Markierung, um festzustellen, ob die Konsolenausgabe generiert und auf die Festplatte geschrieben wird. Standardwert "false".
- `test-results-xml`— Markierung, um festzustellen, ob ein resultierender JUnit-XML-Bericht generiert und auf die Festplatte geschrieben wird. Standardwert ist „true“.
- `test-temp-path`— Verzeichnis zum Generieren lokaler Testartefakte. Standardmäßig wird ein zufälliges temporäres Verzeichnis mit dem Präfix gg-testing verwendet.
- `timeout-multiplier`— Für alle Test-Timeouts wird ein Multiplikator bereitgestellt. Standard = 1.0.

Tutorial: Ausführen von end-to-end Tests mit dem Greengrass Testing Framework und dem Greengrass Development Kit

AWS IoT Greengrass Testing Framework (GTF) und Greengrass Development Kit (GDK) bieten Entwicklern Möglichkeiten, end-to-end Tests durchzuführen. Sie können dieses Tutorial abschließen, um ein GDK-Projekt mit einer Komponente zu initialisieren, ein GDK-Projekt mit einem end-to-end Testmodul zu initialisieren und einen benutzerdefinierten Testfall zu erstellen. Nachdem Sie Ihren benutzerdefinierten Testfall erstellt haben, können Sie den Test ausführen.

In diesem Tutorial führen Sie folgende Aufgaben aus:

1. Initialisieren Sie ein GDK-Projekt mit einer Komponente.
2. Initialisieren Sie ein GDK-Projekt mit einem - end-to-end Testmodul.
3. Erstellen Sie einen benutzerdefinierten Testfall.
4. Fügen Sie dem neuen Testfall ein Tag hinzu.
5. Erstellen Sie das Test-JAR.
6. Führen Sie den Test aus.

Themen

Voraussetzungen

Zum Durcharbeiten dieses Tutorials ist Folgendes erforderlich:

- GDK Version 1.3.0 oder höher
- Java
- Maven
- Git

Schritt 1: Initialisieren eines GDK-Projekts mit einer Komponente

- Initialisieren Sie einen leeren Ordner mit einem GDK-Projekt. Laden Sie die in Python implementierte `HelloWorld` Komponente herunter, indem Sie den folgenden Befehl ausführen.

  ```
  gdk component init -t HelloWorld -l python -n HelloWorld
  ```

  Dieser Befehl erstellt ein neues Verzeichnis mit dem Namen `HelloWorld` im aktuellen Verzeichnis.

Schritt 2: Initialisieren eines GDK-Projekts mit einem end-to-end Testmodul

- Mit GDK können Sie die Testmodulvorlage herunterladen, die aus einer Feature- und Schrittimplementierung besteht. Führen Sie den folgenden Befehl aus, um das `HelloWorld` Verzeichnis zu öffnen und das vorhandene GDK-Projekt mit einem Testmodul zu initialisieren.

```
cd HelloWorld
gdk test-e2e init
```

Dieser Befehl erstellt ein neues Verzeichnis mit dem Namen `gg-e2e-tests` innerhalb des `HelloWorld` Verzeichnisses . Dieses Testverzeichnis ist ein Maven-Projekt, das von der eigenständigen Greengrass-Test-JAR abhängig ist.

Schritt 3: Erstellen eines benutzerdefinierten Testfalls

Das Schreiben eines benutzerdefinierten Testfalls besteht aus zwei Schritten: Erstellen Sie eine Feature-Datei mit einem Testszenario und implementieren Sie Schrittdefinitionen. Ein Beispiel für die Erstellung eines benutzerdefinierten Testfalls finden Sie unter Beispiel: Erstellen eines benutzerdefinierten Testfalls. Führen Sie die folgenden Schritte aus, um Ihren benutzerdefinierten Testfall zu erstellen:

1. Erstellen einer Feature-Datei mit einem Testszenario

   Ein Feature beschreibt in der Regel eine bestimmte Funktionalität der Software, die getestet wird. In Cucumber wird jedes Feature als einzelne Feature-Datei mit einem Titel, einer detaillierten Beschreibung und einem oder mehreren Beispielen für bestimmte Fälle angegeben, die als Szenarien bezeichnet werden. Jedes Szenario besteht aus einem Titel, einer detaillierten Beschreibung und einer Reihe von Schritten, die die Interaktionen und erwarteten Ergebnisse definieren. Szenarien werden in einem strukturierten Format geschrieben, das die Schlüsselwörter „Gut“, „Wann“ und „Donn“ verwendet.
2. Schrittdefinitionen implementieren

   Eine Schrittdefinition verknüpft den Gherkin-Schritt in einfacher Sprache mit dem programmatischen Code. Wenn Cucumber einen Gherkin-Schritt in einem Szenario identifiziert, sucht es nach einer übereinstimmenden Schrittdefinition, die ausgeführt werden soll.

Schritt 4: Hinzufügen eines Tags zum neuen Testfall

- Sie können den Funktionen und Szenarien Tags zuweisen, um den Testprozess zu organisieren. Sie können Tags verwenden, um die Teilmengen von Szenarien zu kategorisieren und auch Hooks bedingt für die Ausführung auszuwählen. Funktionen und Szenarien können mehrere Tags haben, die durch ein Leerzeichen getrennt sind.

In diesem Beispiel verwenden wir die -`HelloWorld`Komponente.

Fügen Sie in der Feature-Datei neben dem `@Sample` Tag ein neues Tag mit `@HelloWorld` dem Namen hinzu.

```
@Sample @HelloWorld
Scenario: As a developer, I can create a component and deploy it on my device
....
```

Schritt 5: Erstellen der Test-JAR

1. Erstellen Sie die Komponente. Sie müssen die Komponente erstellen, bevor Sie das Testmodul erstellen.

   ```
   gdk component build
   ```

2. Erstellen Sie das Testmodul mit dem folgenden Befehl. Mit diesem Befehl wird das Test-JAR im `greengrass-build` Ordner erstellt.

   ```
   gdk test-e2e build
   ```

Schritt 6: Ausführen des Tests

Wenn Sie einen benutzerdefinierten Testfall ausführen, automatisiert die GTF den Lebenszyklus des Tests sowie die Verwaltung von Ressourcen, die während des Tests erstellt wurden. Es stellt zunächst ein getestetes Gerät (DUT) als -AWS IoTObjekt bereit und installiert darauf die Greengrass-Kernsoftware. Anschließend wird eine neue Komponente mit dem Namen unter `HelloWorld` Verwendung des in diesem Pfad angegebenen Rezepts erstellt. Die `HelloWorld` Komponente wird dann über eine Greengrass-Objektbereitstellung auf dem Core-Gerät bereitgestellt. Sie wird dann überprüft, wenn die Bereitstellung erfolgreich ist. Der Bereitstellungsstatus ändert sich `COMPLETED` innerhalb von 3 Minuten in , wenn die Bereitstellung erfolgreich ist.

1. Rufen Sie die `gdk-config.json` Datei im Projektverzeichnis auf, um die Tests mit dem -`HelloWorld`Tag abzuzielen. Aktualisieren Sie den `test-e2e` Schlüssel mit dem folgenden Befehl.

   ```
     "test-e2e":{
   ```

```
    "gtf_options" : {
         "tags":"HelloWorld"
     }
  }
```

2. Bevor Sie die Tests ausführen, müssen Sie dem Host-Gerät AWS Anmeldeinformationen bereitstellen. GTF verwendet diese Anmeldeinformationen, um die AWS Ressourcen während des Testprozesses zu verwalten. Stellen Sie sicher, dass die von Ihnen bereitgestellte Rolle über Berechtigungen zur Automatisierung der erforderlichen Operationen verfügt, die im Test enthalten sind.

   Führen Sie die folgenden Befehle aus, um die AWS Anmeldeinformationen bereitzustellen.

   - Linux or Unix

     ```
     export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
     export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
     ```

     Windows Command Prompt (CMD)

     ```
     set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
     set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
     ```

     PowerShell

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

3. Führen Sie den Test mit dem folgenden Befehl aus.

   ```
   gdk test-e2e run
   ```

   Dieser Befehl lädt die neueste Version des Greengrass-Kerns im `greengrass-build` Ordner herunter und führt Tests mit diesem aus. Dieser Befehl zielt auch nur auf die Szenarien mit dem `-HelloWorld`Tag ab und generiert einen Bericht für diese Szenarien. Die AWS Ressourcen, die während dieses Tests erstellt wurden, werden am Ende des Tests verworfen.

Beispiel: Erstellen eines benutzerdefinierten Testfalls

Example

Das heruntergeladene Testmodul im Projekt GDK besteht aus einer Beispielfunktion und einer Schrittimplementierungsdatei.

Im folgenden Beispiel erstellen wir eine Feature-Datei zum Testen der Objektbereitstellungsfunktion der Greengrass-Software. Wir testen die Funktionalität dieser Funktion teilweise mit einem Szenario, das die Bereitstellung einer Komponente über die Greengrass- durchführtAWS Cloud. Dies ist eine Reihe von Schritten, die uns helfen, die Interaktionen und die erwarteten Ergebnisse dieses Anwendungsfalls zu verstehen.

1. Erstellen einer Feature-Datei

   Navigieren Sie zum `gg-e2e-tests/src/main/resources/greengrass/features` Ordner im aktuellen Verzeichnis. Sie finden das Beispiel`component.feature`, das wie im folgenden Beispiel aussieht.

   In dieser Feature-Datei können Sie die Objektbereitstellungsfunktion der Greengrass-Software testen. Sie können die Funktionalität dieses Features teilweise mit einem Szenario testen, das eine Bereitstellung einer Komponente über die Greengrass-Cloud durchführt. Das Szenario besteht aus einer Reihe von Schritten, die dabei helfen, die Interaktionen und die erwarteten Ergebnisse dieses Anwendungsfalls zu verstehen.

   ```
   Feature: Testing features of Greengrassv2 component

   Background:
       Given my device is registered as a Thing
       And my device is running Greengrass

   @Sample
   Scenario: As a developer, I can create a component and deploy it on my device
       When I create a Greengrass deployment with components
           HelloWorld | /path/to/recipe/file
       And I deploy the Greengrass deployment configuration
       Then the Greengrass deployment is COMPLETED on the device after 180 seconds
       And I call my custom step
   ```

   GTF enthält die Schrittdefinitionen aller folgenden Schritte, mit Ausnahme des Schritts mit dem Namen `And I call my custom step`.

2. Schrittdefinitionen implementieren

   GTF Standalone JAR enthält die Schrittdefinitionen aller Schritte mit Ausnahme eines Schritts: `And I call my custom step`. Sie können diesen Schritt im Testmodul implementieren.

   Navigieren Sie zum Quellcode der Testdatei. Sie können Ihren benutzerdefinierten Schritt mithilfe einer Schrittdefinition verknüpfen, indem Sie den folgenden Befehl verwenden.

```
@And("I call my custom step")
public void customStep() {
    System.out.println("My custom step was called ");
}
```

Tutorial: Verwenden Sie einen Konfidenztest aus der Konfidenztestsuite

AWS IoT Greengrass Testing Framework (GTF) und Greengrass Development Kit (GDK) bieten Entwicklern Möglichkeiten, end-to-end Tests durchzuführen. Sie können dieses Tutorial abschließen, um ein GDK-Projekt mit einer Komponente zu initialisieren, ein GDK-Projekt mit einem end-to-end Testmodul zu initialisieren und einen Zuverlässigkeitstest aus der Konfidenztestsuite zu verwenden. Nachdem Sie Ihren benutzerdefinierten Testfall erstellt haben, können Sie den Test ausführen.

Ein Zuverlässigkeitstest ist ein generischer Test, der von Greengrass bereitgestellt wird und das grundlegende Komponentenverhalten validiert. Diese Tests können an spezifischere Komponentenanforderungen angepasst oder erweitert werden.

Für dieses Tutorial verwenden wir eine - HelloWorld Komponente. Wenn Sie eine andere Komponente verwenden, ersetzen Sie die HelloWorld Komponente durch Ihre Komponente.

In diesem Tutorial führen Sie folgende Aufgaben aus:

1. Initialisieren Sie ein GDK-Projekt mit einer Komponente.
2. Initialisieren Sie ein GDK-Projekt mit einem - end-to-end Testmodul.
3. Verwenden Sie einen Test aus der Konfidenztestsuite.
4. Fügen Sie dem neuen Testfall ein Tag hinzu.
5. Erstellen Sie das Test-JAR.
6. Führen Sie den Test aus.

Themen

Voraussetzungen

Zum Durcharbeiten dieses Tutorials ist Folgendes erforderlich:

- GDK Version 1.6.0 oder höher
- Java
- Maven
- Git

Schritt 1: Initialisieren eines GDK-Projekts mit einer Komponente

- Initialisieren Sie einen leeren Ordner mit einem GDK-Projekt. Laden Sie die in Python implementierte `HelloWorld` Komponente herunter, indem Sie den folgenden Befehl ausführen.

  ```
  gdk component init -t HelloWorld -l python -n HelloWorld
  ```

  Dieser Befehl erstellt ein neues Verzeichnis mit dem Namen `HelloWorld` im aktuellen Verzeichnis.

Schritt 2: Initialisieren eines GDK-Projekts mit einem end-to-end Testmodul

- Mit GDK können Sie die Testmodulvorlage herunterladen, die aus einer Feature- und Schrittimplementierung besteht. Führen Sie den folgenden Befehl aus, um das `HelloWorld` Verzeichnis zu öffnen und das vorhandene GDK-Projekt mit einem Testmodul zu initialisieren.

```
cd HelloWorld
gdk test-e2e init
```

Dieser Befehl erstellt ein neues Verzeichnis mit dem Namen `gg-e2e-tests` innerhalb des `HelloWorld` Verzeichnisses . Dieses Testverzeichnis ist ein Maven-Projekt, das von der eigenständigen Greengrass-Test-JAR abhängig ist.

Schritt 3: Verwenden eines Tests aus der Konfidenztestsuite

Das Schreiben eines Konfidenztestfalls besteht aus der Verwendung der bereitgestellten Feature-Datei und bei Bedarf der Änderung der Szenarien. Ein Beispiel für die Verwendung eines Zuverlässigkeitstests finden Sie unter Beispiel: Erstellen eines benutzerdefinierten Testfalls. Gehen Sie wie folgt vor, um einen Zuverlässigkeitstest zu verwenden:

- Verwenden Sie die bereitgestellte Feature-Datei.

  Navigieren Sie zum `gg-e2e-tests/src/main/resources/greengrass/features` Ordner im aktuellen Verzeichnis. Öffnen Sie die `confidenceTest.feature` Beispieldatei, um den Zuverlässigkeitstest zu verwenden.

Schritt 4: Hinzufügen eines Tags zum neuen Testfall

- Sie können den Funktionen und Szenarien Tags zuweisen, um den Testprozess zu organisieren. Sie können Tags verwenden, um die Teilmengen von Szenarien zu kategorisieren und auch Hooks bedingt für die Ausführung auszuwählen. Funktionen und Szenarien können mehrere Tags haben, die durch ein Leerzeichen getrennt sind.

  In diesem Beispiel verwenden wir die -`HelloWorld`Komponente.

  Jedes Szenario ist mit gekennzeichnet`@ConfidenceTest`. Ändern oder fügen Sie Tags hinzu, wenn Sie nur eine Teilmenge der Testsuite ausführen möchten. Jedes Testszenario wird oben in jedem Zuverlässigkeitstest beschrieben. Das Szenario besteht aus einer Reihe von Schritten, die dabei helfen, die Interaktionen und die erwarteten Ergebnisse jedes Testfalls zu verstehen. Sie können diese Tests erweitern, indem Sie Ihre eigenen Schritte hinzufügen oder die vorhandenen ändern.

```
@ConfidenceTest
```

```
Scenario: As a Developer, I can deploy GDK_COMPONENT_NAME to my device and see it
 is working as expected
....
```

Schritt 5: Erstellen des Test-JAR

1. Erstellen Sie die Komponente. Sie müssen die Komponente erstellen, bevor Sie das Testmodul erstellen.

   ```
   gdk component build
   ```

2. Erstellen Sie das Testmodul mit dem folgenden Befehl. Mit diesem Befehl wird das Test-JAR im `greengrass-build` Ordner erstellt.

   ```
   gdk test-e2e build
   ```

Schritt 6: Ausführen des Tests

Wenn Sie einen Zuverlässigkeitstest durchführen, automatisiert die GTF den Lebenszyklus des Tests sowie die Verwaltung von Ressourcen, die während des Tests erstellt wurden. Es stellt zunächst ein getestetes Gerät (DUT) als -AWS IoTObjekt bereit und installiert darauf die Greengrass-Core-Software. Anschließend wird eine neue Komponente mit dem Namen unter `HelloWorld` Verwendung des in diesem Pfad angegebenen Rezepts erstellt. Die `HelloWorld` Komponente wird dann über eine Greengrass-Objektbereitstellung auf dem Core-Gerät bereitgestellt. Sie wird dann überprüft, wenn die Bereitstellung erfolgreich ist. Der Bereitstellungsstatus ändert sich `COMPLETED` innerhalb von 3 Minuten in , wenn die Bereitstellung erfolgreich ist.

1. Gehen Sie zur `gdk-config.json` Datei im Projektverzeichnis, um die Tests mit dem `ConfidenceTest` Tag oder dem in Schritt 4 angegebenen Tag io8u abzuzielen. Aktualisieren Sie den `test-e2e` Schlüssel mit dem folgenden Befehl.

   ```
   "test-e2e":{
     "gtf_options" : {
          "tags":"ConfidenceTest"
      }
   }
   ```

2. Bevor Sie die Tests ausführen, müssen Sie dem Host-Gerät AWS Anmeldeinformationen bereitstellen. GTF verwendet diese Anmeldeinformationen, um die AWS Ressourcen während des Testprozesses zu verwalten. Stellen Sie sicher, dass die von Ihnen bereitgestellte Rolle über Berechtigungen zur Automatisierung der erforderlichen Operationen verfügt, die im Test enthalten sind.

   Führen Sie die folgenden Befehle aus, um die AWS Anmeldeinformationen bereitzustellen.

   - Linux or Unix

     ```
     export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
     export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
     ```

     Windows Command Prompt (CMD)

     ```
     set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
     set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
     ```

     PowerShell

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

3. Führen Sie den Test mit dem folgenden Befehl aus.

   ```
   gdk test-e2e run
   ```

   Dieser Befehl lädt die neueste Version des Greengrass-Kerns im `greengrass-build` Ordner herunter und führt Tests mit diesem aus. Dieser Befehl zielt auch nur auf die Szenarien mit dem -`ConfidenceTest`Tag ab und generiert einen Bericht für diese Szenarien. Die AWS Ressourcen, die während dieses Tests erstellt wurden, werden am Ende des Tests verworfen.

Beispiel: Verwenden eines Zuverlässigkeitstests

Example

Das heruntergeladene Testmodul im Projekt GDK besteht aus einer bereitgestellten Feature-Datei.

Im folgenden Beispiel verwenden wir eine Feature-Datei zum Testen der Objektbereitstellungsfunktion der Greengrass-Software. Wir testen die Funktionalität dieser Funktion teilweise mit einem Szenario, das die Bereitstellung einer Komponente über die Greengrass-durchführtAWS Cloud. Dies ist eine Reihe von Schritten, die uns helfen, die Interaktionen und die erwarteten Ergebnisse dieses Anwendungsfalls zu verstehen.

- Verwenden Sie die bereitgestellte Feature-Datei.

  Navigieren Sie zum `gg-e2e-tests/src/main/resources/greengrass/features` Ordner im aktuellen Verzeichnis. Sie finden das Beispiel`confidenceTest.feature`, das wie im folgenden Beispiel aussieht.

```
Feature: Confidence Test Suite

Background:
    Given my device is registered as a Thing
    And my device is running Greengrass

@ConfidenceTest
Scenario: As a Developer, I can deploy GDK_COMPONENT_NAME to my device and see it
 is working as expected
    When I create a Greengrass deployment with components
      | GDK_COMPONENT_NAME | GDK_COMPONENT_RECIPE_FILE |
      | aws.greengrass.Cli | LATEST                    |
    And I deploy the Greengrass deployment configuration
    Then the Greengrass deployment is COMPLETED on the device after 180 seconds
    # Update component state accordingly. Possible states: {RUNNING, FINISHED,
 BROKEN, STOPPING}
    And I verify the GDK_COMPONENT_NAME component is RUNNING using the greengrass-
cli
```

  Jedes Testszenario wird oben in jedem Zuverlässigkeitstest beschrieben. Das Szenario besteht aus einer Reihe von Schritten, die dabei helfen, die Interaktionen und die erwarteten Ergebnisse jedes Testfalls zu verstehen. Sie können diese Tests erweitern, indem Sie Ihre eigenen Schritte hinzufügen oder die vorhandenen ändern. Jedes der Szenarien enthält Kommentare, die Ihnen helfen, diese Anpassungen vorzunehmen.

Entwickeln von AWS IoT Greengrass Komponenten

Sie können Komponenten auf Ihrem Greengrass-Kerngerät entwickeln und testen. Daher können Sie Ihre AWS IoT Greengrass Software erstellen und iterieren, ohne mit der zu interagierenAWS Cloud. Wenn Sie eine Version Ihrer Komponente abgeschlossen haben, können Sie sie in AWS IoT Greengrass in die Cloud hochladen, sodass Sie und Ihr Team die Komponente auf anderen Geräten in Ihrer Flotte bereitstellen können. Weitere Informationen zum Bereitstellen von Komponenten finden Sie unter Bereitstellen von AWS IoT Greengrass Komponenten auf Geräten.

Jede Komponente besteht aus einem Rezept und Artefakten .

- Rezepte

  Jede Komponente enthält eine Rezeptdatei, die ihre Metadaten definiert. Das Rezept gibt auch die Konfigurationsparameter, Komponentenabhängigkeiten, den Lebenszyklus und die Plattformkompatibilität der Komponente an. Der Komponentenlebenszyklus definiert die Befehle, die die Komponente installieren, ausführen und herunterfahren. Weitere Informationen finden Sie unter AWS IoT Greengrass Referenz zum Komponenten-Rezept.

  Sie können Rezepte im JSON- oder YAML-Format definieren.

- Artefakte

  Komponenten können eine beliebige Anzahl von Artefakten haben, bei denen es sich um Komponenten-Binärdateien handelt. Artefakte können Skripts, kompilierten Code, statische Ressourcen und alle anderen Dateien enthalten, die eine Komponente verbraucht. Komponenten können auch Artefakte aus Komponentenabhängigkeiten verwenden.

AWS IoT Greengrass bietet vorgefertigte Komponenten, die Sie in Ihren Anwendungen verwenden und auf Ihren Geräten bereitstellen können. Sie können beispielsweise die Stream-Manager-Komponente verwenden, um Daten in verschiedene -AWSServices hochzuladen, oder Sie können die CloudWatch Metrikkomponente verwenden, um benutzerdefinierte Metriken in Amazon zu veröffentlichen CloudWatch. Weitere Informationen finden Sie unter AWSVon bereitgestellte Komponenten.

AWS IoT Greengrass kuratiert einen Index von Greengrass-Komponenten, der als Greengrass Software Catalog bezeichnet wird. Dieser Katalog verfolgt Greengrass-Komponenten, die von der Greengrass-Community entwickelt wurden. In diesem Katalog können Sie Komponenten

herunterladen, ändern und bereitstellen, um Ihre Greengrass-Anwendungen zu erstellen. Weitere Informationen finden Sie unter Komponenten der Gemeinschaft.

Die AWS IoT Greengrass Core-Software führt Komponenten als Systembenutzer und -gruppe aus, z. B. `ggc_user` und `ggc_group`, die Sie auf dem Core-Gerät konfigurieren. Das bedeutet, dass Komponenten über die Berechtigungen dieses Systembenutzers verfügen. Wenn Sie einen Systembenutzer ohne Basisverzeichnis verwenden, können Komponenten keine Ausführungsbefehle oder Code verwenden, die ein Basisverzeichnis verwenden. Das bedeutet, dass Sie den `pip install some-library --user` Befehl nicht verwenden können, um Python-Pakete zu installieren. Wenn Sie das Tutorial „Erste Schritte“ befolgt haben, um Ihr Core-Gerät einzurichten, hat Ihr Systembenutzer kein Stammverzeichnis. Weitere Informationen zum Konfigurieren des Benutzers und der Gruppe, die Komponenten ausführen, finden Sie unter Konfigurieren Sie den Benutzer, der die Komponenten ausführt.

> **Note**
>
> AWS IoT Greengrass verwendet semantische Versionen für -Komponenten. Semantische Versionen folgen einem größeren Patch-Nummernsystem. Die -Version `1.0.0` stellt beispielsweise die erste Hauptversion für eine Komponente dar. Weitere Informationen finden Sie in der semantischen Versionsspezifikation .

Themen

Komponentenlebenszyklus

Der Komponentenlebenszyklus definiert die Phasen, die die AWS IoT Greengrass Core-Software zum Installieren und Ausführen von Komponenten verwendet. Jede Phase definiert ein Skript und andere Informationen, die angeben, wie sich die Komponente verhält. Wenn Sie beispielsweise eine Komponente installieren, führt die AWS IoT Greengrass Core-Software das `Install` Lebenszyklusskript für diese Komponente aus. Komponenten auf Core-Geräten haben die folgenden Lebenszyklusstatus:

- `NEW` – Das Rezept und die Artefakte der Komponente werden auf das Core-Gerät geladen, die Komponente ist jedoch nicht installiert. Nachdem eine Komponente in diesen Zustand übergegangen ist, führt sie ihr Installationsskript aus.
- `INSTALLED` – Die Komponente ist auf dem Core-Gerät installiert. Die Komponente wechselt in diesen Zustand, nachdem sie ihr Installationsskript ausgeführt hat.
- `STARTING` – Die Komponente beginnt auf dem Core-Gerät. Die Komponente wechselt in diesen Zustand, wenn sie ihr Startskript ausführt. Wenn der Startvorgang erfolgreich ist, wechselt die Komponente in den -`RUNNING`Status.
- `RUNNING` – Die Komponente wird auf dem Core-Gerät ausgeführt. Die Komponente wechselt in diesen Zustand, wenn sie ihr Ausführungsskript ausführt oder wenn sie aktive Hintergrundprozesse von ihrem Startskript hat.
- `FINISHED` – Die Komponente wurde erfolgreich ausgeführt und ihre Ausführung wurde abgeschlossen.
- `STOPPING` – Die Komponente wird angehalten. Die Komponente wechselt in diesen Zustand, wenn sie ihr Shutdown-Skript ausführt.
- `ERRORED` – Bei der Komponente ist ein Fehler aufgetreten. Wenn die Komponente in diesen Zustand wechselt, führt sie ihr Wiederherstellungsskript aus. Anschließend wird die Komponente neu gestartet, um zu versuchen, zur normalen Verwendung zurückzukehren. Wenn die Komponente ohne erfolgreichen Lauf dreimal in den -`ERRORED`Zustand wechselt, wird die Komponente zu `BROKEN`.
- `BROKEN` – Bei der Komponente sind mehrmals Fehler aufgetreten und sie kann nicht wiederhergestellt werden. Sie müssen die Komponente erneut bereitstellen, um sie zu reparieren.

Komponententypen

Der Komponententyp gibt an, wie die AWS IoT Greengrass Core-Software die Komponente ausführt. Komponenten können die folgenden Typen haben:

- Bol (`aws.greengrass.nucleus`)

  Der Greengrass-Kern ist die Komponente, die die Mindestfunktionalität der AWS IoT Greengrass-Core-Software bietet. Weitere Informationen finden Sie unter Grüngraskern.

- Plugin (`aws.greengrass.plugin`)

  Der Greengrass-Kernus führt eine Plugin-Komponente in derselben Java Virtual Machine (JVM) wie der Kernus aus. Der Kern wird neu gestartet, wenn Sie die Version einer Plugin-Komponente auf einem Core-Gerät ändern. Um Plugin-Komponenten zu installieren und auszuführen, müssen Sie den Greengrass-Kern für die Ausführung als Systemservice konfigurieren. Weitere Informationen finden Sie unter Den Greengrass Nucleus als Systemdienst konfigurieren.

  Mehrere Komponenten, die von bereitgestellt werden, AWS sind Plugin-Komponenten, mit denen sie direkt mit dem Greengrass-Kern verbunden werden können. Plugin-Komponenten verwenden dieselbe Protokolldatei wie der Greengrass-Kern. Weitere Informationen finden Sie unter Überwachen von AWS IoT Greengrass Protokollen.

- Allgemein (`aws.greengrass.generic`)

  Der Greengrass-Kern führt die Lebenszyklusskripte einer generischen Komponente aus, wenn die Komponente einen Lebenszyklus definiert.

  Dieser Typ ist der Standardtyp für benutzerdefinierte Komponenten.

- Lambda (`aws.greengrass.lambda`)

  Der Greengrass-Kern führt eine Lambda-Funktionskomponente mit der Lambda-Launcher-Komponente aus.

  Wenn Sie eine Komponente aus einer Lambda-Funktion erstellen, hat die Komponente diesen Typ. Weitere Informationen finden Sie unter Ausführen von -AWS LambdaFunktionen.

> **Note**
>
> Es wird nicht empfohlen, den Komponententyp in einem Rezept anzugeben. AWS IoT Greengrass legt den Typ für Sie fest, wenn Sie eine Komponente erstellen.

Erstellen von AWS IoT Greengrass Komponenten

Sie können benutzerdefinierte AWS IoT Greengrass Komponenten auf einem lokalen Entwicklungscomputer oder einem Greengrass-Core-Gerät entwickeln. AWS IoT Greengrass stellt die AWS IoT Greengrass Development Kit Command-Line Interface (GDK CLI) bereit, um Sie beim Erstellen, Erstellen und Veröffentlichen von Komponenten aus vordefinierten Komponentenvorlagen und Community-Komponenten zu unterstützen. Sie können auch integrierte Shell-Befehle ausführen, um Komponenten zu erstellen, zu erstellen und zu veröffentlichen. Wählen Sie aus den folgenden Optionen, um benutzerdefinierte Greengrass-Komponenten zu erstellen:

- Verwenden der Greengrass Development Kit CLI

  Verwenden Sie die GDK-CLI, um Komponenten auf einem lokalen Entwicklungscomputer zu entwickeln. Die GDK-CLI erstellt und verpackt Quellcode der Komponenten in ein Rezept und Artefakte, die Sie als private Komponente für den AWS IoT Greengrass Service veröffentlichen können. Sie können die GDK-CLI so konfigurieren, dass die Versions- und Artefakt-URIs der Komponente automatisch aktualisiert werden, wenn Sie die Komponente veröffentlichen, sodass Sie das Rezept nicht jedes Mal aktualisieren müssen. Um eine Komponente mit der GDK-CLI zu entwickeln, können Sie mit einer Vorlage oder einer Community-Komponente aus dem Greengrass Software Catalog beginnen. Weitere Informationen finden Sie unter AWS IoT Greengrass Befehlszeilenschnittstelle des Development Kit.
- Ausführen von integrierten Shell-Befehlen

  Sie können integrierte Shell-Befehle ausführen, um Komponenten auf einem lokalen Entwicklungscomputer oder auf einem Greengrass-Core-Gerät zu entwickeln. Sie verwenden Shell-Befehle zum Kopieren oder Erstellen von Komponentenquellcode in Artefakte. Jedes Mal, wenn Sie eine neue Version einer Komponente erstellen, müssen Sie das Rezept mit der neuen Komponentenversion erstellen oder aktualisieren. Wenn Sie die Komponente im AWS IoT Greengrass Service veröffentlichen, müssen Sie den URI auf jedes Komponentenartefakt im Rezept aktualisieren.

Themen

Erstellen einer Komponente (GDK CLI)

Folgen Sie den Anweisungen in diesem Abschnitt, um eine Komponente mit der GDK-CLI zu erstellen und zu erstellen.

So entwickeln Sie eine Greengrass-Komponente (GDK CLI)

1. Falls noch nicht geschehen, installieren Sie die GDK-CLI auf Ihrem Entwicklungscomputer. Weitere Informationen finden Sie unter Installieren oder Aktualisieren der AWS IoT Greengrass Development-Kit-Befehlszeilenschnittstelle.
2. Wechseln Sie zu dem Ordner, in dem Sie Komponentenordner erstellen möchten.

   Linux or Unix

   ```
   mkdir ~/greengrassv2
   cd ~/greengrassv2
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir %USERPROFILE%\greengrassv2
   cd %USERPROFILE%\greengrassv2
   ```

   PowerShell

   ```
   mkdir ~/greengrassv2
   cd ~/greengrassv2
   ```

3. Wählen Sie eine Komponentenvorlage oder Community-Komponente zum Herunterladen aus. Die GDK-CLI lädt die Vorlage oder die Community-Komponente herunter, sodass Sie von einem funktionierenden Beispiel ausgehen können. Verwenden Sie den Komponentenlistenbefehl, um die Liste der verfügbaren Vorlagen oder Community-Komponenten abzurufen.
   - Führen Sie den folgenden Befehl aus, um Komponentenvorlagen aufzulisten. Jede Zeile in der Antwort enthält den Namen und die Programmiersprache einer Vorlage.

```
gdk component list --template
```

- Führen Sie den folgenden Befehl aus, um Community-Komponenten aufzulisten.

```
gdk component list --repository
```

4. Erstellen Sie einen Komponentenordner, in dem die GDK-CLI die Vorlage oder die Community-Komponente herunterlädt, und ändern Sie ihn. Ersetzen Sie *HelloWorld* durch den Namen der Komponente oder einen anderen Namen, der Ihnen hilft, diesen Komponentenordner zu identifizieren.

   Linux or Unix

   ```
   mkdir HelloWorld
   cd HelloWorld
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir HelloWorld
   cd HelloWorld
   ```

   PowerShell

   ```
   mkdir HelloWorld
   cd HelloWorld
   ```

5. Laden Sie die Vorlage oder die Community-Komponente in den aktuellen Ordner herunter. Verwenden Sie den Befehl component init.

   - Führen Sie den folgenden Befehl aus, um einen Komponentenordner aus einer Vorlage zu erstellen. Ersetzen Sie *HelloWorld* durch den Namen der Vorlage und ersetzen Sie *Python* durch den Namen der Programmiersprache.

     ```
     gdk component init --template HelloWorld --language python
     ```

   - Führen Sie den folgenden Befehl aus, um einen Komponentenordner aus einer Community-Komponente zu erstellen. Ersetzen Sie *ComponentName* durch den Namen der Community-Komponente.

```
gdk component init --repository ComponentName
```

> **Note**
>
> Wenn Sie GDK CLI v1.0.0 verwenden, müssen Sie diesen Befehl in einem leeren Ordner ausführen. Die GDK-CLI lädt die Vorlage oder die Community-Komponente in den aktuellen Ordner herunter.
> Wenn Sie GDK CLI v1.1.0 oder höher verwenden, können Sie das `--name` Argument angeben, um den Ordner anzugeben, in den die GDK CLI die Vorlage oder Community-Komponente herunterlädt. Wenn Sie dieses Argument verwenden, geben Sie einen Ordner an, der nicht vorhanden ist. Die GDK-CLI erstellt den Ordner für Sie. Wenn Sie dieses Argument nicht angeben, verwendet die GDK-CLI den aktuellen Ordner, der leer sein muss.

6. Die GDK-CLI liest aus der GDK-CLI-Konfigurationsdatei mit dem Namen `gdk-config.json`, um Komponenten zu erstellen und zu veröffentlichen. Diese Konfigurationsdatei ist im Stammverzeichnis des Komponentenordners vorhanden. Im vorherigen Schritt wird diese Datei für Sie erstellt. In diesem Schritt aktualisieren Sie `gdk-config.json` mit Informationen zu Ihrer Komponente. Gehen Sie wie folgt vor:

   a. Öffnen Sie `gdk-config.json` in einem Texteditor.

   b. (Optional) Ändern Sie den Namen der Komponente. Der Komponentenname ist der Schlüssel im `component` Objekt.

   c. Ändern Sie den Autor der Komponente.

   d. (Optional) Ändern Sie die Version der Komponente. Geben Sie eines der folgenden Elemente an:

      - `NEXT_PATCH` – Wenn Sie diese Option wählen, legt die GDK-CLI die Version fest, wenn Sie die Komponente veröffentlichen. Die GDK-CLI fragt den AWS IoT Greengrass Service ab, um die neueste veröffentlichte Version der Komponente zu identifizieren. Anschließend wird die Version auf die nächste Patch-Version nach dieser Version festgelegt. Wenn Sie die Komponente noch nicht veröffentlicht haben, verwendet die GDK-CLI Version `1.0.0`.

        Wenn Sie diese Option wählen, können Sie die Greengrass-CLI nicht verwenden, um die Komponente lokal auf Ihrem lokalen Entwicklungscomputer bereitzustellen und zu

testen, auf dem die AWS IoT Greengrass Core-Software ausgeführt wird. Um lokale Bereitstellungen zu aktivieren, müssen Sie stattdessen eine semantische Version angeben.

- Eine semantische Version, z. B. **`1.0.0`**. Semantische Versionen verwenden ein Hauptversions-.Nebenversions-.Patch-Nummerierungssystem. Weitere Informationen finden Sie in der semantischen Versionsspezifikation .

  Wenn Sie Komponenten auf einem Greengrass-Core-Gerät entwickeln, auf dem Sie die Komponente bereitstellen und testen möchten, wählen Sie diese Option. Sie müssen die Komponente mit einer bestimmten Version erstellen, um lokale Bereitstellungen mit der Greengrass-CLI zu erstellen.

e. (Optional) Ändern Sie die Build-Konfiguration für die Komponente. Die Build-Konfiguration definiert, wie die GDK-CLI die Quelle der Komponente in Artefakte umwandelt. Wählen Sie aus den folgenden Optionen für `build_system`:

- `zip` – Verpackt den Ordner der Komponente in eine ZIP-Datei, um als einziges Artefakt der Komponente zu definieren. Wählen Sie diese Option für die folgenden Komponententypen aus:
  - Komponenten, die interpretierte Programmiersprachen verwenden, wie Python oder JavaScript.
  - Komponenten, die andere Dateien als Code verpacken, z. B. Machine-Learning-Modelle oder andere Ressourcen.

  Die GDK-CLI komprimiert den Ordner der Komponente in eine ZIP-Datei mit demselben Namen wie der Komponentenordner. Wenn der Name des Komponentenordners beispielsweise lautet`HelloWorld`, erstellt die GDK-CLI eine ZIP-Datei mit dem Namen `HelloWorld.zip`.

  > **Note**
  >
  > Wenn Sie GDK CLI Version 1.0.0 auf einem Windows-Gerät verwenden, dürfen der Komponentenordner und die ZIP-Dateinamen nur Kleinbuchstaben enthalten.

  Wenn die GDK-CLI den Ordner der Komponente in eine ZIP-Datei komprimiert, überspringt sie die folgenden Dateien:

  - Die Datei `gdk-config.json`

- Die Rezeptdatei (`recipe.json` oder `recipe.yaml`)
- Erstellen von Ordnern, z. B. `greengrass-build`

- `maven` – Führt den `mvn clean package` Befehl aus, um die Quelle der Komponente in Artefakte zu erstellen. Wählen Sie diese Option für Komponenten aus, die Maven verwenden, z. B. Java-Komponenten.

  Auf Windows-Geräten ist diese Funktion für GDK CLI v1.1.0 und höher verfügbar.

- `gradle` – Führt den `gradle build` Befehl aus, um die Quelle der Komponente in Artefakte zu erstellen. Wählen Sie diese Option für Komponenten aus, die Gradle verwenden. Diese Funktion ist für GDK CLI v1.1.0 und höher verfügbar.

  Das `gradle` Build-System unterstützt Kotlin DSL als Build-Datei. Diese Funktion ist für GDK CLI v1.2.0 und höher verfügbar.

- `gradlew` – Führt den `gradlew` Befehl aus, um die Quelle der Komponente in Artefakte zu erstellen. Wählen Sie diese Option für Komponenten aus, die den Gradle Wrapper verwenden.

  Diese Funktion ist für GDK CLI v1.2.0 und höher verfügbar.

- `custom` – Führt einen benutzerdefinierten Befehl aus, um die Quelle der Komponente in ein Rezept und Artefakte zu integrieren. Geben Sie den benutzerdefinierten Befehl im `custom_build_command` Parameter an.

f. Wenn Sie `custom` für angeben`build_system`, fügen Sie dem `build` Objekt `custom_build_command` hinzu. Geben Sie `custom_build_command`in eine einzelne Zeichenfolge oder eine Liste von Zeichenfolgen an, wobei jede Zeichenfolge ein Wort im Befehl ist. Um beispielsweise einen benutzerdefinierten Build-Befehl für eine C++-Komponente auszuführen, können Sie angeben**`["cmake", "--build", "build", "--config", "Release"]`**.

g. Wenn Sie GDK CLI v1.1.0 oder höher verwenden, können Sie das `--bucket` Argument angeben, um den S3-Bucket anzugeben, in den die GDK CLI die Artefakte der Komponente hochlädt. Wenn Sie dieses Argument nicht angeben, lädt die GDK-CLI in den S3-Bucket hoch, dessen Name lautet*`bucket`*`-`*`region`*`-`*`accountId`*, wobei *`Bucket`* und *`Region`* die Werte sind, die Sie in angeben`gdk-config.json`, und *`accountId`* Ihre AWS-Konto -ID ist. Die GDK-CLI erstellt den Bucket, wenn er nicht vorhanden ist.

   Ändern Sie die Veröffentlichungskonfiguration für die Komponente. Gehen Sie wie folgt vor:

i. Geben Sie den Namen des S3-Buckets an, der zum Hosten von Komponentenartefakten verwendet werden soll.

ii. Geben Sie die an AWS-Region , in der die GDK-CLI die Komponente veröffentlicht.

Wenn Sie mit diesem Schritt fertig sind, könnte die `gdk-config.json` Datei ähnlich wie im folgenden Beispiel aussehen.

```
{
  "component": {
    "com.example.PythonHelloWorld": {
      "author": "Amazon",
      "version": "NEXT_PATCH",
      "build": {
        "build_system" : "zip"
      },
      "publish": {
        "bucket": "greengrass-component-artifacts",
        "region": "us-west-2"
      }
    }
  },
  "gdk_version": "1.0.0"
}
```

7. Aktualisieren Sie die Komponentenrezeptdatei mit dem Namen `recipe.yaml` oder `recipe.json`. Gehen Sie wie folgt vor:

   a. Wenn Sie eine Vorlage oder Community-Komponente heruntergeladen haben, die das `zip` Build-System verwendet, überprüfen Sie, ob der Name des ZIP-Artefakts mit dem Namen des Komponentenordners übereinstimmt. Die GDK-CLI komprimiert den Komponentenordner in eine ZIP-Datei mit demselben Namen wie der Komponentenordner. Das Rezept enthält den Namen des ZIP-Artefakts in der Liste der Komponentenartefakte und in Lebenszyklusskripten, die Dateien im ZIP-Artefakt verwenden. Aktualisieren Sie die `Lifecycle` Definitionen `Artifacts` und so, dass der ZIP-Dateiname mit dem Namen des Komponentenordners übereinstimmt. In den folgenden partiellen Rezeptbeispielen wird der ZIP-Dateiname in den `Lifecycle` Definitionen `Artifacts` und hervorgehoben.

JSON

```
{
  ...
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Artifacts": [
        {
          "URI": "s3://{COMPONENT_NAME}/{COMPONENT_VERSION}/HelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ],
      "Lifecycle": {
        "run": "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
      }
    }
  ]
}
```

YAML

```
---
...
Manifests:
  - Platform:
      os: all
    Artifacts:
      - URI: "s3://BUCKET_NAME/COMPONENT_NAME/
COMPONENT_VERSION/HelloWorld.zip"
        Unarchive: ZIP
    Lifecycle:
      run: "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
```

b. (Optional) Aktualisieren Sie die Beschreibung der Komponente, die Standardkonfiguration, Artefakte, Lebenszyklusskripte und die Plattformunterstützung. Weitere Informationen finden Sie unter AWS IoT Greengrass Referenz zum Komponenten-Rezept.

Wenn Sie mit diesem Schritt fertig sind, könnte die Rezeptdatei den folgenden Beispielen ähneln.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "{COMPONENT_NAME}",
  "ComponentVersion": "{COMPONENT_VERSION}",
  "ComponentDescription": "This is a simple Hello World component written in
 Python.",
  "ComponentPublisher": "{COMPONENT_AUTHOR}",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "Message": "World"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Artifacts": [
        {
          "URI": "s3://{COMPONENT_NAME}/{COMPONENT_VERSION}/HelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ],
      "Lifecycle": {
        "run": "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: "2020-01-25"
ComponentName: "{COMPONENT_NAME}"
ComponentVersion: "{COMPONENT_VERSION}"
```

```
ComponentDescription: "This is a simple Hello World component written in
 Python."
ComponentPublisher: "{COMPONENT_AUTHOR}"
ComponentConfiguration:
  DefaultConfiguration:
    Message: "World"
Manifests:
  - Platform:
      os: all
    Artifacts:
      - URI: "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/HelloWorld.zip"
        Unarchive: ZIP
    Lifecycle:
      run: "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
```

8. Entwickeln und erstellen Sie die Greengrass-Komponente. Der Befehl component build erzeugt ein Rezept und Artefakte im `greengrass-build` Ordner im Komponentenordner. Führen Sie den folgenden Befehl aus.

   ```
   gdk component build
   ```

Wenn Sie bereit sind, Ihre Komponente zu testen, verwenden Sie die GDK-CLI, um sie im AWS IoT Greengrass Service zu veröffentlichen. Anschließend können Sie die Komponente auf Greengrass-Core-Geräten bereitstellen. Weitere Informationen finden Sie unter Veröffentlichen Sie Komponenten zur Bereitstellung auf Ihren Kerngeräten.

Erstellen einer Komponente (Shell-Befehle)

Folgen Sie den Anweisungen in diesem Abschnitt, um Rezept- und Artefaktordner zu erstellen, die Quellcode und Artefakte für mehrere Komponenten enthalten.

So entwickeln Sie eine Greengrass-Komponente (Shell-Befehle)

1. Erstellen Sie einen Ordner für Ihre Komponenten mit Unterordnern für Rezepte und Artefakte. Führen Sie die folgenden Befehle auf Ihrem Greengrass-Core-Gerät aus, um diese Ordner zu erstellen und zum Komponentenordner zu wechseln. Ersetzen Sie *`~/greengrassv2`* oder *`%USERPROFILE%\greengrassv2`* durch den Pfad zum Ordner, der für die lokale Entwicklung verwendet werden soll.

Linux or Unix

```
mkdir -p ~/greengrassv2/{recipes,artifacts}
cd ~/greengrassv2
```

Windows Command Prompt (CMD)

```
mkdir %USERPROFILE%\greengrassv2\\recipes, %USERPROFILE%\greengrassv2\\artifacts
cd %USERPROFILE%\greengrassv2
```

PowerShell

```
mkdir ~/greengrassv2/recipes, ~/greengrassv2/artifacts
cd ~/greengrassv2
```

2. Verwenden Sie einen Texteditor, um eine Rezeptdatei zu erstellen, die die Metadaten, Parameter, Abhängigkeiten, den Lebenszyklus und die Plattformfunktionen Ihrer Komponente definiert. Fügen Sie die Komponentenversion in den Rezeptdateinamen ein, damit Sie identifizieren können, welches Rezept welche Komponentenversion widerspiegelt. Sie können das YAML- oder JSON-Format für Ihr Rezept auswählen.

   Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Erstellen der Datei zu verwenden.

   JSON

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

   YAML

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

   > **Note**
   >
   > AWS IoT Greengrass verwendet semantische Versionen für -Komponenten. Semantische Versionen folgen einem größeren Patch-Nummernsystem. Die -Version

`1.0.0` stellt beispielsweise die erste Hauptversion für eine Komponente dar. Weitere Informationen finden Sie in der semantischen Versionsspezifikation .

3. Definieren Sie das Rezept für Ihre Komponente. Weitere Informationen finden Sie unter AWS IoT Greengrass Referenz zum Komponenten-Rezept.

   Ihr Rezept könnte dem folgenden Hello World-Beispielrezept ähneln.

   JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "My first AWS IoT Greengrass component.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "Message": "world"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/hello_world.py {configuration:/
Message}"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "run": "py -3 -u {artifacts:path}/hello_world.py {configuration:/
Message}"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
ComponentDescription: My first AWS IoT Greengrass component.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    Message: world
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      run: |
        python3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
  - Platform:
      os: windows
    Lifecycle:
      run: |
        py -3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
```

Dieses Rezept führt ein Hello World Python-Skript aus, das dem folgenden Beispielskript ähneln könnte.

```
import sys

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

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

4. Erstellen Sie einen Ordner für die zu entwickelnde Komponentenversion. Wir empfehlen Ihnen, für die Artefakte jeder Komponentenversion einen separaten Ordner zu verwenden, damit Sie ermitteln können, welche Artefakte für jede Komponentenversion gelten. Führen Sie den folgenden Befehl aus.

Linux or Unix

```
mkdir -p artifacts/com.example.HelloWorld/1.0.0
```

Windows Command Prompt (CMD)

```
mkdir artifacts/com.example.HelloWorld/1.0.0
```

PowerShell

```
mkdir artifacts/com.example.HelloWorld/1.0.0
```

> **Important**
>
> Sie müssen das folgende Format für den Pfad des Artefaktordners verwenden. Fügen Sie den Komponentennamen und die Version ein, die Sie im Rezept angeben.
>
> ```
> artifacts/componentName/componentVersion/
> ```

5. Erstellen Sie die Artefakte für Ihre Komponente in dem Ordner, den Sie im vorherigen Schritt erstellt haben. Artefakte können Software, Bilder und andere Binärdateien enthalten, die Ihre Komponente verwendet.

   Wenn Ihre Komponente bereit ist, testen Sie Ihre Komponente .

Testen von AWS IoT Greengrass Komponenten mit lokalen Bereitstellungen

Wenn Sie eine Greengrass-Komponente auf einem Core-Gerät entwickeln, können Sie eine lokale Bereitstellung erstellen, um sie zu installieren und zu testen. Führen Sie die Schritte in diesem Abschnitt aus, um eine lokale Bereitstellung zu erstellen.

Wenn Sie die Komponente auf einem anderen Computer entwickeln, z. B. einem lokalen Entwicklungscomputer, können Sie keine lokale Bereitstellung erstellen. Veröffentlichen Sie stattdessen die Komponente im AWS IoT Greengrass Service, damit Sie sie auf Greengrass-Core-Geräten bereitstellen können, um sie zu testen. Weitere Informationen finden Sie unter

Veröffentlichen Sie Komponenten zur Bereitstellung auf Ihren Kerngeräten und Bereitstellen von AWS IoT Greengrass Komponenten auf Geräten.

So testen Sie eine Komponente auf einem Greengrass-Kerngerät

1. Das Core-Gerät protokolliert Ereignisse wie Komponentenaktualisierungen. Sie können diese Protokolldatei anzeigen, um Fehler mit Ihrer Komponente zu erkennen und zu beheben, z. B. ein ungültiges Rezept. Diese Protokolldatei zeigt auch Nachrichten an, die Ihre Komponente auf Standardausgang (stdout) druckt. Wir empfehlen Ihnen, eine zusätzliche Terminalsitzung auf Ihrem Core-Gerät zu öffnen, um neue Protokollmeldungen in Echtzeit zu beobachten. Öffnen Sie eine neue Terminalsitzung, z. B. über SSH, und führen Sie den folgenden Befehl aus, um die Protokolle anzuzeigen. Ersetzen Sie durch */greengrass/v2* den Pfad zum AWS IoT Greengrass Stammordner.

   Linux or Unix

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

   PowerShell

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

   Sie können auch die Protokolldatei für Ihre Komponente anzeigen.

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
   ```

   PowerShell

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

2. Führen Sie in Ihrer ursprünglichen Terminalsitzung den folgenden Befehl aus, um das Core-Gerät mit Ihrer Komponente zu aktualisieren. Ersetzen Sie */greengrass/v2* durch den Pfad zum AWS IoT Greengrass Stammordner und ersetzen Sie *~/greengrassv2* durch den Pfad zu Ihrem lokalen Entwicklungsordner.

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli deployment create \
  --recipeDir ~/greengrassv2/recipes \
  --artifactDir ~/greengrassv2/artifacts \
  --merge "com.example.HelloWorld=1.0.0"
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli deployment create ^
  --recipeDir %USERPROFILE%\greengrassv2\recipes ^
  --artifactDir %USERPROFILE%\greengrassv2\artifacts ^
  --merge "com.example.HelloWorld=1.0.0"
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli deployment create `
  --recipeDir ~/greengrassv2/recipes `
  --artifactDir ~/greengrassv2/artifacts `
  --merge "com.example.HelloWorld=1.0.0"
```

> **Note**
>
> Sie können auch den -`greengrass-cli deployment create`Befehl verwenden, um den Wert der Konfigurationsparameter Ihrer Komponente festzulegen. Weitere Informationen finden Sie unter create.

3. Verwenden Sie den `greengrass-cli deployment status` Befehl , um den Fortschritt der Bereitstellung Ihrer Komponente zu überwachen.

   Unix or Linux

   ```
   sudo /greengrass/v2/bin/greengrass-cli deployment status \
     -i deployment-id
   ```

   Windows Command Prompt (CMD)

   ```
   C:\greengrass\v2\bin\greengrass-cli deployment status ^
   ```

```
  -i deployment-id
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli deployment status `
  -i deployment-id
```

4. Testen Sie Ihre Komponente, während sie auf dem Greengrass-Kerngerät ausgeführt wird. Wenn Sie diese Version Ihrer Komponente abgeschlossen haben, können Sie sie in den AWS IoT Greengrass Service hochladen. Anschließend können Sie die Komponente auf anderen Core-Geräten bereitstellen. Weitere Informationen finden Sie unter Veröffentlichen Sie Komponenten zur Bereitstellung auf Ihren Kerngeräten.

Veröffentlichen Sie Komponenten zur Bereitstellung auf Ihren Kerngeräten

Nachdem Sie eine Version einer Komponente erstellt oder fertiggestellt haben, können Sie sie im AWS IoT Greengrass Service veröffentlichen. Anschließend können Sie es auf Greengrass-Kerngeräten bereitstellen.

Wenn Sie die Greengrass Development Kit CLI (GDK CLI) verwenden, um eine Komponente zu entwickeln und zu erstellen, können Sie die GDK-CLI verwenden, um die Komponente auf der zu veröffentlichen. AWS CloudVerwenden Sie andernfalls die integrierten Shell-Befehle und die, AWS CLI um die Komponente zu veröffentlichen.

Sie können es auch verwenden AWS CloudFormation , um Komponenten und andere AWS Ressourcen aus Vorlagen zu erstellen. Weitere Informationen finden Sie unter Was ist AWS CloudFormation? und AWS::GreengrassV2::ComponentVersionim AWS CloudFormation Benutzerhandbuch.

Themen

- Eine Komponente veröffentlichen (GDK CLI)
- Veröffentlichen Sie eine Komponente (Shell-Befehle)

Eine Komponente veröffentlichen (GDK CLI)

Folgen Sie den Anweisungen in diesem Abschnitt, um eine Komponente mit der GDK-CLI zu veröffentlichen. Die GDK-CLI lädt Build-Artefakte in einen S3-Bucket hoch, aktualisiert die Artefakt-

URIs im Rezept und erstellt die Komponente aus dem Rezept. Sie geben den S3-Bucket und die Region an, die in der GDK-CLI-Konfigurationsdatei verwendet werden sollen.

Wenn Sie GDK CLI v1.1.0 oder höher verwenden, können Sie das `--bucket` Argument angeben, um den S3-Bucket anzugeben, in den die GDK-CLI die Artefakte der Komponente hochlädt. Wenn Sie dieses Argument nicht angeben, lädt die GDK-CLI in den S3-Bucket hoch, dessen Name lautet*`bucket`*`-`*`region`*`-`*`accountId`*, wobei *`Bucket`* und *`Region`* die Werte sind, in denen Sie angeben`gdk-config.json`, und *`accountId Ihre ID`* ist. AWS-Konto Die GDK-CLI erstellt den Bucket, falls er nicht existiert.

> **Important**
>
> Core-Geräterollen erlauben standardmäßig keinen Zugriff auf S3-Buckets. Wenn Sie diesen S3-Bucket zum ersten Mal verwenden, müssen Sie der Rolle Berechtigungen hinzufügen, damit Kerngeräte Komponentenartefakte aus diesem S3-Bucket abrufen können. Weitere Informationen finden Sie unter Erlauben Sie den Zugriff auf S3-Buckets für Komponentenartefakte.

Um eine Greengrass-Komponente zu veröffentlichen (GDK CLI)

1. Öffnen Sie den Komponentenordner in einer Befehlszeile oder einem Terminal.
2. Falls Sie es noch nicht getan haben, erstellen Sie die Greengrass-Komponente. Der Befehl zum Erstellen von Komponenten erzeugt ein Rezept und Artefakte in dem `greengrass-build` Ordner im Komponentenordner. Führen Sie den folgenden Befehl aus.

   ```
   gdk component build
   ```

3. Publizieren Sie die Komponente im AWS Cloud. Der Befehl zum Veröffentlichen von Komponenten lädt die Artefakte der Komponente auf Amazon S3 hoch und aktualisiert das Rezept der Komponente mit der URI jedes Artefakts. Anschließend wird die Komponente im Service erstellt. AWS IoT Greengrass

   > **Note**
   >
   > AWS IoT Greengrass berechnet den Digest jedes Artefakts, wenn Sie die Komponente erstellen. Das bedeutet, dass Sie die Artefaktdateien in Ihrem S3-Bucket nicht ändern können, nachdem Sie eine Komponente erstellt haben. Wenn Sie dies tun, schlagen Bereitstellungen fehl, die diese Komponente enthalten, da der Datei-Digest nicht

übereinstimmt. Wenn Sie eine Artefaktdatei ändern, müssen Sie eine neue Version der Komponente erstellen.

Wenn Sie in der GDK-CLI-Konfigurationsdatei `NEXT_PATCH` für die Komponentenversion angeben, verwendet die GDK-CLI die nächste Patch-Version, die noch nicht im Dienst vorhanden ist. AWS IoT Greengrass

Führen Sie den folgenden Befehl aus.

```
gdk component publish
```

Die Ausgabe gibt Ihnen die Version der Komponente an, die die GDK-CLI erstellt hat.

Nachdem Sie die Komponente veröffentlicht haben, können Sie die Komponente auf Kerngeräten bereitstellen. Weitere Informationen finden Sie unter Bereitstellen von AWS IoT Greengrass Komponenten auf Geräten.

Veröffentlichen Sie eine Komponente (Shell-Befehle)

Gehen Sie wie folgt vor, um eine Komponente mithilfe von Shell-Befehlen und AWS Command Line Interface (AWS CLI) zu veröffentlichen. Wenn Sie eine Komponente veröffentlichen, gehen Sie wie folgt vor:

1. Veröffentlichen Sie Komponentenartefakte in einem S3-Bucket.
2. Fügen Sie die Amazon S3 S3-URI jedes Artefakts zum Komponentenrezept hinzu.
3. Erstellen Sie eine Komponentenversion AWS IoT Greengrass aus dem Komponentenrezept.

> **Note**
>
> Jede Komponentenversion, die Sie hochladen, muss einzigartig sein. Stellen Sie sicher, dass Sie die richtige Komponentenversion hochladen, da Sie sie nach dem Hochladen nicht mehr bearbeiten können.

Sie können diesen Schritten folgen, um eine Komponente von Ihrem Entwicklungscomputer oder Ihrem Greengrass-Core-Gerät aus zu veröffentlichen.

Um eine Komponente zu veröffentlichen (Shell-Befehle)

1. Wenn die Komponente eine Version verwendet, die im AWS IoT Greengrass Service vorhanden ist, müssen Sie die Version der Komponente ändern. Öffnen Sie das Rezept in einem Texteditor, erhöhen Sie die Version und speichern Sie die Datei. Wählen Sie eine neue Version, die die Änderungen widerspiegelt, die Sie an der Komponente vorgenommen haben.

   > **Note**
   >
   > AWS IoT Greengrass verwendet semantische Versionen für Komponenten. Semantische Versionen folgen einem Hauptteil. geringfügig. Patch-Nummernsystem. Version `1.0.0` steht beispielsweise für die erste Hauptversion einer Komponente. Weitere Informationen finden Sie in der semantischen Versionsspezifikation.

2. Wenn Ihre Komponente Artefakte enthält, gehen Sie wie folgt vor:

   a. Veröffentlichen Sie die Artefakte der Komponente in einem S3-Bucket in Ihrem AWS-Konto.

      > **Tip**
      >
      > Wir empfehlen, dass Sie den Namen und die Version der Komponente in den Pfad zum Artefakt im S3-Bucket aufnehmen. Dieses Benennungsschema kann Ihnen helfen, die Artefakte beizubehalten, die in früheren Versionen der Komponente verwendet wurden, sodass Sie frühere Komponentenversionen weiterhin unterstützen können.

      Führen Sie den folgenden Befehl aus, um eine Artefaktdatei in einem S3-Bucket zu veröffentlichen. *Ersetzen Sie DOC-EXAMPLE-BUCKET durch den Namen des Buckets und ersetzen Sie artifacts/com.example. HelloWorld/1.0.0/ artifact.py durch den Pfad zur Artefaktdatei.*

      ```
      aws s3 cp artifacts/com.example.HelloWorld/1.0.0/artifact.py s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/artifact.py
      ```

> **Important**
>
> Core-Geräterollen erlauben standardmäßig keinen Zugriff auf S3-Buckets. Wenn Sie diesen S3-Bucket zum ersten Mal verwenden, müssen Sie der Rolle Berechtigungen hinzufügen, damit Kerngeräte Komponentenartefakte aus diesem S3-Bucket abrufen können. Weitere Informationen finden Sie unter Erlauben Sie den Zugriff auf S3-Buckets für Komponentenartefakte.

b. Fügen Sie dem Komponentenrezept eine Liste mit einem Namen `Artifacts` hinzu, falls diese nicht vorhanden ist. Die `Artifacts` Liste erscheint in jedem Manifest, das die Anforderungen der Komponente auf jeder Plattform definiert, die sie unterstützt (oder die Standardanforderungen der Komponente für alle Plattformen).

c. Fügen Sie jedes Artefakt zur Liste der Artefakte hinzu, oder aktualisieren Sie die URI vorhandener Artefakte. Die Amazon S3 S3-URI besteht aus dem Bucket-Namen und dem Pfad zum Artefaktobjekt im Bucket. Die Amazon S3-URIs Ihrer Artefakte sollten dem folgenden Beispiel ähneln.

```
s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/artifact.py
```

Nachdem Sie diese Schritte abgeschlossen haben, sollte Ihr Rezept eine `Artifacts` Liste enthalten, die wie die folgende aussieht.

JSON

```
{
  ...
  "Manifests": [
    {
      "Lifecycle": {
        ...
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/MyGreengrassComponent/1.0.0/
artifact.py",
          "Unarchive": "NONE"
        }
      ]
```

```
    }
  ]
}
```

> **Note**
>
> Sie können die `"Unarchive": "ZIP"` Option für ein ZIP-Artefakt hinzufügen, um die AWS IoT Greengrass Core-Software so zu konfigurieren, dass das Artefakt bei der Bereitstellung der Komponente entpackt wird.

YAML

```
...
Manifests:
  - Lifecycle:
      ...
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/MyGreengrassComponent/1.0.0/
artifact.py
        Unarchive: NONE
```

> **Note**
>
> Sie können die `Unarchive: ZIP` Option verwenden, um die AWS IoT Greengrass Core-Software so zu konfigurieren, dass ein ZIP-Artefakt entpackt wird, wenn die Komponente bereitgestellt wird. Weitere Informationen zur Verwendung von ZIP-Artefakten in einer Komponente finden Sie in der Rezeptvariablen artifacts:decompressedPath.

Weitere Informationen zu Rezepten finden Sie unter AWS IoT Greengrass Referenz zum Komponenten-Rezept.

3. Verwenden Sie die AWS IoT Greengrass Konsole, um eine Komponente aus der Rezeptdatei zu erstellen.

   Führen Sie den folgenden Befehl aus, um die Komponente aus einer Rezeptdatei zu erstellen. Dieser Befehl erstellt die Komponente und veröffentlicht sie als private AWS IoT Greengrass

Komponente in Ihrem AWS-Konto. Ersetzen Sie *path/to/RecipeFile* durch den Pfad zur Rezeptdatei.

```
aws greengrassv2 create-component-version --inline-recipe fileb://path/to/
recipeFile
```

Kopieren Sie das `arn` aus der Antwort, um im nächsten Schritt den Status der Komponente zu überprüfen.

> **Note**
>
> AWS IoT Greengrass berechnet den Digest jedes Artefakts, wenn Sie die Komponente erstellen. Das bedeutet, dass Sie die Artefaktdateien in Ihrem S3-Bucket nicht ändern können, nachdem Sie eine Komponente erstellt haben. Wenn Sie dies tun, schlagen Bereitstellungen fehl, die diese Komponente enthalten, da der Datei-Digest nicht übereinstimmt. Wenn Sie eine Artefaktdatei ändern, müssen Sie eine neue Version der Komponente erstellen.

4. Jede Komponente im AWS IoT Greengrass Service hat einen Status. Führen Sie den folgenden Befehl aus, um den Status der Komponentenversion zu überprüfen, die Sie in diesem Verfahren veröffentlichen. Ersetzen Sie *com.example. HelloWorld*und *1.0.0* mit der abzufragenden Komponentenversion. Ersetzen Sie das `arn` durch den ARN aus dem vorherigen Schritt.

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

   Der Vorgang gibt eine Antwort zurück, die die Metadaten der Komponente enthält. Die Metadaten enthalten ein `status` Objekt, das den Status der Komponente und etwaige Fehler enthält.

   Wenn der Komponentenstatus lautet`DEPLOYABLE`, können Sie die Komponente auf Geräten bereitstellen. Weitere Informationen finden Sie unter Bereitstellen von AWS IoT Greengrass Komponenten auf Geräten.

Interagieren mit -AWSServices

Greengrass-Core-Geräte verwenden X.509-Zertifikate, um AWS IoT Core mithilfe von TLS-Protokollen für die gegenseitige Authentifizierung eine Verbindung zu herzustellen. Mit diesen Zertifikaten können Geräte AWS IoT ohne AWS Anmeldeinformationen mit interagieren, die in der Regel eine Zugriffsschlüssel-ID und einen geheimen Zugriffsschlüssel umfassen. Andere -AWSServices benötigen AWS Anmeldeinformationen anstelle von X.509-Zertifikaten, um API-Operationen an Service-Endpunkten aufzurufen. AWS IoT Core verfügt über einen Anbieter von Anmeldeinformationen, der es Geräten ermöglicht, ihr X.509-Zertifikat zur Authentifizierung von AWS Anforderungen zu verwenden. Der AWS IoT Anmeldeinformationsanbieter authentifiziert Geräte mit einem X.509-Zertifikat und gibt AWS Anmeldeinformationen in Form eines temporären Sicherheitstokens mit eingeschränkten Berechtigungen aus. Geräte können dieses Token verwenden, um jede AWS Anforderung zu signieren und zu authentifizieren. Dadurch entfällt die Notwendigkeit, AWS Anmeldeinformationen auf Greengrass-Core-Geräten zu speichern. Weitere Informationen finden Sie unter Autorisieren von direkten Aufrufen an -AWSServices im AWS IoT Core -Entwicklerhandbuch.

Um Anmeldeinformationen von abzurufenAWS IoT, verwenden Greengrass-Core-Geräte einen -AWS IoTRollenalias, der auf eine IAM-Rolle verweist. Diese IAM-Rolle wird als Token-Exchange-Rolle bezeichnet. Sie erstellen den Rollenalias und die Token-Exchange-Rolle, wenn Sie die AWS IoT Greengrass Core-Software installieren. Um den Rollenalias anzugeben, den ein Core-Gerät verwendet, konfigurieren Sie den `iotRoleAlias` Parameter von Grüngraskern.

Der AWS IoT Anmeldeinformationsanbieter übernimmt die Token-Exchange-Rolle in Ihrem Namen, um -AWSAnmeldeinformationen für -Core-Geräte bereitzustellen. Sie können dieser Rolle entsprechende IAM-Richtlinien anfügen, um Ihren -Core-Geräten den Zugriff auf Ihre -AWSRessourcen zu ermöglichen, z. B. Komponentenartefakte in S3-Buckets. Weitere Informationen zum Konfigurieren der Token-Exchange-Rolle finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

Greengrass-Core-Geräte speichern AWS Anmeldeinformationen im Speicher, und die Anmeldeinformationen laufen standardmäßig nach einer Stunde ab. Wenn die AWS IoT Greengrass Core-Software neu gestartet wird, muss sie die Anmeldeinformationen erneut abrufen. Sie können die -UpdateRoleAliasOperation verwenden, um die Gültigkeitsdauer der Anmeldeinformationen zu konfigurieren.

AWS IoT Greengrass bietet eine öffentliche Komponente, die Token-Exchange-Servicekomponente, die Sie als Abhängigkeit in Ihrer benutzerdefinierten Komponente definieren können, um mit

-AWSServices zu interagieren. Der Token-Exchange-Service stellt Ihrer Komponente eine Umgebungsvariable zur Verfügung, `AWS_CONTAINER_CREDENTIALS_FULL_URI`, die den URI für einen lokalen Server definiert, der AWS Anmeldeinformationen bereitstellt. Wenn Sie einen AWS SDK-Client erstellen, sucht der Client nach dieser Umgebungsvariablen und stellt eine Verbindung zum lokalen Server her, um AWS Anmeldeinformationen abzurufen, und verwendet sie zum Signieren von API-Anforderungen. Auf diese Weise können Sie AWS -SDKs und andere Tools verwenden, um -AWSServices in Ihren Komponenten aufzurufen. Weitere Informationen finden Sie unter Token-Exchange-Service.

> **Important**
>
> Unterstützung für den Erhalt von AWS Anmeldeinformationen auf diese Weise wurde den AWS -SDKs am 13. Juli 2016 hinzugefügt. Ihre Komponente muss eine AWS SDK-Version verwenden, die an oder nach diesem Datum erstellt wurde. Weitere Informationen finden Sie unter Verwenden eines unterstützten AWS SDK im Amazon Elastic Container Service-Entwicklerhandbuch.

Um AWS Anmeldeinformationen in Ihrer benutzerdefinierten Komponente zu erhalten, definieren Sie `aws.greengrass.TokenExchangeService` als Abhängigkeit im Komponentenrezept. Das folgende Beispielrezept definiert eine Komponente, die boto3 installiert und ein Python-Skript ausführt, das AWS Anmeldeinformationen aus dem Token-Exchange-Service verwendet, um Amazon S3-Buckets aufzulisten.

> **Note**
>
> Um diese Beispielkomponente auszuführen, muss Ihr Gerät über die `-s3:ListAllMyBuckets`Berechtigung verfügen. Weitere Informationen finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.ListS3Buckets",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that uses the token exchange service to list
 S3 buckets.",
```

```
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.TokenExchangeService": {
      "VersionRequirement": "^2.0.0",
      "DependencyType": "HARD"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "pip3 install --user boto3",
        "run": "python3 -u {artifacts:path}/list_s3_buckets.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "pip3 install --user boto3",
        "run": "py -3 -u {artifacts:path}/list_s3_buckets.py"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.ListS3Buckets
ComponentVersion: '1.0.0'
ComponentDescription: A component that uses the token exchange service to list S3
 buckets.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.TokenExchangeService:
    VersionRequirement: '^2.0.0'
    DependencyType: HARD
Manifests:
```

```
  - Platform:
      os: linux
    Lifecycle:
      install:
        pip3 install --user boto3
      run: |-
        python3 -u {artifacts:path}/list_s3_buckets.py
  - Platform:
      os: windows
    Lifecycle:
      install:
        pip3 install --user boto3
      run: |-
        py -3 -u {artifacts:path}/list_s3_buckets.py
```

Diese Beispielkomponente führt das folgende Python-Skript aus, `list_s3_buckets.py` das Amazon S3-Buckets auflistet.

```
import boto3
import os

try:
    print("Creating boto3 S3 client...")
    s3 = boto3.client('s3')
    print("Successfully created boto3 S3 client")
except Exception as e:
    print("Failed to create boto3 s3 client. Error: " + str(e))
    exit(1)

try:
    print("Listing S3 buckets...")
    response = s3.list_buckets()
    for bucket in response['Buckets']:
        print(f'\t{bucket["Name"]}')
    print("Successfully listed S3 buckets")
except Exception as e:
    print("Failed to list S3 buckets. Error: " + str(e))
    exit(1)
```

Führen Sie einen Docker-Container aus

Sie können AWS IoT Greengrass Komponenten so konfigurieren, dass sie einen Docker-Container aus Images ausführen, die an den folgenden Speicherorten gespeichert sind:

- Öffentliche und private Image-Repositorys in Amazon Elastic Container Registry (Amazon ECR)
- Öffentliches Docker Hub-Repository
- Vertrauenswürdiges öffentliches Docker-Register
- S3-Bucket

Fügen Sie in Ihrer benutzerdefinierten Komponente den Docker-Image-URI als Artefakt ein, um das Image abzurufen und auf dem Kerngerät auszuführen. Für Amazon ECR- und Docker Hub-Images können Sie die Docker Application Manager-Komponente verwenden, um die Images herunterzuladen und Anmeldeinformationen für private Amazon ECR-Repositorys zu verwalten.

Themen

Voraussetzungen

Um einen Docker-Container in einer Komponente auszuführen, benötigen Sie Folgendes:

- Ein Greengrass-Core-Gerät. Falls Sie noch keines haben, beachten Sie die Informationen unter Tutorial: Erste Schritte mit AWS IoT Greengrass V2.
- Docker Engine 1.9.1 oder höher ist auf dem Greengrass-Core-Gerät installiert. Version 20.10 ist die neueste Version, für die verifiziert wurde, dass sie mit der Core-Software funktioniert. AWS IoT Greengrass Sie müssen Docker direkt auf dem Kerngerät installieren, bevor Sie Komponenten bereitstellen, auf denen Docker-Container ausgeführt werden.

> **Tip**
>
> Sie können das Kerngerät auch so konfigurieren, dass die Docker Engine bei der Installation der Komponente installiert wird. Das folgende Installationsskript installiert beispielsweise die Docker Engine, bevor das Docker-Image geladen wird. Dieses Installationsskript funktioniert auf Debian-basierten Linux-Distributionen wie Ubuntu. Wenn Sie die Komponente für die Installation von Docker Engine mit diesem Befehl konfigurieren, müssen Sie möglicherweise `true` im Lifecycle-Skript auf einstellen, `RequiresPrivilege` um die Installation und die Befehle auszuführen. `docker` Weitere Informationen finden Sie unter AWS IoT Greengrass Referenz zum Komponenten-Rezept.
>
> ```
> apt-get install docker-ce docker-ce-cli containerd.io && docker load -i
>  {artifacts:path}/hello-world.tar
> ```

- Der Systembenutzer, der eine Docker-Container-Komponente ausführt, muss über Root- oder Administratorrechte verfügen, oder Sie müssen Docker so konfigurieren, dass es als Benutzer ohne Root- oder Administratorrechte ausgeführt wird.
  - Auf Linux-Geräten können Sie der Gruppe einen Benutzer hinzufügen, um Befehle ohne Befehle `docker` aufzurufen. `docker` `sudo`
  - Auf Windows-Geräten können Sie der `docker-users` Gruppe einen Benutzer hinzufügen, um `docker` Befehle ohne Administratorrechte aufzurufen.

  Linux or Unix

  Führen Sie den folgenden Befehl aus`ggc_user`, um der `docker` Gruppe einen Nicht-Root-Benutzer, den Sie zum Ausführen von Docker-Container-Komponenten verwenden, hinzuzufügen.

  ```
  sudo usermod -aG docker ggc_user
  ```

  Weitere Informationen finden Sie unter Docker als Nicht-Root-Benutzer verwalten.

  Windows Command Prompt (CMD)

  Um der `docker-users` Gruppe den Benutzer`ggc_user`, den Sie zum Ausführen von Docker-Container-Komponenten verwenden, hinzuzufügen, führen Sie den folgenden Befehl als Administrator aus.

```
net localgroup docker-users ggc_user /add
```

Windows PowerShell

Um der `docker-users` Gruppe den Benutzer`ggc_user`, den Sie zum Ausführen von Docker-Container-Komponenten verwenden, hinzuzufügen, führen Sie den folgenden Befehl als Administrator aus.

```
Add-LocalGroupMember -Group docker-users -Member ggc_user
```

- Dateien, auf die die Docker-Container-Komponente zugreift, die als Volume im Docker-Container gemountet ist.
- Wenn Sie die AWS IoT Greengrass Core-Software für die Verwendung eines Netzwerk-Proxys konfigurieren, müssen Sie Docker für die Verwendung desselben Proxyservers konfigurieren.

Zusätzlich zu diesen Anforderungen müssen Sie auch die folgenden Anforderungen erfüllen, wenn sie für Ihre Umgebung gelten:

- Um Docker Compose zum Erstellen und Starten Ihrer Docker-Container zu verwenden, installieren Sie Docker Compose auf Ihrem Greengrass-Core-Gerät und laden Sie Ihre Docker Compose-Datei in einen S3-Bucket hoch. Sie müssen Ihre Compose-Datei in einem S3-Bucket im selben und wie AWS-Konto die Komponente speichern. AWS-Region Ein Beispiel, das den `docker-compose up` Befehl in einer benutzerdefinierten Komponente verwendet, finden Sie unterFühren Sie einen Docker-Container von einem öffentlichen Image in Amazon ECR oder Docker Hub aus.
- Wenn Sie AWS IoT Greengrass hinter einem Netzwerk-Proxy laufen, konfigurieren Sie den Docker-Daemon für die Verwendung eines Proxyservers.
- Wenn Ihre Docker-Images in Amazon ECR oder Docker Hub gespeichert sind, fügen Sie die Komponente Docker Component Manager als Abhängigkeit in Ihre Docker-Container-Komponente ein. Sie müssen den Docker-Daemon auf dem Kerngerät starten, bevor Sie Ihre Komponente bereitstellen.

  Fügen Sie außerdem die Image-URIs als Komponentenartefakte hinzu. Bild-URIs müssen das Format `docker:`*`registry`*`/`*`image`*`[`*`:tag|@digest`*`]` haben, das in den folgenden Beispielen gezeigt wird:

  - Privates Amazon ECR-Bild: `docker:`*`account-id`*`.dkr.ecr.`*`region`*`.amazonaws.com/`*`repository`*`/`*`image`*`[`*`:tag|@digest`*`]`

- Öffentliches Amazon ECR-Bild: `docker:public.ecr.aws/`*`repository/image`*`[`*`:tag|@digest`*`]`
- Öffentliches Docker Hub-Bild: `docker:`*`name`*`[`*`:tag|@digest`*`]`

Weitere Informationen zum Ausführen von Docker-Containern aus Images, die in öffentlichen Repositorys gespeichert sind, finden Sie unter. Führen Sie einen Docker-Container von einem öffentlichen Image in Amazon ECR oder Docker Hub aus

- Wenn Ihre Docker-Images in einem privaten Amazon ECR-Repository gespeichert sind, müssen Sie die Token-Exchange-Servicekomponente als Abhängigkeit in die Docker-Container-Komponente aufnehmen. Außerdem muss die Greengrass-Geräterolle die `ecr:GetDownloadUrlForLayer` Aktionen `ecr:GetAuthorizationTokenecr:BatchGetImage`, und zulassen, wie in der folgenden IAM-Beispielrichtlinie gezeigt.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

Informationen zum Ausführen von Docker-Containern aus Images, die in einem privaten Amazon ECR-Repository gespeichert sind, finden Sie unter. Führen Sie einen Docker-Container von einem privaten Image in Amazon ECR aus

- Um Docker-Images zu verwenden, die in einem privaten Amazon ECR-Repository gespeichert sind, muss sich das private Repository auf demselben Gerät befinden AWS-Region wie das Kerngerät.

- Wenn Ihre Docker-Images oder Compose-Dateien in einem S3-Bucket gespeichert sind, muss die Greengrass-Geräterolle die `s3:GetObject` Berechtigung gewähren, Kerngeräten das Herunterladen der Images als Komponentenartefakte zu gestatten, wie in der folgenden IAM-Beispielrichtlinie gezeigt.

  ```
  {
    "Version": "2012-10-17",
    "Statement": [
      {
        "Action": [
          "s3:GetObject"
        ],
        "Resource": [
          "*"
        ],
        "Effect": "Allow"
      }
    ]
  }
  ```

  Informationen zum Ausführen von Docker-Containern aus in Amazon S3 gespeicherten Bildern finden Sie unterFühren Sie einen Docker-Container von einem Image in Amazon S3 aus.

- Um Interprozesskommunikation (IPC), AWS Anmeldeinformationen oder Stream Manager in Ihrer Docker-Container-Komponente zu verwenden, müssen Sie bei der Ausführung des Docker-Containers zusätzliche Optionen angeben. Weitere Informationen finden Sie hier:
  - Verwenden Sie die Interprozesskommunikation in Docker-Container-Komponenten
  - Verwenden Sie AWS Anmeldeinformationen in Docker-Container-Komponenten (Linux)
  - Verwenden Sie den Stream-Manager in Docker-Container-Komponenten (Linux)

Führen Sie einen Docker-Container von einem öffentlichen Image in Amazon ECR oder Docker Hub aus

In diesem Abschnitt wird beschrieben, wie Sie eine benutzerdefinierte Komponente erstellen können, die Docker Compose verwendet, um einen Docker-Container aus Docker-Images auszuführen, die in Amazon ECR und Docker Hub gespeichert sind.

Um einen Docker-Container mit Docker Compose auszuführen

1. Erstellen Sie eine Docker Compose-Datei und laden Sie sie in einen Amazon S3 S3-Bucket hoch. Stellen Sie sicher, dass die Greengrass-Geräterolle die `s3:GetObject` Erlaubnis erteilt, dem Gerät den Zugriff auf die Compose-Datei zu ermöglichen. Die im folgenden Beispiel gezeigte Compose-Beispieldatei enthält das Amazon CloudWatch Agent-Image von Amazon ECR und das MySQL-Image von Docker Hub.

   ```
   version: "3"
   services:
     cloudwatchagent:
       image: "public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest"
     mysql:
       image: "mysql:8.0"
   ```

2. Erstellen Sie eine benutzerdefinierte Komponente auf Ihrem AWS IoT Greengrass Kerngerät. Das im folgenden Beispiel gezeigte Beispielrezept hat die folgenden Eigenschaften:

   - Die Docker-Anwendungsmanager-Komponente als Abhängigkeit. Diese Komponente ermöglicht AWS IoT Greengrass das Herunterladen von Bildern aus öffentlichen Amazon ECR- und Docker Hub-Repositorys.
   - Ein Komponentenartefakt, das ein Docker-Image in einem öffentlichen Amazon ECR-Repository spezifiziert.
   - Ein Komponentenartefakt, das ein Docker-Image in einem öffentlichen Docker Hub-Repository spezifiziert.
   - Ein Komponentenartefakt, das die Docker Compose-Datei spezifiziert, die Container für die Docker-Images enthält, die Sie ausführen möchten.
   - Ein Lifecycle-Run-Skript, das Docker-Compose verwendet, um einen Container aus den angegebenen Images zu erstellen und zu starten.

   JSON

   ```
   {
     "RecipeFormatVersion": "2020-01-25",
     "ComponentName": "com.example.MyDockerComposeComponent",
     "ComponentVersion": "1.0.0",
     "ComponentDescription": "A component that uses Docker Compose to run images
    from public Amazon ECR and Docker Hub.",
   ```

```
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.DockerApplicationManager": {
      "VersionRequirement": "~2.0.0"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Lifecycle": {
        "run": "docker-compose -f {artifacts:path}/docker-compose.yaml up"
      },
      "Artifacts": [
        {
          "URI": "docker:public.ecr.aws/cloudwatch-agent/cloudwatch-
agent:latest"
        },
        {
          "URI": "docker:mysql:8.0"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/folder/docker-compose.yaml"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.MyDockerComposeComponent
ComponentVersion: '1.0.0'
ComponentDescription: 'A component that uses Docker Compose to run images from
 public Amazon ECR and Docker Hub.'
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.DockerApplicationManager:
    VersionRequirement: ~2.0.0
Manifests:
```

```
  - Platform:
      os: all
    Lifecycle:
        run: docker-compose -f {artifacts:path}/docker-compose.yaml up
    Artifacts:
      - URI: "docker:public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest"
      - URI: "docker:mysql:8.0"
      - URI: "s3://DOC-EXAMPLE-BUCKET/folder/docker-compose.yaml"
```

> **Note**
>
> Um Interprozesskommunikation (IPC), AWS Anmeldeinformationen oder Stream Manager in Ihrer Docker-Container-Komponente zu verwenden, müssen Sie bei der Ausführung des Docker-Containers zusätzliche Optionen angeben. Weitere Informationen finden Sie hier:
>
> - Verwenden Sie die Interprozesskommunikation in Docker-Container-Komponenten
> - Verwenden Sie AWS Anmeldeinformationen in Docker-Container-Komponenten (Linux)
> - Verwenden Sie den Stream-Manager in Docker-Container-Komponenten (Linux)

3. Testen Sie die Komponente, um sicherzustellen, dass sie wie erwartet funktioniert.

   > **Important**
   >
   > Sie müssen den Docker-Daemon installieren und starten, bevor Sie die Komponente bereitstellen.

   Nachdem Sie die Komponente lokal bereitgestellt haben, können Sie den Befehl docker container ls ausführen, um zu überprüfen, ob Ihr Container ausgeführt wird.

   ```
   docker container ls
   ```

4. Wenn die Komponente bereit ist, laden Sie die Komponente hoch, AWS IoT Greengrass um sie auf anderen Kerngeräten bereitzustellen. Weitere Informationen finden Sie unter Veröffentlichen Sie Komponenten zur Bereitstellung auf Ihren Kerngeräten.

Führen Sie einen Docker-Container von einem privaten Image in Amazon ECR aus

In diesem Abschnitt wird beschrieben, wie Sie eine benutzerdefinierte Komponente erstellen können, die einen Docker-Container aus einem Docker-Image ausführt, das in einem privaten Repository in Amazon ECR gespeichert ist.

Um einen Docker-Container auszuführen

1. Erstellen Sie eine benutzerdefinierte Komponente auf Ihrem AWS IoT Greengrass Kerngerät. Verwenden Sie das folgende Beispielrezept mit den folgenden Eigenschaften:

   - Die Docker-Anwendungsmanager-Komponente als Abhängigkeit. Diese Komponente ermöglicht die Verwaltung von Anmeldeinformationen AWS IoT Greengrass zum Herunterladen von Bildern aus privaten Repositorys.
   - Die Token-Exchange-Dienstkomponente als Abhängigkeit. Diese Komponente ermöglicht AWS IoT Greengrass das Abrufen von AWS Anmeldeinformationen für die Interaktion mit Amazon ECR.
   - Ein Komponentenartefakt, das ein Docker-Image in einem privaten Amazon ECR-Repository spezifiziert.
   - Ein Lifecycle-Run-Skript, das Docker Run verwendet, um einen Container aus dem Image zu erstellen und zu starten.

   JSON

   ```
   {
     "RecipeFormatVersion": "2020-01-25",
     "ComponentName": "com.example.MyPrivateDockerComponent",
     "ComponentVersion": "1.0.0",
     "ComponentDescription": "A component that runs a Docker container from a
    private Amazon ECR image.",
     "ComponentPublisher": "Amazon",
     "ComponentDependencies": {
       "aws.greengrass.DockerApplicationManager": {
         "VersionRequirement": "~2.0.0"
       },
       "aws.greengrass.TokenExchangeService": {
         "VersionRequirement": "~2.0.0"
       }
     },
   ```

```
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Lifecycle": {
        "run": "docker run account-
id.dkr.ecr.region.amazonaws.com/repository[:tag|@digest]"
      },
      "Artifacts": [
        {
          "URI": "docker:account-
id.dkr.ecr.region.amazonaws.com/repository[:tag|@digest]"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.MyPrivateDockerComponent
ComponentVersion: '1.0.0'
ComponentDescription: 'A component that runs a Docker container from a private
 Amazon ECR image.'
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.DockerApplicationManager:
    VersionRequirement: ~2.0.0
  aws.greengrass.TokenExchangeService:
    VersionRequirement: ~2.0.0
Manifests:
  - Platform:
      os: all
    Lifecycle:
        run: docker run account-id.dkr.ecr.region.amazonaws.com/repository[:tag|
@digest]
    Artifacts:
      - URI: "docker:account-id.dkr.ecr.region.amazonaws.com/repository[:tag|
@digest]"
```

> **Note**
>
> Um Interprozesskommunikation (IPC), AWS Anmeldeinformationen oder Stream Manager in Ihrer Docker-Container-Komponente zu verwenden, müssen Sie bei der Ausführung des Docker-Containers zusätzliche Optionen angeben. Weitere Informationen finden Sie hier:
>
> - Verwenden Sie die Interprozesskommunikation in Docker-Container-Komponenten
> - Verwenden Sie AWS Anmeldeinformationen in Docker-Container-Komponenten (Linux)
> - Verwenden Sie den Stream-Manager in Docker-Container-Komponenten (Linux)

2. Testen Sie die Komponente, um sicherzustellen, dass sie wie erwartet funktioniert.

   > **Important**
   >
   > Sie müssen den Docker-Daemon installieren und starten, bevor Sie die Komponente bereitstellen.

   Nachdem Sie die Komponente lokal bereitgestellt haben, können Sie den Befehl docker container ls ausführen, um zu überprüfen, ob Ihr Container ausgeführt wird.

   ```
   docker container ls
   ```

3. Laden Sie die Komponente hoch, AWS IoT Greengrass um sie auf anderen Kerngeräten bereitzustellen. Weitere Informationen finden Sie unter Veröffentlichen Sie Komponenten zur Bereitstellung auf Ihren Kerngeräten.

Führen Sie einen Docker-Container von einem Image in Amazon S3 aus

In diesem Abschnitt wird beschrieben, wie Sie einen Docker-Container in einer Komponente von einem Docker-Image aus ausführen können, das in Amazon S3 gespeichert ist.

So führen Sie einen Docker-Container in einer Komponente aus einem Image in Amazon S3 aus

1. Führen Sie den Befehl docker save aus, um ein Backup eines Docker-Containers zu erstellen. Sie stellen dieses Backup als Komponentenartefakt bereit, auf dem der Container ausgeführt werden kann. AWS IoT Greengrass Ersetzen Sie *hello-world* durch den Namen des Images und *hello-world.tar* durch den Namen der zu erstellenden Archivdatei.

   ```
   docker save hello-world > artifacts/com.example.MyDockerComponent/1.0.0/hello-world.tar
   ```

2. Erstellen Sie eine benutzerdefinierte Komponente auf Ihrem AWS IoT Greengrass Core-Gerät. Verwenden Sie das folgende Beispielrezept mit den folgenden Eigenschaften:

   - Ein Lifecycle-Installationsskript, das Docker Load verwendet, um ein Docker-Image aus einem Archiv zu laden.
   - Ein Lifecycle-Run-Skript, das Docker Run verwendet, um einen Container aus dem Image zu erstellen und zu starten. Die `--rm` Option bereinigt den Container, wenn er beendet wird.

   JSON

   ```
   {
     "RecipeFormatVersion": "2020-01-25",
     "ComponentName": "com.example.MyS3DockerComponent",
     "ComponentVersion": "1.0.0",
     "ComponentDescription": "A component that runs a Docker container from an
    image in an S3 bucket.",
     "ComponentPublisher": "Amazon",
     "Manifests": [
       {
         "Platform": {
           "os": "linux"
         },
         "Lifecycle": {
           "install": {
             "Script": "docker load -i {artifacts:path}/hello-world.tar"
           },
           "run": {
             "Script": "docker run --rm hello-world"
           }
         }
       }
   ```

```
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.MyS3DockerComponent
ComponentVersion: '1.0.0'
ComponentDescription: 'A component that runs a Docker container from an image in
 an S3 bucket.'
ComponentPublisher: Amazon
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install:
        Script: docker load -i {artifacts:path}/hello-world.tar
      run:
        Script: docker run --rm hello-world
```

> **Note**
>
> Um Interprozesskommunikation (IPC), AWS Anmeldeinformationen oder Stream Manager in Ihrer Docker-Container-Komponente zu verwenden, müssen Sie bei der Ausführung des Docker-Containers zusätzliche Optionen angeben. Weitere Informationen finden Sie hier:
>
> - Verwenden Sie die Interprozesskommunikation in Docker-Container-Komponenten
> - Verwenden Sie AWS Anmeldeinformationen in Docker-Container-Komponenten (Linux)
> - Verwenden Sie den Stream-Manager in Docker-Container-Komponenten (Linux)

3. Testen Sie die Komponente, um sicherzustellen, dass sie wie erwartet funktioniert.

   Nachdem Sie die Komponente lokal bereitgestellt haben, können Sie den Befehl docker container ls ausführen, um zu überprüfen, ob Ihr Container ausgeführt wird.

   ```
   docker container ls
   ```

4. Wenn die Komponente bereit ist, laden Sie das Docker-Image-Archiv in einen S3-Bucket hoch und fügen Sie dessen URI dem Komponentenrezept hinzu. Anschließend können Sie die Komponente hochladen, AWS IoT Greengrass um sie auf anderen Kerngeräten bereitzustellen. Weitere Informationen finden Sie unter Veröffentlichen Sie Komponenten zur Bereitstellung auf Ihren Kerngeräten.

   Wenn Sie fertig sind, sollte das Komponentenrezept wie im folgenden Beispiel aussehen.

   JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.MyS3DockerComponent",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that runs a Docker container from an
 image in an S3 bucket.",
  "ComponentPublisher": "Amazon",
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": {
          "Script": "docker load -i {artifacts:path}/hello-world.tar"
        },
        "run": {
          "Script": "docker run --rm hello-world"
        }
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.MyDockerComponent/1.0.0/hello-world.tar"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.MyS3DockerComponent
ComponentVersion: '1.0.0'
ComponentDescription: 'A component that runs a Docker container from an image in
 an S3 bucket.'
ComponentPublisher: Amazon
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install:
        Script: docker load -i {artifacts:path}/hello-world.tar
      run:
        Script: docker run --rm hello-world
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.MyDockerComponent/1.0.0/hello-world.tar
```

Verwenden Sie die Interprozesskommunikation in Docker-Container-Komponenten

Sie können die Greengrass Interprocess Communication (IPC) -Bibliothek in der verwenden, AWS IoT Device SDK um mit dem Greengrass-Kern, anderen Greengrass-Komponenten und zu kommunizieren. AWS IoT Core Weitere Informationen finden Sie unter Verwenden Sie den AWS IoT Device SDK , um mit dem Greengrass-Kern und anderen Komponenten zu kommunizieren und AWS IoT Core.

Um IPC in einer Docker-Container-Komponente zu verwenden, müssen Sie den Docker-Container mit den folgenden Parametern ausführen:

- Mounten Sie den IPC-Socket in den Container. Der Greengrass-Kern stellt den IPC-Socket-Dateipfad in der `AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT` Umgebungsvariablen bereit.
- Stellen Sie die `AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT` Umgebungsvariablen `SVCUID` und auf die Werte ein, die der Greengrass-Kern für Komponenten bereitstellt. Ihre Komponente verwendet diese Umgebungsvariablen, um Verbindungen zum Greengrass-Nucleus zu authentifizieren.

Example Beispielrezept: Eine MQTT-Nachricht veröffentlichen in AWS IoT Core (Python)

Das folgende Rezept definiert ein Beispiel für eine Docker-Container-Komponente, die eine MQTT-Nachricht veröffentlicht. AWS IoT Core Das Rezept hat die folgenden Eigenschaften:

- Eine Autorisierungsrichtlinie (`accessControl`), die es der Komponente ermöglicht, MQTT-Nachrichten zu allen AWS IoT Core Themen zu veröffentlichen. Weitere Informationen finden Sie unter Autorisieren Sie Komponenten zur Ausführung von IPC-Vorgängen und AWS IoT Core MQTT IPC-Autorisierung.
- Ein Komponentenartefakt, das ein Docker-Image als TAR-Archiv in Amazon S3 spezifiziert.
- Ein Lifecycle-Installationsskript, das das Docker-Image aus dem TAR-Archiv lädt.
- Ein Lifecycle-Run-Skript, das einen Docker-Container vom Image aus ausführt. Der Befehl Docker run hat die folgenden Argumente:
  - Das `-v` Argument mountet den Greengrass-IPC-Socket in den Container.
  - Die ersten beiden `-e` Argumente legen die erforderlichen Umgebungsvariablen im Docker-Container fest.
  - Die zusätzlichen `-e` Argumente legen die in diesem Beispiel verwendeten Umgebungsvariablen fest.
  - Das `--rm` Argument bereinigt den Container, wenn er beendet wird.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.python.docker.PublishToIoTCore",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses interprocess communication to publish an MQTT
 message to IoT Core.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "topic": "test/topic/java",
      "message": "Hello, World!",
      "qos": "1",
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.python.docker.PublishToIoTCore:pubsub:1": {
```

```
            "policyDescription": "Allows access to publish to IoT Core on all
 topics.",
            "operations": [
              "aws.greengrass#PublishToIoTCore"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Lifecycle": {
        "install": "docker load -i {artifacts:path}/publish-to-iot-core.tar",
        "run": "docker run -v $AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT:
$AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT -e SVCUID -e
 AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT -e MQTT_TOPIC=
\"{configuration:/topic}\" -e MQTT_MESSAGE=\"{configuration:/message}\" -e MQTT_QOS=
\"{configuration:/qos}\" --rm publish-to-iot-core"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.python.docker.PublishToIoTCore/1.0.0/publish-to-iot-core.tar"
        }
      ]
    }
  ]
}
```

YAML

```
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.python.docker.PublishToIoTCore
ComponentVersion: 1.0.0
ComponentDescription: Uses interprocess communication to publish an MQTT message to
 IoT Core.
```

```
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    topic: 'test/topic/java'
    message: 'Hello, World!'
    qos: '1'
    accessControl:
      aws.greengrass.ipc.mqttproxy:
        'com.example.python.docker.PublishToIoTCore:pubsub:1':
          policyDescription: Allows access to publish to IoT Core on all topics.
          operations:
            - 'aws.greengrass#PublishToIoTCore'
          resources:
            - '*'
Manifests:
  - Platform:
      os: all
    Lifecycle:
      install: 'docker load -i {artifacts:path}/publish-to-iot-core.tar'
      run: |
        docker run \
          -v $AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT:
$AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT \
          -e SVCUID \
          -e AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT \
          -e MQTT_TOPIC="{configuration:/topic}" \
          -e MQTT_MESSAGE="{configuration:/message}" \
          -e MQTT_QOS="{configuration:/qos}" \
          --rm publish-to-iot-core
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.python.docker.PublishToIoTCore/1.0.0/publish-to-iot-core.tar
```

Verwenden Sie AWS Anmeldeinformationen in Docker-Container-Komponenten (Linux)

Sie können die Token-Exchange-Dienstkomponente verwenden, um mit AWS Diensten in Greengrass-Komponenten zu interagieren. Diese Komponente stellt AWS Anmeldeinformationen aus der Token-Austauschrolle des Kerngeräts mithilfe eines lokalen Containerservers bereit. Weitere Informationen finden Sie unter Interagieren mit -AWSServices.

> **Note**
>
> Das Beispiel in diesem Abschnitt funktioniert nur auf Linux-Core-Geräten.

Um AWS Anmeldeinformationen aus dem Token-Exchange-Dienst in einer Docker-Container-Komponente zu verwenden, müssen Sie den Docker-Container mit den folgenden Parametern ausführen:

- Stellen Sie mithilfe des Arguments Zugriff auf das Host-Netzwerk bereit. `--network=host` Diese Option ermöglicht es dem Docker-Container, eine Verbindung zum lokalen Token-Austauschdienst herzustellen, um AWS Anmeldeinformationen abzurufen. Dieses Argument funktioniert nur auf Docker für Linux.

  > **Warning**
  >
  > Diese Option gewährt dem Container Zugriff auf alle lokalen Netzwerkschnittstellen auf dem Host, sodass diese Option weniger sicher ist, als wenn Sie Docker-Container ohne diesen Zugriff auf das Host-Netzwerk ausführen. Berücksichtigen Sie dies, wenn Sie Docker-Container-Komponenten entwickeln und ausführen, die diese Option verwenden. Weitere Informationen finden Sie unter Network: host in der Docker-Dokumentation.

- Stellen Sie die `AWS_CONTAINER_AUTHORIZATION_TOKEN` Umgebungsvariablen `AWS_CONTAINER_CREDENTIALS_FULL_URI` und auf die Werte ein, die der Greengrass-Kern für Komponenten bereitstellt. AWS SDKs verwenden diese Umgebungsvariablen, um Anmeldeinformationen abzurufen AWS .

Example Beispielrezept: S3-Buckets in einer Docker-Container-Komponente auflisten (Python)

Das folgende Rezept definiert ein Beispiel für eine Docker-Container-Komponente, die die S3-Buckets in Ihrem auflistet. AWS-Konto Das Rezept hat die folgenden Eigenschaften:

- Die Token-Exchange-Dienstkomponente als Abhängigkeit. Diese Abhängigkeit ermöglicht es der Komponente, AWS Anmeldeinformationen für die Interaktion mit anderen AWS Diensten abzurufen.
- Ein Komponentenartefakt, das ein Docker-Image als Tar-Archiv in Amazon S3 spezifiziert.
- Ein Lifecycle-Installationsskript, das das Docker-Image aus dem TAR-Archiv lädt.

- Ein Lifecycle-Run-Skript, das einen Docker-Container vom Image aus ausführt. Der Befehl Docker run hat die folgenden Argumente:
  - Das `--network=host` Argument gewährt dem Container Zugriff auf das Host-Netzwerk, sodass der Container eine Verbindung zum Token-Austauschdienst herstellen kann.
  - Das `-e` Argument legt die erforderlichen Umgebungsvariablen im Docker-Container fest.
  - Das `--rm` Argument bereinigt den Container, wenn er beendet wird.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.python.docker.ListS3Buckets",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses the token exchange service to lists your S3
 buckets.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.TokenExchangeService": {
      "VersionRequirement": "^2.0.0",
      "DependencyType": "HARD"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "docker load -i {artifacts:path}/list-s3-buckets.tar",
        "run": "docker run --network=host -e AWS_CONTAINER_AUTHORIZATION_TOKEN -e
 AWS_CONTAINER_CREDENTIALS_FULL_URI --rm list-s3-buckets"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.python.docker.ListS3Buckets/1.0.0/list-s3-buckets.tar"
        }
      ]
    }
  ]
```

```
}
```

YAML

```
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.python.docker.ListS3Buckets
ComponentVersion: 1.0.0
ComponentDescription: Uses the token exchange service to lists your S3 buckets.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.TokenExchangeService:
    VersionRequirement: ^2.0.0
    DependencyType: HARD
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: 'docker load -i {artifacts:path}/list-s3-buckets.tar'
      run: |
        docker run \
          --network=host \
          -e AWS_CONTAINER_AUTHORIZATION_TOKEN \
          -e AWS_CONTAINER_CREDENTIALS_FULL_URI \
          --rm list-s3-buckets
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.python.docker.ListS3Buckets/1.0.0/list-s3-buckets.tar
```

Verwenden Sie den Stream-Manager in Docker-Container-Komponenten (Linux)

Sie können die Stream-Manager-Komponente verwenden, um Datenströme in Greengrass-Komponenten zu verwalten. Mit dieser Komponente können Sie Datenströme verarbeiten und umfangreiche IoT-Daten an die AWS Cloudübertragen. AWS IoT Greengrass stellt ein Stream-Manager-SDK bereit, das Sie für die Interaktion mit der Stream Manager-Komponente verwenden. Weitere Informationen finden Sie unter Verwalten von Datenströmen auf Greengrass-Core-Geräten.

> **Note**
>
> Das Beispiel in diesem Abschnitt funktioniert nur auf Linux-Core-Geräten.

Um das Stream Manager SDK in einer Docker-Container-Komponente zu verwenden, müssen Sie den Docker-Container mit den folgenden Parametern ausführen:

- Stellen Sie mithilfe des Arguments Zugriff auf das Host-Netzwerk bereit. `--network=host` Diese Option ermöglicht es dem Docker-Container, über eine lokale TLS-Verbindung mit der Stream Manager-Komponente zu interagieren. Dieses Argument funktioniert nur auf Docker für Linux

  > **Warning**
  >
  > Diese Option gewährt dem Container Zugriff auf alle lokalen Netzwerkschnittstellen auf dem Host, sodass diese Option weniger sicher ist, als wenn Sie Docker-Container ohne diesen Zugriff auf das Host-Netzwerk ausführen. Berücksichtigen Sie dies, wenn Sie Docker-Container-Komponenten entwickeln und ausführen, die diese Option verwenden. Weitere Informationen finden Sie unter Network: host in der Docker-Dokumentation.

- Wenn Sie die Stream-Manager-Komponente so konfigurieren, dass eine Authentifizierung erforderlich ist, was das Standardverhalten ist, setzen Sie die `AWS_CONTAINER_CREDENTIALS_FULL_URI` Umgebungsvariable auf den Wert, den der Greengrass-Nucleus den Komponenten zur Verfügung stellt. Weitere Informationen finden Sie unter Stream Manager-Konfiguration.
- Wenn Sie die Stream Manager-Komponente so konfigurieren, dass sie einen nicht standardmäßigen Port verwendet, verwenden Sie Interprocess Communication (IPC), um den Port aus der Stream Manager-Komponentenkonfiguration abzurufen. Sie müssen den Docker-Container mit zusätzlichen Optionen ausführen, um IPC verwenden zu können. Weitere Informationen finden Sie hier:
  - Stellen Sie im Anwendungscode eine Connect zum Stream Manager her
  - Verwenden Sie die Interprozesskommunikation in Docker-Container-Komponenten

Example Beispielrezept: Streamen Sie eine Datei in einen S3-Bucket in einer Docker-Container-Komponente (Python)

Das folgende Rezept definiert ein Beispiel für eine Docker-Container-Komponente, die eine Datei erstellt und sie in einen S3-Bucket streamt. Das Rezept hat die folgenden Eigenschaften:

- Die Stream-Manager-Komponente als Abhängigkeit. Diese Abhängigkeit ermöglicht es der Komponente, das Stream-Manager-SDK für die Interaktion mit der Stream-Manager-Komponente zu verwenden.

- Ein Komponentenartefakt, das ein Docker-Image als TAR-Archiv in Amazon S3 spezifiziert.
- Ein Lifecycle-Installationsskript, das das Docker-Image aus dem TAR-Archiv lädt.
- Ein Lifecycle-Run-Skript, das einen Docker-Container vom Image aus ausführt. Der Befehl Docker run hat die folgenden Argumente:
  - Das `--network=host` Argument ermöglicht dem Container den Zugriff auf das Host-Netzwerk, sodass der Container eine Verbindung zur Stream Manager-Komponente herstellen kann.
  - Das erste `-e` Argument legt die erforderliche `AWS_CONTAINER_AUTHORIZATION_TOKEN` Umgebungsvariable im Docker-Container fest.
  - Die zusätzlichen `-e` Argumente legen die in diesem Beispiel verwendeten Umgebungsvariablen fest.
  - Das `-v` Argument hängt den Arbeitsordner der Komponente im Container ein. In diesem Beispiel wird eine Datei im Arbeitsordner erstellt, um diese Datei mithilfe des Stream-Managers auf Amazon S3 hochzuladen.
  - Das `--rm` Argument bereinigt den Container, wenn er beendet wird.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.python.docker.StreamFileToS3",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Creates a text file and uses stream manager to stream the
 file to S3.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.StreamManager": {
      "VersionRequirement": "^2.0.0",
      "DependencyType": "HARD"
    }
  },
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "bucketName": ""
    }
  },
  "Manifests": [
    {
      "Platform": {
```

```
        "os": "linux"
      },
      "Lifecycle": {
        "install": "docker load -i {artifacts:path}/stream-file-to-s3.tar",
        "run": "docker run --network=host -e AWS_CONTAINER_AUTHORIZATION_TOKEN
 -e BUCKET_NAME=\"{configuration:/bucketName}\" -e WORK_PATH=\"{work:path}\" -v
 {work:path}:{work:path} --rm stream-file-to-s3"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.python.docker.StreamFileToS3/1.0.0/stream-file-to-s3.tar"
        }
      ]
    }
  ]
}
```

YAML

```
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.python.docker.StreamFileToS3
ComponentVersion: 1.0.0
ComponentDescription: Creates a text file and uses stream manager to stream the file
 to S3.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.StreamManager:
    VersionRequirement: ^2.0.0
    DependencyType: HARD
ComponentConfiguration:
  DefaultConfiguration:
    bucketName: ''
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: 'docker load -i {artifacts:path}/stream-file-to-s3.tar'
      run: |
        docker run \
          --network=host \
          -e AWS_CONTAINER_AUTHORIZATION_TOKEN \
          -e BUCKET_NAME="{configuration:/bucketName}" \
```

```
            -e WORK_PATH="{work:path}" \
            -v {work:path}:{work:path} \
            --rm stream-file-to-s3
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.python.docker.StreamFileToS3/1.0.0/stream-file-to-s3.tar
```

AWS IoT Greengrass Referenz zum Komponenten-Rezept

Das Komponentenrezept ist eine Datei, die die Details, Abhängigkeiten, Artefakte und Lebenszyklen einer Komponente definiert. Der Komponentenlebenszyklus spezifiziert die Befehle, die ausgeführt werden müssen, um beispielsweise die Komponente zu installieren, auszuführen und herunterzufahren. Der AWS IoT Greengrass Kern verwendet die Lebenszyklen, die Sie im Rezept definieren, um Komponenten zu installieren und auszuführen. Der AWS IoT Greengrass Service verwendet das Rezept, um die Abhängigkeiten und Artefakte zu identifizieren, die bei der Bereitstellung der Komponente auf Ihren Kerngeräten bereitgestellt werden sollen.

Im Rezept können Sie eindeutige Abhängigkeiten und Lebenszyklen für jede Plattform definieren, die eine Komponente unterstützt. Sie können diese Funktion verwenden, um eine Komponente auf Geräten mit mehreren Plattformen mit unterschiedlichen Anforderungen bereitzustellen. Sie können damit auch AWS IoT Greengrass verhindern, dass eine Komponente auf Geräten installiert wird, die sie nicht unterstützen.

Jedes Rezept enthält eine Liste von Manifesten. Jedes Manifest spezifiziert eine Reihe von Plattformanforderungen sowie den Lebenszyklus und die Artefakte, die für Kerngeräte verwendet werden sollen, deren Plattform diese Anforderungen erfüllt. Das Kerngerät verwendet das erste Manifest mit den Plattformanforderungen, die das Gerät erfüllt. Geben Sie ein Manifest ohne Plattformanforderungen an, das einem beliebigen Kerngerät entspricht.

Sie können auch einen globalen Lebenszyklus angeben, der nicht in einem Manifest enthalten ist. Im globalen Lebenszyklus können Sie Auswahlschlüssel verwenden, die Unterabschnitte des Lebenszyklus identifizieren. Anschließend können Sie diese Auswahlschlüssel innerhalb eines Manifests angeben, um diese Abschnitte des globalen Lebenszyklus zusätzlich zum Lebenszyklus des Manifests zu verwenden. Das Kerngerät verwendet die Auswahltasten des Manifests nur, wenn das Manifest keinen Lebenszyklus definiert. Sie können die `all` Auswahl in einem Manifest verwenden, um Abschnitte des globalen Lebenszyklus ohne Auswahlschlüssel zuzuordnen.

Nachdem die AWS IoT Greengrass Core-Software ein Manifest ausgewählt hat, das dem Kerngerät entspricht, geht sie wie folgt vor, um die zu verwendenden Lebenszyklusschritte zu identifizieren:

- Wenn das ausgewählte Manifest einen Lebenszyklus definiert, verwendet das Kerngerät diesen Lebenszyklus.
- Wenn das ausgewählte Manifest keinen Lebenszyklus definiert, verwendet das Kerngerät den globalen Lebenszyklus. Das Kerngerät geht wie folgt vor, um zu ermitteln, welche Abschnitte des globalen Lebenszyklus verwendet werden sollen:
  - Wenn das Manifest Auswahlschlüssel definiert, verwendet das Kerngerät die Abschnitte des globalen Lebenszyklus, die die Auswahlschlüssel des Manifests enthalten.
  - Wenn das Manifest keine Auswahltasten definiert, verwendet das Kerngerät die Abschnitte des globalen Lebenszyklus, die keine Auswahltasten haben. Dieses Verhalten entspricht einem Manifest, das die `all` Auswahl definiert.

> **Important**
>
> Ein Kerngerät muss die Plattformanforderungen mindestens eines Manifests erfüllen, um die Komponente installieren zu können. Wenn kein Manifest mit dem Kerngerät übereinstimmt, installiert die AWS IoT Greengrass Core-Software die Komponente nicht und die Bereitstellung schlägt fehl.

Sie können Rezepte im JSON - oder YAML-Format definieren. Der Abschnitt mit den Rezeptbeispielen enthält Rezepte in jedem Format.

Themen

Validierung von Rezepturen

Greengrass validiert ein JSON- oder YAML-Komponentenrezept, wenn eine Komponentenversion erstellt wird. Bei dieser Rezeptvalidierung wird Ihr JSON- oder YAML-Komponentenrezept auf häufig auftretende Fehler überprüft, um mögliche Probleme bei der Bereitstellung zu vermeiden. Bei der Validierung wird das Rezept auf häufige Fehler (z. B. fehlende Kommas, geschweifte Klammern und Felder) überprüft und sichergestellt, dass das Rezept korrekt formuliert ist.

Wenn Sie eine Fehlermeldung bei der Rezeptvalidierung erhalten, überprüfen Sie Ihr Rezept auf fehlende Kommas, Klammern oder Felder. Vergewissern Sie sich anhand des Rezeptformats, dass Ihnen keine Felder fehlen.

Rezeptformat

Wenn Sie ein Rezept für eine Komponente definieren, geben Sie die folgenden Informationen im Rezeptdokument an. Dieselbe Struktur gilt für Rezepte in den Formaten YAML und JSON.

`RecipeFormatVersion`

Die Vorlagenversion für das Rezept. Wählen Sie die folgende Option:

- `2020-01-25`

`ComponentName`

Der Name der Komponente, die dieses Rezept definiert. Der Komponentenname muss für Sie AWS-Konto in jeder Region einzigartig sein.

> **Tipps**
>
> - Verwenden Sie das umgekehrte Domainnamenformat, um Namenskonflikte innerhalb Ihres Unternehmens zu vermeiden. Wenn Ihr Unternehmen beispielsweise Eigentümer eines Solarenergieprojekts ist `example.com` und Sie daran arbeiten, können Sie Ihrer Komponente `com.example.solar.HelloWorld` den Namen Hello World geben. Auf diese Weise können Kollisionen bei Komponentennamen innerhalb Ihres Unternehmens vermieden werden.
> - Vermeiden Sie das `aws.greengrass` Präfix in Ihren Komponentennamen. AWS IoT Greengrass verwendet dieses Präfix für die öffentlichen Komponenten, die es bereitstellt. Wenn Sie denselben Namen wie eine öffentliche Komponente wählen, ersetzt Ihre Komponente diese Komponente. AWS IoT Greengrass Stellt dann Ihre Komponente anstelle der öffentlichen Komponente bereit, wenn Komponenten bereitgestellt werden, die von dieser öffentlichen Komponente abhängig sind. Mit dieser Funktion können Sie das Verhalten öffentlicher Komponenten außer Kraft setzen, aber sie kann auch andere Komponenten beschädigen, wenn Sie nicht beabsichtigen, eine öffentliche Komponente zu überschreiben.

`ComponentVersion`

Die Version der Komponente. Der Höchstwert für die Werte „Major“, „Minor“ und „Patch“ ist 999999.

> **Note**
>
> AWS IoT Greengrass verwendet semantische Versionen für Komponenten. Semantische Versionen folgen einem Hauptteil. unbedeutend. Patch-Nummernsystem. Version `1.0.0` steht beispielsweise für die erste Hauptversion einer Komponente. Weitere Informationen finden Sie in der semantischen Versionsspezifikation.

`ComponentDescription`

(Optional) Die Beschreibung der Komponente.

`ComponentPublisher`

Der Herausgeber oder Autor der Komponente.

`ComponentConfiguration`

(Optional) Ein Objekt, das die Konfiguration oder die Parameter für die Komponente definiert. Sie definieren die Standardkonfiguration, und wenn Sie dann die Komponente bereitstellen, können Sie das Konfigurationsobjekt angeben, das der Komponente zur Verfügung gestellt werden soll. Die Komponentenkonfiguration unterstützt verschachtelte Parameter und Strukturen. Dieses Objekt enthält die folgenden Informationen:

`DefaultConfiguration`

Ein Objekt, das die Standardkonfiguration für die Komponente definiert. Sie definieren die Struktur dieses Objekts.

> **Note**
>
> AWS IoT Greengrass verwendet JSON für Konfigurationswerte. JSON spezifiziert einen Zahlentyp, unterscheidet aber nicht zwischen Ganzzahlen und Gleitkommazahlen. Aus diesem Grund können Konfigurationswerte in Fließkommazahlen umgewandelt werden. AWS IoT Greengrass Um sicherzustellen, dass Ihre Komponente den richtigen Datentyp verwendet, empfehlen wir, numerische

Konfigurationswerte als Zeichenfolgen zu definieren. Lassen Sie Ihre Komponente sie dann als Ganzzahlen oder Gleitkommazahlen analysieren. Dadurch wird sichergestellt, dass Ihre Konfigurationswerte in der Konfiguration und auf Ihrem Kerngerät denselben Typ haben.

`ComponentDependencies`

(Optional) Ein Wörterbuch mit Objekten, die jeweils eine Komponentenabhängigkeit für die Komponente definieren. Der Schlüssel für jedes Objekt identifiziert den Namen der Komponentenabhängigkeit. AWS IoT Greengrass installiert Komponentenabhängigkeiten, wenn die Komponente installiert wird. AWS IoT Greengrass wartet darauf, dass Abhängigkeiten beginnen, bevor die Komponente gestartet wird. Jedes Objekt enthält die folgenden Informationen:

`VersionRequirement`

Die semantische Versionsbeschränkung im NPM-Stil, die die kompatiblen Komponentenversionen für diese Abhängigkeit definiert. Sie können eine Version oder einen Versionsbereich angeben. Weitere Informationen finden Sie im npm Semantic Version Calculator.

`DependencyType`

(Optional) Der Typ dieser Abhängigkeit. Wählen Sie aus den folgenden Optionen aus.

- `SOFT` — Die Komponente startet nicht neu, wenn die Abhängigkeit den Status ändert.
- `HARD` — Die Komponente wird neu gestartet, wenn die Abhängigkeit den Status ändert.

Standardeinstellung: `HARD`.

`ComponentType`

(Optional) Der Typ der Komponente.

> **Note**
>
> Es wird nicht empfohlen, den Komponententyp in einer Rezeptur anzugeben. AWS IoT Greengrass legt den Typ für Sie fest, wenn Sie eine Komponente erstellen.

Bei dem Typ kann es sich um einen der folgenden Typen handeln:

- `aws.greengrass.generic`— Die Komponente führt Befehle aus oder stellt Artefakte bereit.
- `aws.greengrass.lambda`— Die Komponente führt mithilfe der Lambda-Launcher-Komponente eine Lambda-Funktion aus. Der `ComponentSource` Parameter gibt den ARN der Lambda-Funktion an, die diese Komponente ausführt.

  Es wird nicht empfohlen, diese Option zu verwenden, da sie festgelegt wird, AWS IoT Greengrass wenn Sie eine Komponente aus einer Lambda-Funktion erstellen. Weitere Informationen finden Sie unter Ausführen von -AWS LambdaFunktionen.
- `aws.greengrass.plugin`— Die Komponente läuft auf derselben Java Virtual Machine (JVM) wie der Greengrass-Kern. Wenn Sie eine Plugin-Komponente bereitstellen oder neu starten, wird der Greengrass-Nucleus neu gestartet.

  Plugin-Komponenten verwenden dieselbe Protokolldatei wie der Greengrass-Kern. Weitere Informationen finden Sie unter Überwachen von AWS IoT Greengrass Protokollen.

  Wir empfehlen nicht, diese Option in Komponentenrezepten zu verwenden, da sie für in Java geschriebene Komponenten AWS vorgesehen ist, die direkt mit dem Greengrass-Kern verbunden sind. Weitere Hinweise darüber, welche öffentlichen Komponenten Plugins sind, finden Sie unter. AWSVon bereitgestellte Komponenten
- `aws.greengrass.nucleus`— Die Nukleus-Komponente. Weitere Informationen finden Sie unter Grüngraskern.

  Wir empfehlen nicht, diese Option in Rezepturen für Komponenten zu verwenden. Es ist für die Greengrass Nucleus-Komponente vorgesehen, die die Mindestfunktionalität der AWS IoT Greengrass Core-Software bietet.

Standardmäßig, `aws.greengrass.generic` wenn Sie eine Komponente aus einem Rezept erstellen oder `aws.greengrass.lambda` wenn Sie eine Komponente aus einer Lambda-Funktion erstellen.

Weitere Informationen finden Sie unter Komponententypen.

`ComponentSource`

(Optional) Der ARN der Lambda-Funktion, die eine Komponente ausführt.

Es wird nicht empfohlen, die Komponentenquelle in einem Rezept anzugeben. AWS IoT Greengrass legt diesen Parameter für Sie fest, wenn Sie eine Komponente aus einer Lambda-Funktion erstellen. Weitere Informationen finden Sie unter Ausführen von -AWS LambdaFunktionen.

`Manifests`

Eine Liste von Objekten, die jeweils den Lebenszyklus, die Parameter und Anforderungen der Komponente für eine Plattform definieren. Wenn ein Core-Gerät die Plattformanforderungen mehrerer Manifeste erfüllt, AWS IoT Greengrass wird das erste Manifest verwendet, dem das Core-Gerät entspricht. Um sicherzustellen, dass die Kerngeräte das richtige Manifest verwenden, definieren Sie zunächst die Manifeste mit strengeren Plattformanforderungen. Ein Manifest, das für alle Plattformen gilt, muss das letzte Manifest in der Liste sein.

> **Important**
>
> Ein Kerngerät muss die Plattformanforderungen mindestens eines Manifests erfüllen, um die Komponente installieren zu können. Wenn kein Manifest mit dem Kerngerät übereinstimmt, installiert die AWS IoT Greengrass Core-Software die Komponente nicht und die Bereitstellung schlägt fehl.

Jedes Objekt enthält die folgenden Informationen:

`Name`

(Optional) Ein benutzerfreundlicher Name für die Plattform, die dieses Manifest definiert.

Wenn Sie diesen Parameter weglassen, AWS IoT Greengrass wird ein Name aus der Plattform `os` und `architecture` erstellt.

`Platform`

(Optional) Ein Objekt, das die Plattform definiert, für die dieses Manifest gilt. Lassen Sie diesen Parameter weg, um ein Manifest zu definieren, das für alle Plattformen gilt.

Dieses Objekt spezifiziert Schlüssel-Wert-Paare für die Plattform, auf der ein Core-Gerät läuft. Wenn Sie diese Komponente bereitstellen, vergleicht die AWS IoT Greengrass Core-Software diese Schlüssel-Wert-Paare mit den Plattformattributen auf dem Core-Gerät. Die AWS IoT Greengrass Core-Software definiert immer `os` `architecture` und kann zusätzliche Attribute definieren. Sie können benutzerdefinierte Plattformattribute für ein Kerngerät angeben, wenn Sie die Greengrass Nucleus-Komponente bereitstellen. Weitere Informationen finden Sie unter dem Parameter Platform Overrides der Greengrass Nucleus-Komponente.

Für jedes Schlüssel-Wert-Paar können Sie einen der folgenden Werte angeben:

- Ein exakter Wert, z. B. `linux` oder. `windows` Exakte Werte müssen mit einem Buchstaben oder einer Zahl beginnen.
- `*`, was einem beliebigen Wert entspricht. Dies gilt auch, wenn kein Wert vorhanden ist.
- Ein regulärer Ausdruck im Java-Stil, z. B. `/windows|linux/` Der reguläre Ausdruck muss mit einem Schrägstrich () beginnen und enden. / Der reguläre Ausdruck `/.+/` entspricht beispielsweise jedem Wert, der nicht leer ist.

Dieses Objekt enthält die folgenden Informationen:

`os`

(Optional) Der Name des Betriebssystems für die Plattform, die dieses Manifest unterstützt. Zu den gängigen Plattformen gehören die folgenden Werte:

- `linux`
- `windows`
- `darwin` (macOS)

`architecture`

(Optional) Die Prozessorarchitektur für die Plattform, die dieses Manifest unterstützt. Zu den gängigen Architekturen gehören die folgenden Werte:

- `amd64`
- `arm`
- `aarch64`
- `x86`

`architecture.detail`

(Optional) Die Details zur Prozessorarchitektur für die Plattform, die dieses Manifest unterstützt. Zu den allgemeinen Architekturdetails gehören die folgenden Werte:

- `arm61`
- `arm71`
- `arm81`

`key`

(Optional) Ein Plattformattribut, das Sie für dieses Manifest definieren. Ersetzen Sie *`key`* durch den Namen des Plattformattributs. Die AWS IoT Greengrass Core-Software ordnet dieses Plattformattribut den Schlüssel-Wert-Paaren zu, die Sie in der Greengrass

Nucleus-Komponentenkonfiguration angeben. Weitere Informationen finden Sie unter dem Parameter Platform Overrides der Greengrass Nucleus-Komponente.

> **Tip**
>
> Verwenden Sie das umgekehrte Domainnamenformat, um Namenskonflikte innerhalb Ihres Unternehmens zu vermeiden. Wenn Ihr Unternehmen beispielsweise Eigentümer eines Radioprojekts ist `example.com` und Sie daran arbeiten, können Sie ein benutzerdefiniertes Plattformattribut `com.example.radio.RadioModule` benennen. Auf diese Weise können Konflikte mit den Namen von Plattformattributen innerhalb Ihres Unternehmens vermieden werden.

Sie können beispielsweise ein Plattformattribut definieren`com.example.radio.RadioModule`, um ein anderes Manifest zu spezifizieren, das darauf basiert, welches Funkmodul auf einem Kerngerät verfügbar ist. Jedes Manifest kann unterschiedliche Artefakte enthalten, die für unterschiedliche Hardwarekonfigurationen gelten, sodass Sie die minimale Menge an Software auf dem Kerngerät bereitstellen.

`Lifecycle`

Ein Objekt oder eine Zeichenfolge, die definiert, wie die Komponente auf der Plattform installiert und ausgeführt wird, die dieses Manifest definiert. Sie können auch einen globalen Lebenszyklus definieren, der für alle Plattformen gilt. Das Kerngerät verwendet den globalen Lebenszyklus nur, wenn das zu verwendende Manifest keinen Lebenszyklus spezifiziert.

> **Note**
>
> Sie definieren diesen Lebenszyklus in einem Manifest. Die Lebenszyklusschritte, die Sie hier angeben, gelten nur für die Plattform, die dieses Manifest definiert. Sie können auch einen globalen Lebenszyklus definieren, der für alle Plattformen gilt.

Dieses Objekt oder diese Zeichenfolge enthält die folgenden Informationen:

`Setenv`

(Optional) Ein Wörterbuch mit Umgebungsvariablen, das allen Lebenszyklusskripten zur Verfügung gestellt wird. Sie können diese Umgebungsvariablen `Setenv` in jedem Lebenszyklusskript überschreiben.

`install`

(Optional) Ein Objekt oder eine Zeichenfolge, die das Skript definiert, das bei der Installation der Komponente ausgeführt werden soll. Die AWS IoT Greengrass Core-Software führt diesen Lebenszyklusschritt auch bei jedem Start der Software aus.

Wenn das `install` Skript mit einem Erfolgscode beendet wird, wechselt die Komponente in den `INSTALLED` Status.

Dieses Objekt oder diese Zeichenfolge enthält die folgenden Informationen:

`Script`

Das auszuführende Skript.

`RequiresPrivilege`

(Optional) Sie können das Skript mit Root-Rechten ausführen. Wenn Sie diese Option auf setzen`true`, führt die AWS IoT Greengrass Core-Software dieses Lifecycle-Skript als Root aus und nicht als Systembenutzer, den Sie für die Ausführung dieser Komponente konfigurieren. Standardeinstellung: `false`.

`Skipif`

(Optional) Die Prüfung, um festzustellen, ob das Skript ausgeführt werden soll oder nicht. Sie können festlegen, dass geprüft wird, ob sich eine ausführbare Datei im Pfad befindet oder ob eine Datei existiert. Wenn die Ausgabe wahr ist, überspringt die AWS IoT Greengrass Core-Software den Schritt. Wählen Sie eine der folgenden Prüfungen:

- `onpath` *`runnable`*— Prüft, ob sich ein Runnable im Systempfad befindet. Verwenden Sie dies beispielsweise, **`onpath python3`** um diesen Lebenszyklusschritt zu überspringen, wenn Python 3 verfügbar ist.
- `exists` *`file`*— Prüft, ob eine Datei existiert. Verwenden Sie zum Beispiel, **`exists /tmp/my-configuration.db`** um diesen Lebenszyklusschritt zu überspringen, falls `/tmp/my-configuration.db` vorhanden.

`Timeout`

(Optional) Die maximale Zeit in Sekunden, die das Skript ausführen kann, bevor die AWS IoT Greengrass Core-Software den Prozess beendet.

Standard: 120 Sekunden

`Setenv`

(Optional) Das Wörterbuch der Umgebungsvariablen, die dem Skript zur Verfügung gestellt werden sollen. Diese Umgebungsvariablen überschreiben die Variablen, die Sie in angeben`Lifecycle.Setenv`.

`run`

(Optional) Ein Objekt oder eine Zeichenfolge, die das Skript definiert, das beim Start der Komponente ausgeführt werden soll.

Die Komponente wechselt in den `RUNNING` Status, wenn dieser Lebenszyklusschritt ausgeführt wird. Wenn das `run` Skript mit einem Erfolgscode beendet wird, wechselt die Komponente in den `STOPPING` Status. Wenn ein `shutdown` Skript angegeben ist, wird es ausgeführt; andernfalls wechselt die Komponente in den `FINISHED` Status.

Komponenten, die von dieser Komponente abhängen, werden gestartet, wenn dieser Lebenszyklusschritt ausgeführt wird. Um einen Hintergrundprozess auszuführen, z. B. einen Dienst, den abhängige Komponenten verwenden, verwenden Sie stattdessen den `startup` Lebenszyklusschritt.

Wenn Sie Komponenten mit einem `run` Lebenszyklus bereitstellen, kann das Kerngerät die Bereitstellung als abgeschlossen melden, sobald dieses Lifecycle-Skript ausgeführt wird. Dadurch kann die Bereitstellung abgeschlossen und erfolgreich sein, auch wenn das `run` Lifecycle-Skript kurz nach der Ausführung fehlschlägt. Wenn der Bereitstellungsstatus vom Ergebnis des Startskripts der Komponente abhängen soll, verwenden Sie stattdessen den `startup` Lebenszyklusschritt.

> **Note**
>
> Sie können nur einen `startup` oder einen `run` Lebenszyklus definieren.

Dieses Objekt oder diese Zeichenfolge enthält die folgenden Informationen:

`Script`

Das auszuführende Skript.

`RequiresPrivilege`

(Optional) Sie können das Skript mit Root-Rechten ausführen. Wenn Sie diese Option auf setzen`true`, führt die AWS IoT Greengrass Core-Software dieses Lifecycle-Skript als Root aus und nicht als Systembenutzer, den Sie für die Ausführung dieser Komponente konfigurieren. Standardeinstellung: `false`.

`Skipif`

(Optional) Die Prüfung, um festzustellen, ob das Skript ausgeführt werden soll oder nicht. Sie können festlegen, dass geprüft wird, ob sich eine ausführbare Datei im Pfad befindet oder ob eine Datei existiert. Wenn die Ausgabe wahr ist, überspringt die AWS IoT Greengrass Core-Software den Schritt. Wählen Sie eine der folgenden Prüfungen:

- `onpath` *`runnable`*— Prüft, ob sich ein Runnable im Systempfad befindet. Verwenden Sie dies beispielsweise, **`onpath python3`** um diesen Lebenszyklusschritt zu überspringen, wenn Python 3 verfügbar ist.
- `exists` *`file`*— Prüft, ob eine Datei existiert. Verwenden Sie zum Beispiel, **`exists /tmp/my-configuration.db`** um diesen Lebenszyklusschritt zu überspringen, falls `/tmp/my-configuration.db` vorhanden.

`Timeout`

(Optional) Die maximale Zeit in Sekunden, die das Skript ausführen kann, bevor die AWS IoT Greengrass Core-Software den Prozess beendet.

Für diesen Lebenszyklusschritt gibt es standardmäßig kein Timeout. Wenn Sie dieses Timeout weglassen, wird das `run` Skript ausgeführt, bis es beendet wird.

`Setenv`

(Optional) Das Wörterbuch der Umgebungsvariablen, die dem Skript zur Verfügung gestellt werden sollen. Diese Umgebungsvariablen überschreiben die Variablen, die Sie in angeben`Lifecycle.Setenv`.

`startup`

(Optional) Ein Objekt oder eine Zeichenfolge, die den Hintergrundprozess definiert, der ausgeführt werden soll, wenn die Komponente gestartet wird.

Wird verwendet`startup`, um einen Befehl auszuführen, der erfolgreich beendet oder der Status der Komponente aktualisiert werden muss, `RUNNING` bevor abhängige Komponenten gestartet werden können. Verwenden Sie den UpdateStateIPC-Vorgang, um den Status der Komponente auf `RUNNING` oder zu setzen, `ERRORED` wenn die Komponente ein Skript startet, das nicht beendet wird. Sie könnten beispielsweise einen `startup` Schritt definieren, mit dem der MySQL-Prozess gestartet wird`/etc/init.d/mysqld start`.

Die Komponente wechselt in den `STARTING` Status, wenn dieser Lebenszyklusschritt ausgeführt wird. Wenn das `startup` Skript mit einem Erfolgscode beendet wird, wechselt die Komponente in den `RUNNING` Status. Dann können abhängige Komponenten gestartet werden.

Wenn Sie Komponenten mit einem `startup` Lebenszyklus bereitstellen, kann das Kerngerät die Bereitstellung als abgeschlossen melden, nachdem dieses Lifecycle-Skript beendet wurde oder seinen Status gemeldet hat. Mit anderen Worten, der Status der Bereitstellung ist so lange gültig, `IN_PROGRESS` bis die Startskripts aller Komponenten beendet werden oder einen Status melden.

> **Note**
>
> Sie können nur einen `startup` oder einen `run` Lebenszyklus definieren.

Dieses Objekt oder diese Zeichenfolge enthält die folgenden Informationen:

`Script`

Das auszuführende Skript.

`RequiresPrivilege`

(Optional) Sie können das Skript mit Root-Rechten ausführen. Wenn Sie diese Option auf setzen`true`, führt die AWS IoT Greengrass Core-Software dieses Lifecycle-Skript als Root aus und nicht als Systembenutzer, den Sie für die Ausführung dieser Komponente konfigurieren. Standardeinstellung: `false`.

`Skipif`

(Optional) Die Prüfung, um festzustellen, ob das Skript ausgeführt werden soll oder nicht. Sie können festlegen, dass geprüft wird, ob sich eine ausführbare Datei im Pfad

befindet oder ob eine Datei existiert. Wenn die Ausgabe wahr ist, überspringt die AWS IoT Greengrass Core-Software den Schritt. Wählen Sie eine der folgenden Prüfungen:

- `onpath` *`runnable`*— Prüft, ob sich ein Runnable im Systempfad befindet. Verwenden Sie dies beispielsweise, **`onpath python3`** um diesen Lebenszyklusschritt zu überspringen, wenn Python 3 verfügbar ist.
- `exists` *`file`*— Prüft, ob eine Datei existiert. Verwenden Sie zum Beispiel, **`exists /tmp/my-configuration.db`** um diesen Lebenszyklusschritt zu überspringen, falls `/tmp/my-configuration.db` vorhanden.

`Timeout`

(Optional) Die maximale Zeit in Sekunden, die das Skript ausführen kann, bevor die AWS IoT Greengrass Core-Software den Prozess beendet.

Standard: 120 Sekunden

`Setenv`

(Optional) Das Wörterbuch der Umgebungsvariablen, die dem Skript zur Verfügung gestellt werden sollen. Diese Umgebungsvariablen überschreiben die Variablen, die Sie in angeben`Lifecycle.Setenv`.

`shutdown`

(Optional) Ein Objekt oder eine Zeichenfolge, die das Skript definiert, das ausgeführt werden soll, wenn die Komponente heruntergefahren wird. Verwenden Sie den Shutdown-Lebenszyklus, um Code auszuführen, den Sie ausführen möchten, wenn sich die Komponente im `STOPPING` Status befindet. Der Shutdown-Lebenszyklus kann verwendet werden, um einen Prozess zu stoppen, der von den `run` Skripten `startup` oder gestartet wurde.

Wenn Sie einen Hintergrundprozess in starten`startup`, verwenden Sie diesen `shutdown` Schritt, um diesen Prozess zu beenden, wenn die Komponente heruntergefahren wird. Sie könnten beispielsweise einen `shutdown` Schritt definieren, mit dem der MySQL-Prozess gestoppt wird`/etc/init.d/mysqld stop`.

Das `shutdown` Skript wird ausgeführt, nachdem die Komponente den `STOPPING` Status erreicht hat. Wenn das Skript erfolgreich abgeschlossen wurde, wechselt die Komponente in den `FINISHED` Status.

Dieses Objekt oder diese Zeichenfolge enthält die folgenden Informationen:

`Script`

Das auszuführende Skript.

`RequiresPrivilege`

(Optional) Sie können das Skript mit Root-Rechten ausführen. Wenn Sie diese Option auf setzen`true`, führt die AWS IoT Greengrass Core-Software dieses Lifecycle-Skript als Root aus und nicht als Systembenutzer, den Sie für die Ausführung dieser Komponente konfigurieren. Standardeinstellung: `false`.

`Skipif`

(Optional) Die Prüfung, um festzustellen, ob das Skript ausgeführt werden soll oder nicht. Sie können festlegen, dass geprüft wird, ob sich eine ausführbare Datei im Pfad befindet oder ob eine Datei existiert. Wenn die Ausgabe wahr ist, überspringt die AWS IoT Greengrass Core-Software den Schritt. Wählen Sie eine der folgenden Prüfungen:

- `onpath` *`runnable`*— Prüft, ob sich ein Runnable im Systempfad befindet. Verwenden Sie dies beispielsweise, **`onpath python3`** um diesen Lebenszyklusschritt zu überspringen, wenn Python 3 verfügbar ist.
- `exists` *`file`*— Prüft, ob eine Datei existiert. Verwenden Sie zum Beispiel, **`exists /tmp/my-configuration.db`** um diesen Lebenszyklusschritt zu überspringen, falls `/tmp/my-configuration.db` vorhanden.

`Timeout`

(Optional) Die maximale Zeit in Sekunden, die das Skript ausführen kann, bevor die AWS IoT Greengrass Core-Software den Prozess beendet.

Standard: 15 Sekunden.

`Setenv`

(Optional) Das Wörterbuch der Umgebungsvariablen, die dem Skript zur Verfügung gestellt werden sollen. Diese Umgebungsvariablen überschreiben die Variablen, die Sie in angeben`Lifecycle.Setenv`.

`recover`

(Optional) Ein Objekt oder eine Zeichenfolge, die das Skript definiert, das ausgeführt werden soll, wenn die Komponente auf einen Fehler stößt.

Dieser Schritt wird ausgeführt, wenn eine Komponente in den `ERRORED` Status wechselt. Wenn die Komponente `ERRORED` dreimal in den Status wechselt, ohne dass sie erfolgreich wiederhergestellt werden konnte, wechselt die Komponente in den `BROKEN` Status. Um eine `BROKEN` Komponente zu reparieren, müssen Sie sie erneut bereitstellen.

Dieses Objekt oder diese Zeichenfolge enthält die folgenden Informationen:

`Script`

Das auszuführende Skript.

`RequiresPrivilege`

(Optional) Sie können das Skript mit Root-Rechten ausführen. Wenn Sie diese Option auf setzen`true`, führt die AWS IoT Greengrass Core-Software dieses Lifecycle-Skript als Root aus und nicht als Systembenutzer, den Sie für die Ausführung dieser Komponente konfigurieren. Standardeinstellung: `false`.

`Skipif`

(Optional) Die Prüfung, um festzustellen, ob das Skript ausgeführt werden soll oder nicht. Sie können festlegen, dass geprüft wird, ob sich eine ausführbare Datei im Pfad befindet oder ob eine Datei existiert. Wenn die Ausgabe wahr ist, überspringt die AWS IoT Greengrass Core-Software den Schritt. Wählen Sie eine der folgenden Prüfungen:

- `onpath` *`runnable`*— Prüft, ob sich ein Runnable im Systempfad befindet. Verwenden Sie dies beispielsweise, **`onpath python3`** um diesen Lebenszyklusschritt zu überspringen, wenn Python 3 verfügbar ist.
- `exists` *`file`*— Prüft, ob eine Datei existiert. Verwenden Sie zum Beispiel, **`exists /tmp/my-configuration.db`** um diesen Lebenszyklusschritt zu überspringen, falls `/tmp/my-configuration.db` vorhanden.

`Timeout`

(Optional) Die maximale Zeit in Sekunden, die das Skript ausführen kann, bevor die AWS IoT Greengrass Core-Software den Prozess beendet.

Standard: 60 Sekunden.

`Setenv`

(Optional) Das Wörterbuch der Umgebungsvariablen, die dem Skript zur Verfügung gestellt werden sollen. Diese Umgebungsvariablen überschreiben die Variablen, die Sie in angeben`Lifecycle.Setenv`.

`bootstrap`

(Optional) Ein Objekt oder eine Zeichenfolge, die ein Skript definiert, für das die AWS IoT Greengrass Core-Software oder das Core-Gerät neu gestartet werden muss. Auf diese Weise können Sie eine Komponente entwickeln, die beispielsweise nach der Installation von Betriebssystem- oder Runtime-Updates einen Neustart durchführt.

> **Note**
>
> Verwenden Sie den Installationslebenszyklus, um Updates oder Abhängigkeiten zu installieren, für die kein Neustart der AWS IoT Greengrass Core-Software oder des Core-Geräts erforderlich ist.

Dieser Lebenszyklusschritt wird in den folgenden Fällen, wenn die AWS IoT Greengrass Core-Software die Komponente bereitstellt, vor dem Schritt des Installationszyklus ausgeführt:

- Die Komponente wird zum ersten Mal auf dem Kerngerät bereitgestellt.
- Die Version der Komponente ändert sich.
- Das Bootstrap-Skript ändert sich als Ergebnis eines Updates der Komponentenkonfiguration.

Nachdem die AWS IoT Greengrass Core-Software den Bootstrap-Schritt für alle Komponenten abgeschlossen hat, die in einer Bereitstellung über einen Bootstrap-Schritt verfügen, wird die Software neu gestartet.

> **Important**
>
> Sie müssen die AWS IoT Greengrass Core-Software als Systemdienst konfigurieren, um die AWS IoT Greengrass Core-Software oder das Core-Gerät neu zu starten. Wenn Sie die AWS IoT Greengrass Core-Software nicht als Systemdienst konfigurieren, wird die Software nicht neu gestartet. Weitere Informationen finden Sie unter Den Greengrass Nucleus als Systemdienst konfigurieren.

Dieses Objekt oder diese Zeichenfolge enthält die folgenden Informationen:

`BootstrapOnRollback`

> **Note**
>
> Wenn diese Funktion aktiviert ist, `BootstrapOnRollback` wird sie nur für Komponenten ausgeführt, die die Bootstrap-Lebenszyklusschritte im Rahmen einer fehlgeschlagenen Zielbereitstellung entweder abgeschlossen haben oder versucht haben, sie auszuführen. Diese Funktion ist für Greengrass Nucleus-Versionen 2.12.0 und höher verfügbar.

(Optional) Sie können die Bootstrap-Lebenszyklusschritte als Teil einer Rollback-Bereitstellung ausführen. Wenn Sie diese Option auf setzen`true`, werden die in einer Rollback-Bereitstellung definierten Bootstrap-Lebenszyklusschritte ausgeführt. Wenn eine Bereitstellung fehlschlägt, wird die vorherige Version des Komponenten-Bootstrap-Lebenszyklus während einer Rollback-Bereitstellung erneut ausgeführt.

Standardeinstellung: `false`.

`Script`

Das auszuführende Skript. Der Exit-Code dieses Skripts definiert die Neustart-Anweisung. Verwenden Sie die folgenden Exit-Codes:

- `0`— Starten Sie die AWS IoT Greengrass Core-Software oder das Core-Gerät nicht neu. Die AWS IoT Greengrass Core-Software wird nach dem Bootstrap aller Komponenten immer noch neu gestartet.
- `100`— Aufforderung zum Neustart der AWS IoT Greengrass Core-Software.
- `101`— Aufforderung zum Neustart des Core-Geräts.

Die Exit-Codes 100 bis 199 sind für besonderes Verhalten reserviert. Andere Exit-Codes stehen für Skriptfehler.

`RequiresPrivilege`

(Optional) Sie können das Skript mit Root-Rechten ausführen. Wenn Sie diese Option auf setzen`true`, führt die AWS IoT Greengrass Core-Software dieses Lifecycle-Skript als Root aus und nicht als Systembenutzer, den Sie für die Ausführung dieser Komponente konfigurieren. Standardeinstellung: `false`.

`Timeout`

(Optional) Die maximale Zeit in Sekunden, die das Skript ausführen kann, bevor die AWS IoT Greengrass Core-Software den Prozess beendet.

Standard: 120 Sekunden

`Setenv`

(Optional) Das Wörterbuch der Umgebungsvariablen, die dem Skript zur Verfügung gestellt werden sollen. Diese Umgebungsvariablen überschreiben die Variablen, die Sie in angeben`Lifecycle.Setenv`.

`Selections`

(Optional) Eine Liste von Auswahlschlüsseln, die Abschnitte des globalen Lebenszyklus angeben, die für dieses Manifest ausgeführt werden sollen. Im globalen Lebenszyklus können Sie Lebenszyklusschritte mit Auswahlschlüsseln auf jeder Ebene definieren, um Unterabschnitte des Lebenszyklus auszuwählen. Anschließend verwendet das Kerngerät die Abschnitte, die den Auswahltasten in diesem Manifest entsprechen. Weitere Informationen finden Sie in den globalen Lebenszyklus-Beispielen.

> **Important**
>
> Das Kerngerät verwendet die Auswahlen aus dem globalen Lebenszyklus nur, wenn dieses Manifest keinen Lebenszyklus definiert.

Sie können den `all` Auswahlschlüssel angeben, um Abschnitte des globalen Lebenszyklus auszuführen, für die es keine Auswahltasten gibt.

`Artifacts`

(Optional) Eine Liste von Objekten, die jeweils ein binäres Artefakt für die Komponente auf der Plattform definieren, die dieses Manifest definiert. Sie können beispielsweise Code oder Bilder als Artefakte definieren.

Wenn die Komponente bereitgestellt wird, lädt die AWS IoT Greengrass Core-Software das Artefakt in einen Ordner auf dem Core-Gerät herunter. Sie können Artefakte auch als Archivdateien definieren, die die Software nach dem Herunterladen extrahiert.

Sie können Rezeptvariablen verwenden, um die Pfade zu den Ordnern abzurufen, in denen die Artefakte auf dem Kerngerät installiert werden.

- Normale Dateien — Verwenden Sie die Rezeptvariable artifacts:path, um den Pfad zu dem Ordner zu ermitteln, der die Artefakte enthält. Geben Sie beispielsweise `{artifacts:path}/my_script.py` in einem Rezept an, dass der Pfad zu einem Artefakt abgerufen werden soll, das den URI hat. `s3://DOC-EXAMPLE-BUCKET/path/to/my_script.py`
- Extrahierte Archive — Verwenden Sie die Rezeptvariable artifacts:decompressedPath, um den Pfad zu dem Ordner abzurufen, der die extrahierten Archivartefakte enthält. Die AWS IoT Greengrass Core-Software extrahiert jedes Archiv in einen Ordner mit demselben Namen wie das Archiv. Geben Sie beispielsweise `{artifacts:decompressedPath}/my_archive/my_script.py` in einem Rezept an, dass der Pfad zu `my_script.py` dem Archivartefakt abgerufen werden soll, das den URI `s3://DOC-EXAMPLE-BUCKET/path/to/my_archive.zip` enthält.

> **Note**
>
> Wenn Sie eine Komponente mit einem Archivartefakt auf einem lokalen Kerngerät entwickeln, haben Sie möglicherweise keinen URI für dieses Artefakt. Um Ihre Komponente mit einer `Unarchive` Option zu testen, die das Artefakt extrahiert, geben Sie eine URI an, deren Dateiname mit dem Namen Ihrer Archivartefaktdatei übereinstimmt. Sie können den URI angeben, an den Sie das Archivartefakt hochladen möchten, oder Sie können einen neuen Platzhalter-URI angeben. Um das `my_archive.zip` Artefakt beispielsweise während einer lokalen Bereitstellung zu extrahieren, können Sie Folgendes angeben. `s3://DOC-EXAMPLE-BUCKET/my_archive.zip`

Jedes Objekt enthält die folgenden Informationen:

`URI`

Die URI eines Artefakts in einem S3-Bucket. Die AWS IoT Greengrass Core-Software ruft das Artefakt bei der Installation der Komponente von diesem URI ab, sofern das Artefakt nicht bereits auf dem Gerät vorhanden ist. Jedes Artefakt muss in jedem Manifest einen eindeutigen Dateinamen haben.

`Unarchive`

(Optional) Der Typ des zu entpackenden Archivs. Wählen Sie aus den folgenden Optionen aus:

- `NONE`— Die Datei ist kein Archiv zum Entpacken. Die AWS IoT Greengrass Core-Software installiert das Artefakt in einem Ordner auf dem Core-Gerät. Sie können die Rezeptvariable artifacts:path verwenden, um den Pfad zu diesem Ordner zu ermitteln.
- `ZIP`— Die Datei ist ein ZIP-Archiv. Die AWS IoT Greengrass Core-Software extrahiert das Archiv in einen Ordner mit demselben Namen wie das Archiv. Sie können die Rezeptvariable artifacts:decompressedPath verwenden, um den Pfad zu dem Ordner zu ermitteln, der diesen Ordner enthält.

Standardeinstellung: `NONE`.

`Permission`

(Optional) Ein Objekt, das die Zugriffsberechtigungen definiert, die für diese Artefaktdatei festgelegt werden sollen. Sie können die Leseberechtigung und die Ausführungsberechtigung festlegen.

> **Note**
>
> Sie können die Schreibberechtigung nicht festlegen, da die AWS IoT Greengrass Core-Software es Komponenten nicht erlaubt, Artefaktdateien im Artefaktordner zu bearbeiten. Um eine Artefaktdatei in einer Komponente zu bearbeiten, kopieren Sie sie an einen anderen Speicherort oder veröffentlichen Sie eine neue Artefaktdatei und stellen Sie sie bereit.

Wenn Sie ein Artefakt als ein zu entpackendes Archiv definieren, legt die AWS IoT Greengrass Core-Software diese Zugriffsberechtigungen für die Dateien fest, die sie aus dem Archiv entpackt. Die AWS IoT Greengrass Core-Software legt die Zugriffsberechtigungen für den Ordner auf `ALL` für `Read` und fest. `Execute` Dadurch können Komponenten die entpackten Dateien im Ordner anzeigen. Um Berechtigungen für einzelne Dateien aus dem Archiv festzulegen, können Sie die Berechtigungen im Install-Lifecycle-Skript festlegen.

Dieses Objekt enthält die folgenden Informationen:

`Read`

(Optional) Die Leseberechtigung, die für diese Artefaktdatei festgelegt werden soll. Geben Sie Folgendes an, damit andere Komponenten auf dieses Artefakt zugreifen

können, z. B. Komponenten, die von dieser Komponente abhängen. `ALL` Wählen Sie aus den folgenden Optionen aus:

- `NONE`— Die Datei ist nicht lesbar.
- `OWNER`— Die Datei ist für den Systembenutzer lesbar, den Sie für die Ausführung dieser Komponente konfigurieren.
- `ALL`— Die Datei ist für alle Benutzer lesbar.

Standardeinstellung: `OWNER`.

`Execute`

(Optional) Die Ausführungsberechtigung, die für diese Artefaktdatei festgelegt werden soll. Die `Execute` Erlaubnis impliziert die `Read` Erlaubnis. Wenn Sie beispielsweise `ALL` for angeben`Execute`, können alle Benutzer diese Artefaktdatei lesen und ausführen.

Wählen Sie aus den folgenden Optionen aus:

- `NONE`— Die Datei kann nicht ausgeführt werden.
- `OWNER`— Die Datei kann von dem Systembenutzer ausgeführt werden, den Sie für die Ausführung der Komponente konfigurieren.
- `ALL`— Die Datei kann von allen Benutzern ausgeführt werden.

Standardeinstellung: `NONE`.

`Digest`

(Schreibgeschützt) Der kryptografische Digest-Hash des Artefakts. Wenn Sie eine Komponente erstellen, AWS IoT Greengrass verwendet einen Hash-Algorithmus, um einen Hash der Artefaktdatei zu berechnen. Wenn Sie dann die Komponente bereitstellen, berechnet der Greengrass-Kern den Hash des heruntergeladenen Artefakts und vergleicht den Hash mit diesem Digest, um das Artefakt vor der Installation zu verifizieren. Wenn der Hash nicht mit dem Digest übereinstimmt, schlägt die Bereitstellung fehl.

Wenn Sie diesen Parameter festlegen, AWS IoT Greengrass ersetzt er den Wert, den Sie bei der Erstellung der Komponente festgelegt haben.

`Algorithm`

(Schreibgeschützt) Der AWS IoT Greengrass Hash-Algorithmus, mit dem der Digest-Hash des Artefakts berechnet wird.

Wenn Sie diesen Parameter festlegen, AWS IoT Greengrass ersetzt er den Wert, den Sie bei der Erstellung der Komponente festgelegt haben.

`Lifecycle`

Ein Objekt, das definiert, wie die Komponente installiert und ausgeführt wird. Das Kerngerät verwendet den globalen Lebenszyklus nur, wenn das zu verwendende Manifest keinen Lebenszyklus spezifiziert.

> **Note**
>
> Sie definieren diesen Lebenszyklus außerhalb eines Manifests. Sie können auch einen Manifest-Lebenszyklus definieren, der für die Plattformen gilt, die diesem Manifest entsprechen.

Im globalen Lebenszyklus können Sie Lebenszyklen angeben, die für bestimmte Auswahlschlüssel ausgeführt werden, die Sie in jedem Manifest angeben. Auswahlschlüssel sind Zeichenfolgen, die Abschnitte des globalen Lebenszyklus identifizieren, die für jedes Manifest ausgeführt werden sollen.

Die `all` Auswahltaste ist die Standardeinstellung für alle Abschnitte ohne Auswahltaste. Das bedeutet, dass Sie den `all` Auswahlschlüssel in einem Manifest angeben können, um die Abschnitte des globalen Lebenszyklus ohne Auswahlschlüssel auszuführen. Sie müssen den `all` Auswahlschlüssel im globalen Lebenszyklus nicht angeben.

Wenn ein Manifest keinen Lebenszyklus oder keine Auswahlschlüssel definiert, verwendet das Kerngerät standardmäßig die `all` Auswahl. Das bedeutet, dass das Kerngerät in diesem Fall die Abschnitte des globalen Lebenszyklus verwendet, die keine Auswahltasten verwenden.

Dieses Objekt enthält dieselben Informationen wie der Manifest-Lebenszyklus, Sie können jedoch Auswahlschlüssel auf jeder Ebene angeben, um Unterabschnitte des Lebenszyklus auszuwählen.

> **Tip**
>
> Es wird empfohlen, für jede Auswahltaste nur Kleinbuchstaben zu verwenden, um Konflikte zwischen Auswahlschlüsseln und Lebenszyklusschlüsseln zu vermeiden. Lebenszyklusschlüssel beginnen mit einem Großbuchstaben.

Example Beispiel für einen globalen Lebenszyklus mit Auswahlschlüsseln der obersten Ebene

```
Lifecycle:
  key1:
    install:
      Skipif: either onpath executable or exists file
      Script: command1
  key2:
    install:
      Script: command2
  all:
    install:
      Script: command3
```

Example Beispiel für einen globalen Lebenszyklus mit Auswahltasten auf unterster Ebene

```
Lifecycle:
  install:
    Script:
      key1: command1
      key2: command2
      all: command3
```

Example Beispiel für einen globalen Lebenszyklus mit mehreren Ebenen von Auswahlschlüsseln

```
Lifecycle:
  key1:
    install:
      Skipif: either onpath executable or exists file
      Script: command1
  key2:
    install:
      Script: command2
  all:
    install:
      Script:
        key3: command3
        key4: command4
        all: command5
```

Rezeptvariablen

Rezeptvariablen stellen Informationen aus der aktuellen Komponente und dem aktuellen Kern zur Verfügung, die Sie in Ihren Rezepten verwenden können. Sie können beispielsweise eine Rezeptvariable verwenden, um Komponentenkonfigurationsparameter an eine Anwendung zu übergeben, die Sie in einem Lifecycle-Skript ausführen.

Sie können Rezeptvariablen in den folgenden Abschnitten von Komponentenrezepten verwenden:

- Lebenszyklus-Definitionen.
- Definitionen der Komponentenkonfiguration, wenn Sie Greengrass Nucleus v2.6.0 oder höher verwenden und die interpolateComponentConfigurationKonfigurationsoption auf setzen. `true` Sie können auch Rezeptvariablen verwenden, wenn Sie Updates für die Komponentenkonfiguration bereitstellen.

Rezeptvariablen verwenden `{recipe_variable}` Syntax. Die geschweiften Klammern stehen für eine Rezeptvariable.

AWS IoT Greengrass unterstützt die folgenden Rezeptvariablen:

`component_dependency_name``:configuration:`*`json_pointer`*

Der Wert eines Konfigurationsparameters für die Komponente, die dieses Rezept definiert, oder für eine Komponente, von der diese Komponente abhängt.

Sie können diese Variable verwenden, um einen Parameter für ein Skript bereitzustellen, das Sie im Lebenszyklus der Komponente ausführen.

> **Note**
>
> AWS IoT Greengrass unterstützt diese Rezeptvariable nur in Definitionen des Komponentenlebenszyklus.

Diese Rezeptvariable hat die folgenden Eingaben:

- `component_dependency_name`— (Optional) Der Name der Komponentenabhängigkeit, die abgefragt werden soll. Lassen Sie dieses Segment weg, um die Komponente abzufragen, die dieses Rezept definiert. Sie können nur direkte Abhängigkeiten angeben.

- `json_pointer`— Der JSON-Zeiger auf den auszuwertenden Konfigurationswert. JSON-Zeiger beginnen mit einem / Schrägstrich. Um einen Wert in einer verschachtelten Komponentenkonfiguration zu identifizieren, verwenden Sie Schrägstriche (`/`), um die Schlüssel für jede Ebene in der Konfiguration voneinander zu trennen. Sie können eine Zahl als Schlüssel verwenden, um einen Index in einer Liste anzugeben. Weitere Informationen finden Sie in der JSON-Zeigerspezifikation.

  AWS IoT Greengrass Core verwendet JSON-Zeiger für Rezepte im YAML-Format.

Der JSON-Pointer kann auf die folgenden Knotentypen verweisen:

- Ein Wertknoten. AWS IoT Greengrass Core ersetzt die Rezeptvariable durch die Zeichenkettendarstellung des Werts. Nullwerte werden in `null` eine Zeichenfolge umgewandelt.
- Ein Objektknoten. AWS IoT Greengrass Core ersetzt die Rezeptvariable durch die serialisierte JSON-Zeichenkettendarstellung dieses Objekts.
- Kein Knoten. AWS IoT Greengrass Core ersetzt die Rezeptvariable nicht.

Beispielsweise ruft die `{configuration:/Message}` Rezeptvariable den Wert des `Message` Schlüssels in der Komponentenkonfiguration ab. Die `{com.example.MyComponentDependency:configuration:/server/port}` Rezeptvariable ruft den Wert von `port` im `server` Konfigurationsobjekt einer Komponentenabhängigkeit ab.

`component_dependency_name``:artifacts:path`

Der Stammpfad der Artefakte für die Komponente, die dieses Rezept definiert, oder für eine Komponente, von der diese Komponente abhängt.

Wenn eine Komponente installiert wird, werden die Artefakte der Komponente in den Ordner AWS IoT Greengrass kopiert, den diese Variable verfügbar macht. Sie können diese Variable verwenden, um beispielsweise den Speicherort eines Skripts zu identifizieren, das im Lebenszyklus einer Komponente ausgeführt werden soll.

Der Ordner in diesem Pfad ist schreibgeschützt. Um Artefaktdateien zu ändern, kopieren Sie die Dateien an einen anderen Speicherort, z. B. in das aktuelle Arbeitsverzeichnis (`$PWD`oder). `.` Ändern Sie dann die Dateien dort.

Um ein Artefakt aus einer Komponentenabhängigkeit zu lesen oder auszuführen, müssen die entsprechenden `Execute` Rechte `Read` oder Rechte für das Artefakt vorliegen. `ALL` Weitere

Informationen finden Sie in den Artefaktberechtigungen, die Sie im Komponentenrezept definieren.

Diese Rezeptvariable hat die folgenden Eingaben:

- `component_dependency_name`— (Optional) Der Name der Komponentenabhängigkeit, die abgefragt werden soll. Lassen Sie dieses Segment weg, um die Komponente abzufragen, die dieses Rezept definiert. Sie können nur direkte Abhängigkeiten angeben.

`component_dependency_name``:artifacts:decompressedPath`

Der Stammpfad der dekomprimierten Archivartefakte für die Komponente, die dieses Rezept definiert, oder für eine Komponente, von der diese Komponente abhängt.

Wenn eine Komponente installiert wird, werden die Archivartefakte der Komponente in den Ordner AWS IoT Greengrass entpackt, den diese Variable verfügbar macht. Sie können diese Variable verwenden, um beispielsweise den Speicherort eines Skripts zu identifizieren, das im Lebenszyklus einer Komponente ausgeführt werden soll.

Jedes Artefakt wird in einen Ordner innerhalb des dekomprimierten Pfads entpackt, wobei der Ordner denselben Namen wie das Artefakt abzüglich seiner Erweiterung hat. Beispielsweise wird ein ZIP-Artefakt mit dem Namen in den Ordner entpackt. `models.zip` `{artifacts:decompressedPath}/models`

Der Ordner in diesem Pfad ist schreibgeschützt. Um Artefaktdateien zu ändern, kopieren Sie die Dateien an einen anderen Speicherort, z. B. in das aktuelle Arbeitsverzeichnis (`$PWD`oder). `.` Ändern Sie dann die Dateien dort.

Um ein Artefakt aus einer Komponentenabhängigkeit zu lesen oder auszuführen, müssen die entsprechenden `Execute` Rechte `Read` oder Rechte für das Artefakt vorliegen. `ALL` Weitere Informationen finden Sie in den Artefaktberechtigungen, die Sie im Komponentenrezept definieren.

Diese Rezeptvariable hat die folgenden Eingaben:

- `component_dependency_name`— (Optional) Der Name der Komponentenabhängigkeit, die abgefragt werden soll. Lassen Sie dieses Segment weg, um die Komponente abzufragen, die dieses Rezept definiert. Sie können nur direkte Abhängigkeiten angeben.

`component_dependency_name``:work:path`

Diese Funktion ist für Version 2.0.4 und höher der Greengrass Nucleus-Komponente verfügbar.

Der Arbeitspfad für die Komponente, die dieses Rezept definiert, oder für eine Komponente, von der diese Komponente abhängt. Der Wert dieser Rezeptvariablen entspricht der Ausgabe der `$PWD` Umgebungsvariablen und des Befehls pwd, wenn sie im Kontext der Komponente ausgeführt wird.

Sie können diese Rezeptvariable verwenden, um Dateien zwischen einer Komponente und einer Abhängigkeit gemeinsam zu nutzen.

Der Ordner in diesem Pfad ist für die Komponente, die dieses Rezept definiert, und für andere Komponenten, die unter demselben Benutzer und derselben Gruppe ausgeführt werden, lesbar und beschreibbar.

Diese Rezeptvariable hat die folgenden Eingaben:

- `component_dependency_name`— (Optional) Der Name der Komponentenabhängigkeit, die abgefragt werden soll. Lassen Sie dieses Segment weg, um die Komponente abzufragen, die dieses Rezept definiert. Sie können nur direkte Abhängigkeiten angeben.

`kernel:rootPath`

Der AWS IoT Greengrass Core-Stammpfad.

`iot:thingName`

Diese Funktion ist für Version 2.3.0 und höher der Greengrass Nucleus-Komponente verfügbar.

Der Name der Sache des Kerngeräts. AWS IoT

Beispiele für Rezepte

Anhand der folgenden Rezeptbeispiele können Sie Rezepte für Ihre Komponenten erstellen.

AWS IoT Greengrass kuratiert einen Index von Greengrass-Komponenten, den sogenannten Greengrass Software Catalog. Dieser Katalog verfolgt Greengrass-Komponenten, die von der Greengrass-Community entwickelt wurden. Aus diesem Katalog können Sie Komponenten herunterladen, ändern und bereitstellen, um Ihre Greengrass-Anwendungen zu erstellen. Weitere Informationen finden Sie unter Komponenten der Gemeinschaft.

Themen

- Rezept für die Komponenten von Hello World
- Beispiel für eine Python-Laufzeitkomponente
- Komponentenrezept, das mehrere Felder spezifiziert

Rezept für die Komponenten von Hello World

Das folgende Rezept beschreibt eine Hello World-Komponente, die ein Python-Skript ausführt. Diese Komponente unterstützt alle Plattformen und akzeptiert einen `Message` Parameter, der als Argument an das Python-Skript AWS IoT Greengrass übergeben wird. Dies ist das Rezept für die Hello World-Komponente im Tutorial Erste Schritte.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "My first AWS IoT Greengrass component.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "Message": "world"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/hello_world.py {configuration:/Message}"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "run": "py -3 -u {artifacts:path}/hello_world.py {configuration:/Message}"
      }
    }
  ]
}
```

YAML

```
---
```

```
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
ComponentDescription: My first AWS IoT Greengrass component.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    Message: world
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      run: |
        python3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
  - Platform:
      os: windows
    Lifecycle:
      run: |
        py -3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
```

Beispiel für eine Python-Laufzeitkomponente

Das folgende Rezept beschreibt eine Komponente, die Python installiert. Diese Komponente unterstützt 64-Bit-Linux-Geräte.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PythonRuntime",
  "ComponentDescription": "Installs Python 3.7",
  "ComponentPublisher": "Amazon",
  "ComponentVersion": "3.7.0",
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "architecture": "amd64"
      },
      "Lifecycle": {
        "install": "apt-get update\napt-get install python3.7"
      }
```

```
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PythonRuntime
ComponentDescription: Installs Python 3.7
ComponentPublisher: Amazon
ComponentVersion: '3.7.0'
Manifests:
  - Platform:
      os: linux
      architecture: amd64
    Lifecycle:
      install: |
        apt-get update
        apt-get install python3.7
```

Komponentenrezept, das mehrere Felder spezifiziert

Das folgende Komponentenrezept verwendet mehrere Rezeptfelder.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.FooService",
  "ComponentDescription": "Complete recipe for AWS IoT Greengrass components",
  "ComponentPublisher": "Amazon",
  "ComponentVersion": "1.0.0",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "TestParam": "TestValue"
    }
  },
  "ComponentDependencies": {
    "BarService": {
      "VersionRequirement": "^1.1.0",
      "DependencyType": "SOFT"
```

```
    },
    "BazService": {
      "VersionRequirement": "^2.0.0"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "architecture": "amd64"
      },
      "Lifecycle": {
        "install": {
          "Skipif": "onpath git",
          "Script": "sudo apt-get install git"
        },
        "Setenv": {
          "environment_variable1": "variable_value1",
          "environment_variable2": "variable_value2"
        }
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/hello_world.zip",
          "Unarchive": "ZIP"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/hello_world_linux.py"
        }
      ]
    },
    {
      "Lifecycle": {
        "install": {
          "Skipif": "onpath git",
          "Script": "sudo apt-get install git",
          "RequiresPrivilege": "true"
        }
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/hello_world.py"
        }
      ]
```

```
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.FooService
ComponentDescription: Complete recipe for AWS IoT Greengrass components
ComponentPublisher: Amazon
ComponentVersion: 1.0.0
ComponentConfiguration:
  DefaultConfiguration:
    TestParam: TestValue
ComponentDependencies:
  BarService:
    VersionRequirement: ^1.1.0
    DependencyType: SOFT
  BazService:
    VersionRequirement: ^2.0.0
Manifests:
  - Platform:
      os: linux
      architecture: amd64
    Lifecycle:
      install:
        Skipif: onpath git
        Script: sudo apt-get install git
      Setenv:
        environment_variable1: variable_value1
        environment_variable2: variable_value2
    Artifacts:
      - URI: 's3://DOC-EXAMPLE-BUCKET/hello_world.zip'
        Unarchive: ZIP
      - URI: 's3://DOC-EXAMPLE-BUCKET/hello_world_linux.py'
  - Lifecycle:
      install:
        Skipif: onpath git
        Script: sudo apt-get install git
        RequiresPrivilege: 'true'
    Artifacts:
      - URI: 's3://DOC-EXAMPLE-BUCKET/hello_world.py'
```

Referenz für die Umgebungsvariablen der Komponente aus der

DieAWS IoT Greengrass Core-Software legt Umgebungsvariablen fest, wenn sie Lebenszyklusskripte für Komponenten ausführt. Sie können diese Umgebungsvariablen in Ihren Komponenten abrufen,AWS-Region um den Dingnamen und die Greengrass-Nucleus-Version zu erhalten. Die Software legt auch Umgebungsvariablen fest, die Ihre Komponente benötigt, um das Interprozesskommunikations-SDK zu verwenden und mitAWS Diensten zu interagieren.

Sie können auch benutzerdefinierte Umgebungsvariablen für die Lebenszyklusskripten Ihrer Komponente festlegen. Weitere Informationen finden Sie unter Setenv.

DieAWS IoT Greengrass Core-Software enthält die folgenden Umgebungsvariablen:

`AWS_IOT_THING_NAME`

Der Name des Dings,AWS IoT das für dieses Greengrass-Core-Gerät steht.

`AWS_REGION`

AWS-RegionWo dieses Greengrass-Core-Gerät arbeitet.

DieAWS SDKs verwenden diese Umgebungsvariable, um die zu verwendende Standardregion zu identifizieren. Diese Variable entsprichtAWS_DEFAULT_REGION.

`AWS_DEFAULT_REGION`

AWS-RegionWo dieses Greengrass-Core-Gerät arbeitet.

DerAWS CLI verwendet diese Umgebungsvariable, um die zu verwendende Standardregion zu identifizieren. Diese Variable entsprichtAWS_REGION.

`GGC_VERSION`

Die Version der Greengrass-Nucleus-Komponente, die auf diesem Greengrass-Kerngerät läuft.

`GG_ROOT_CA_PATH`

Diese Funktion ist für Version 2.5.5 und höher der Greengrass Nucleus-Komponente verfügbar.

Der Pfad zum Zertifikat der Root Certificate Authority (CA), das der Greengrass-Nucleus verwendet.

`AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT`

Der Pfad zum IPC-Socket, den Komponenten für die Kommunikation mit derAWS IoT Greengrass Core-Software verwenden. Weitere Informationen finden Sie unter Verwenden Sie den AWS IoT

Device SDK , um mit dem Greengrass-Kern und anderen Komponenten zu kommunizieren und AWS IoT Core.

`SVCUID`

Das geheime Token, das Komponenten verwenden, um eine Verbindung zum IPC-Socket herzustellen und mit derAWS IoT Greengrass Core-Software zu kommunizieren. Weitere Informationen finden Sie unter Verwenden Sie den AWS IoT Device SDK , um mit dem Greengrass-Kern und anderen Komponenten zu kommunizieren und AWS IoT Core.

`AWS_CONTAINER_AUTHORIZATION_TOKEN`

Das geheime Token, das Komponenten verwenden, um Anmeldeinformationen von der Token-Exchange-Dienstkomponente abzurufen.

`AWS_CONTAINER_CREDENTIALS_FULL_URI`

Die URI, die Komponenten anfordern, um Anmeldeinformationen von der Token-Exchange-Dienstkomponente abzurufen.

Bereitstellen von AWS IoT Greengrass Komponenten auf Geräten

Sie können verwendenAWS IoT Greengrass, um Komponenten auf Geräten oder Gerätegruppen bereitzustellen. Sie verwenden Bereitstellungen, um die Komponenten und Konfigurationen zu definieren, die an die Geräte gesendet werden. AWS IoT Greengrass stellt für Ziele , AWS IoT Objekte oder Objektgruppen bereit, die Greengrass-Core-Geräte darstellen. AWS IoT Greengrass verwendet AWS IoT Core Aufträge zur Bereitstellung auf Ihren Core-Geräten. Sie können konfigurieren, wie der Auftrag auf Ihren Geräten bereitgestellt wird.

Bereitstellungen von Core-Geräten

Jedes Core-Gerät führt die Komponenten der Bereitstellungen für dieses Gerät aus. Eine neue Bereitstellung auf demselben Ziel überschreibt die vorherige Bereitstellung auf das Ziel. Wenn Sie eine Bereitstellung erstellen, definieren Sie die Komponenten und Konfigurationen, die auf die vorhandene Software des Core-Geräts angewendet werden sollen.

Wenn Sie eine Bereitstellung für ein Ziel überarbeiten, ersetzen Sie die Komponenten aus der vorherigen Revision durch die Komponenten in der neuen Revision. Sie stellen beispielsweise die Geheimer Manager Komponenten Protokollmanager und für die Objektgruppe bereit`TestGroup`. Anschließend erstellen Sie eine weitere Bereitstellung für `TestGroup`, die nur die Secret-Manager-

Komponente angibt. Daher führen die Core-Geräte in dieser Gruppe den Protokollmanager nicht mehr aus.

Auflösung der Plattformabhängigkeit

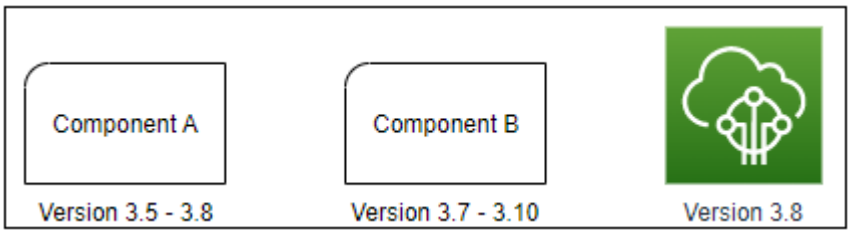
Wenn ein Core-Gerät eine Bereitstellung empfängt, überprüft es, ob die Komponenten mit dem Core-Gerät kompatibel sind. Wenn Sie beispielsweise auf Firehose einem Windows-Ziel bereitstellen, schlägt die Bereitstellung fehl.

Auflösung der Komponentenabhängigkeit

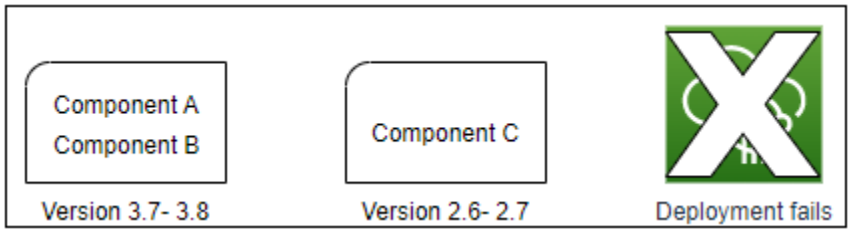
Das Core-Gerät prüft auch, ob alle Komponentenabhängigkeiten mit Versionseinschränkungen für Bereitstellungen anderer Komponenten in dieser Objektgruppe kompatibel sind. Wenn sich die Versionseinschränkungen für eine Komponente überschneiden, verwendet Greengrass die am stärksten anwendbare Version der Komponente. Beispielsweise:

- Sie stellen Komponente A in bereit`TestGroup`. Komponente A hängt von den `com.example.PythonRuntime` Komponentenversionen 3.5 bis 3.10 ab.
- Anschließend stellen Sie Komponente B in bereit`TestGroup`. Komponente B hängt von den `com.example.PythonRuntime` Komponentenversionen 3.7 bis 3.8 ab.

Daher stellen die Core-Geräte in `TestGroup` fest, dass sie Version 3.8 der `com.example.PythonRuntime` Komponente bereitstellen können, da diese Version die am besten anwendbare Version ist, bei der sich die Versionseinschränkungen überschneiden.



Anschließend stellen Sie Komponente C in bereit`TestGroup`. Komponente C hängt von den `com.example.PythonRuntime` Komponentenversionen 2.6 bis 2.7 ab. Diese Bereitstellung schlägt fehl, da es keine Komponentenversion gibt, die der Einschränkung 2.6–2.7 und 3.7–3.8 entspricht.



Entfernen eines Geräts aus einer Objektgruppe

Wenn Sie ein Core-Gerät aus einer Objektgruppe entfernen, hängt das Bereitstellungsverhalten der Komponente von der Version des Greengrass-Kerns ab, die auf dem Core-Gerät ausgeführt wird.

2.5.1 and later

Wenn Sie ein Core-Gerät aus einer Objektgruppe entfernen, hängt das Verhalten davon ab, ob die AWS IoT Richtlinie die `greengrass:ListThingGroupsForCoreDevice` Berechtigung erteilt. Weitere Informationen zu dieser Berechtigung und AWS IoT Richtlinien für -Core-Geräte finden Sie unter Geräteauthentifizierung und -autorisierung für AWS IoT Greengrass.

- Wenn die AWS IoT Richtlinie diese Berechtigung erteilt

  Wenn Sie ein Core-Gerät aus einer Objektgruppe entfernen, AWS IoT Greengrass entfernt die Komponenten der Objektgruppe, wenn das nächste Mal eine Bereitstellung auf dem Gerät durchgeführt wird. Wenn eine Komponente auf dem Gerät in der nächsten Bereitstellung enthalten ist, wird diese Komponente nicht vom Gerät entfernt.
- Wenn die AWS IoT Richtlinie diese Berechtigung nicht erteilt

  Wenn Sie ein Core-Gerät aus einer Objektgruppe entfernen, AWS IoT Greengrasslöscht die Komponenten dieser Objektgruppe nicht vom Gerät.

  Um eine Komponente von einem Gerät zu entfernen, verwenden Sie den Befehl deployment create der Greengrass-CLI. Geben Sie die zu entfernende Komponente mit dem `--remove` -Argument an und geben Sie die Objektgruppe mit dem `--groupId` -Argument an.

2.5.0

Wenn Sie ein Core-Gerät aus einer Objektgruppe entfernen, AWS IoT Greengrass entfernt die Komponenten der Objektgruppe, wenn das nächste Mal eine Bereitstellung auf dem Gerät durchgeführt wird. Wenn eine Komponente auf dem Gerät in der nächsten Bereitstellung enthalten ist, wird diese Komponente nicht vom Gerät entfernt.

Dieses Verhalten erfordert, dass die AWS IoT Richtlinie des Core-Geräts die `-greengrass:ListThingGroupsForCoreDevice`Berechtigung erteilt. Wenn ein Core-Gerät nicht über diese Berechtigung verfügt, kann das Core-Gerät keine Bereitstellungen anwenden. Weitere Informationen finden Sie unter Geräteauthentifizierung und -autorisierung für AWS IoT Greengrass.

2.0.x - 2.4.x

Wenn Sie ein Core-Gerät aus einer Objektgruppe entfernen, AWS IoT Greengrasslöscht die Komponenten dieser Objektgruppe nicht vom Gerät.

Um eine Komponente von einem Gerät zu entfernen, verwenden Sie den Befehl deployment create der Greengrass-CLI. Geben Sie die zu entfernende Komponente mit dem `--remove` -Argument an und geben Sie die Objektgruppe mit dem `--groupId` -Argument an.

Bereitstellungen

Bereitstellungen sind kontinuierlich. Wenn Sie eine Bereitstellung erstellen, AWS IoT Greengrass führt die Bereitstellung auf Zielgeräten aus, die online sind. Wenn ein Zielgerät nicht online ist, erhält es die Bereitstellung, wenn es das nächste Mal eine Verbindung zu herstelltAWS IoT Greengrass. Wenn Sie ein Core-Gerät zu einer Ziel-Objektgruppe hinzufügen, AWS IoT Greengrass sendet dem Gerät die neueste Bereitstellung für diese Objektgruppe.

Bevor ein Core-Gerät eine Komponente bereitstellt, benachrichtigt es standardmäßig jede Komponente auf dem Gerät. Greengrass-Komponenten können auf die Benachrichtigung antworten, um die Bereitstellung aufzuschieben. Möglicherweise möchten Sie die Bereitstellung verschieben, wenn das Gerät einen niedrigen Batteriestand hat oder einen Prozess ausführt, der nicht unterbrochen werden kann. Weitere Informationen finden Sie unter Tutorial: Entwickeln einer Greengrass-Komponente, die Komponentenaktualisierungen verzögert. Wenn Sie eine Bereitstellung erstellen, können Sie sie für die Bereitstellung konfigurieren, ohne Komponenten zu benachrichtigen.

Jedes Zielobjekt oder jede Objektgruppe kann jeweils eine Bereitstellung haben. Das bedeutet, dass beim Erstellen einer Bereitstellung für ein Ziel die vorherige Revision der Bereitstellung dieses Ziels AWS IoT Greengrass nicht mehr bereitstellt.

Optionen für die Bereitstellung

Bereitstellungen bieten mehrere Optionen, mit denen Sie steuern können, welche Geräte ein Update erhalten und wie das Update bereitgestellt wird. Wenn Sie eine Bereitstellung erstellen, können Sie die folgenden Optionen konfigurieren:

- AWS IoT Greengrass -Komponenten

  Definieren Sie die Komponenten, die auf den Zielgeräten installiert und ausgeführt werden sollen. -AWS IoT GreengrassKomponenten sind Softwaremodule, die Sie auf Greengrass-Core-Geräten bereitstellen und ausführen. Geräte erhalten Komponenten nur, wenn die Komponente die Plattform des Geräts unterstützt. Auf diese Weise können Sie auf Gerätegruppen bereitstellen, auch wenn die Zielgeräte auf mehreren Plattformen ausgeführt werden. Wenn eine Komponente die Plattform des Geräts nicht unterstützt, wird die Komponente nicht auf dem Gerät bereitgestellt.

  Sie können benutzerdefinierte Komponenten und AWSvon bereitgestellte Komponenten auf Ihren Geräten bereitstellen. Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass identifiziert alle Komponentenabhängigkeiten und stellt sie ebenfalls bereit. Weitere Informationen finden Sie unter Entwickeln von AWS IoT Greengrass Komponenten und AWSVon bereitgestellte Komponenten.

  Sie definieren die Versions- und Konfigurationsaktualisierung, die für jede Komponente bereitgestellt werden soll. Das Konfigurationsupdate gibt an, wie die vorhandene Konfiguration der Komponente auf dem Core-Gerät oder die Standardkonfiguration der Komponente geändert werden soll, wenn die Komponente auf dem Core-Gerät nicht vorhanden ist. Sie können angeben, welche Konfigurationswerte auf Standardwerte zurückgesetzt werden sollen, und die neuen Konfigurationswerte, die auf dem Core-Gerät zusammengeführt werden sollen. Wenn ein Core-Gerät Bereitstellungen für verschiedene Ziele empfängt und jede Bereitstellung kompatible Komponentenversionen angibt, wendet das Core-Gerät Konfigurationsaktualisierungen der Reihe nach an, basierend auf dem Zeitstempel der Erstellung der Bereitstellung. Weitere Informationen finden Sie unter Komponentenkonfigurationen aktualisieren.

  > **Important**
  >
  > Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass installiert die neuesten unterstützten Versionen aller Abhängigkeiten dieser Komponente. Aus diesem Grund werden neue Patch-Versionen von von bereitgestellten öffentlichen Komponenten möglicherweise automatisch auf Ihren AWS-Core-Geräten bereitgestellt, wenn Sie einer Objektgruppe neue Geräte hinzufügen oder die Bereitstellung aktualisieren, die auf diese Geräte abzielt. Einige automatische Updates, wie z. B. ein Kern-Update, können dazu führen, dass Ihre Geräte unerwartet neu gestartet werden.
  > Um unbeabsichtigte Updates für eine Komponente zu verhindern, die auf Ihrem Gerät ausgeführt wird, empfehlen wir Ihnen, beim Erstellen einer Bereitstellung direkt Ihre bevorzugte Version dieser Komponente anzugeben. Weitere Informationen zum

Aktualisierungsverhalten für AWS IoT Greengrass Core-Software finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

- Bereitstellungsrichtlinien

  Definieren Sie, wann eine Konfiguration sicher bereitgestellt werden kann und was zu tun ist, wenn die Bereitstellung fehlschlägt. Sie können angeben, ob auf die Meldung von Komponenten gewartet werden soll, die sie aktualisieren können. Sie können auch angeben, ob Geräte auf ihre vorherige Konfiguration zurückgesetzt werden sollen, wenn sie eine fehlgeschlagene Bereitstellung anwenden.

- Stoppen der Konfiguration

  Definieren Sie, wann und wie eine Bereitstellung angehalten werden soll. Die Bereitstellung wird beendet und schlägt fehl, wenn die von Ihnen definierten Kriterien erfüllt sind. Sie können beispielsweise eine Bereitstellung so konfigurieren, dass sie angehalten wird, wenn ein Prozentsatz der Geräte diese Bereitstellung nicht anwenden kann, nachdem sie von einer Mindestanzahl von Geräten empfangen wurde.

- Rollout-Konfiguration

  Definieren Sie die Geschwindigkeit, mit der eine Bereitstellung auf den Zielgeräten bereitgestellt wird. Sie können eine exponentielle Ratenerhöhung mit minimalen und maximalen Ratengrenzen konfigurieren.

- Timeout-Konfiguration

  Definieren Sie die maximale Zeit, die jedem Gerät zur Verfügung steht, um eine Bereitstellung anzuwenden. Wenn ein Gerät die von Ihnen angegebene Dauer überschreitet, kann das Gerät die Bereitstellung nicht anwenden.

Important

Benutzerdefinierte Komponenten können Artefakte in S3-Buckets definieren. Wenn die AWS IoT Greengrass -Core-Software eine Komponente bereitstellt, lädt sie die Artefakte der Komponente von herunterAWS Cloud. Core-Geräterollen erlauben standardmäßig keinen Zugriff auf S3-Buckets. Um benutzerdefinierte Komponenten bereitzustellen, die Artefakte in einem S3-Bucket definieren, muss die Rolle des Core-Geräts Berechtigungen

zum Herunterladen von Artefakten aus diesem Bucket erteilen. Weitere Informationen finden Sie unter Erlauben Sie den Zugriff auf S3-Buckets für Komponentenartefakte.

Themen

Erstellen von Bereitstellungen

Sie können eine Bereitstellung erstellen, die auf ein Objekt oder eine Objektgruppe abzielt.

Wenn Sie eine Bereitstellung erstellen, konfigurieren Sie die bereitzustellenden Softwarekomponenten und wie der Bereitstellungsauftrag auf Zielgeräten bereitgestellt wird. Sie können die Bereitstellung in der JSON-Datei definieren, die Sie dem bereitstellenAWS CLI.

Das Bereitstellungsziel bestimmt die Geräte, auf denen Sie Ihre Komponenten ausführen möchten. Um auf einem Core-Gerät bereitzustellen, geben Sie ein Objekt an. Um auf mehreren Core-Geräten bereitzustellen, geben Sie eine Objektgruppe an, die diese Geräte enthält. Weitere Informationen zum Konfigurieren von Objektgruppen finden Sie unter Statische Objektgruppen und Dynamische Objektgruppen im AWS IoT -Entwicklerhandbuch.

Führen Sie die Schritte in diesem Abschnitt aus, um eine Bereitstellung für ein Ziel zu erstellen. Weitere Informationen zum Aktualisieren der Softwarekomponenten auf einem Ziel mit einer Bereitstellung finden Sie unter Überarbeiten von Bereitstellungen.

⚠ Warning

Die -CreateDeploymentOperation kann Komponenten von -Core-Geräten deinstallieren. Wenn eine Komponente in der vorherigen Bereitstellung vorhanden ist und nicht in der neuen Bereitstellung, deinstalliert das Core-Gerät diese Komponente. Um die Deinstallation von Komponenten zu vermeiden, verwenden Sie zunächst die -ListDeploymentsOperation, um zu überprüfen, ob das Ziel für die Bereitstellung bereits über eine vorhandene Bereitstellung

verfügt. Verwenden Sie dann die -GetDeploymentOperation, um mit dieser vorhandenen Bereitstellung zu beginnen, wenn Sie eine neue Bereitstellung erstellen.

So erstellen Sie eine Bereitstellung (AWS CLI)

1. Erstellen Sie eine Datei mit dem Namen `deployment.json`und kopieren Sie dann das folgende JSON-Objekt in die Datei . Ersetzen Sie *`targetArn`* durch den ARN des AWS IoT Objekts oder der Objektgruppe, das/die für die Bereitstellung bestimmt werden soll. Objekt- und Objektgruppen-ARNs haben das folgende Format:

   - Objekt: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`thingName`*
   - Objektgruppe: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thinggroup/`*`thingGroupName`*

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

2. Überprüfen Sie, ob das Bereitstellungsziel über eine vorhandene Bereitstellung verfügt, die Sie überarbeiten möchten. Gehen Sie wie folgt vor:

   a. Führen Sie den folgenden Befehl aus, um die Bereitstellungen für das Bereitstellungsziel aufzulisten. Ersetzen Sie *`targetArn`* durch den ARN des AWS IoT Zielobjekts oder der Objektgruppe.

      ```
      aws greengrassv2 list-deployments --target-arn targetArn
      ```

      Die Antwort enthält eine Liste mit der neuesten Bereitstellung für das Ziel. Wenn die Antwort leer ist, verfügt das Ziel über keine vorhandene Bereitstellung, und Sie können mit fortfahrenStep 3. Andernfalls kopieren Sie die `deploymentId` aus der Antwort, um sie im nächsten Schritt zu verwenden.

      > **Note**
      >
      > Sie können auch eine andere Bereitstellung als die neueste Revision für das Ziel überarbeiten. Geben Sie das `--history-filter ALL` Argument an, um alle

Bereitstellungen für das Ziel aufzulisten. Kopieren Sie dann die ID der Bereitstellung, die Sie überarbeiten möchten.

b. Führen Sie den folgenden Befehl aus, um die Details der Bereitstellung abzurufen. Zu diesen Details gehören Metadaten, Komponenten und Auftragskonfiguration. Ersetzen Sie *deploymentId* durch die ID aus dem vorherigen Schritt.

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

Die Antwort enthält die Details der Bereitstellung.

c. Kopieren Sie eines der folgenden Schlüssel-Wert-Paare aus der Antwort des vorherigen Befehls in `deployment.json`. Sie können diese Werte für die neue Bereitstellung ändern.

- `deploymentName` – Der Name der Bereitstellung.
- `components` – Die Komponenten der Bereitstellung. Um eine Komponente zu deinstallieren, entfernen Sie sie aus diesem Objekt.
- `deploymentPolicies` – Die Richtlinien der Bereitstellung.
- `iotJobConfiguration` – Die Auftragskonfiguration der Bereitstellung.
- `tags` – Die Tags der Bereitstellung.

3. (Optional) Definieren Sie einen Namen für die Bereitstellung. Ersetzen Sie *deploymentName* durch den Namen der Bereitstellung.

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName"
}
```

4. Fügen Sie jede Komponente hinzu, um die Zielgeräte bereitzustellen. Fügen Sie dazu dem `components` Objekt Schlüssel-Wert-Paare hinzu, wobei der Schlüssel der Komponentenname und der Wert ein Objekt ist, das die Details für diese Komponente enthält. Geben Sie die folgenden Details für jede Komponente an, die Sie hinzufügen:

- `version` – Die Komponentenversion, die bereitgestellt werden soll.
- `configurationUpdate` – Das Konfigurationsupdate, das bereitgestellt werden soll. Das Update ist ein Patch-Vorgang, der die vorhandene Konfiguration der Komponente auf jedem Zielgerät oder die Standardkonfiguration der Komponente ändert, wenn sie nicht auf dem Zielgerät vorhanden ist. Sie können die folgenden Konfigurationsaktualisierungen angeben:

- Updates zurücksetzen (`reset`) – (Optional) Eine Liste von JSON-Zeigern, die die Konfigurationswerte definieren, die auf dem Zielgerät auf ihre Standardwerte zurückgesetzt werden sollen. In der AWS IoT Greengrass-Core-Software werden Reset-Updates vor Merge-Updates angewendet. Weitere Informationen finden Sie unter Updates zurücksetzen.
- Aktualisierungen zusammenführen (`merge`) – (Optional) Ein JSON-Dokument, das die Konfigurationswerte definiert, die auf dem Zielgerät zusammengeführt werden sollen. Sie müssen das JSON-Dokument als Zeichenfolge serialisieren. Weitere Informationen finden Sie unter Updates zusammenführen.

- `runWith` – (Optional) Die Systemprozessoptionen, die die AWS IoT Greengrass Core-Software verwendet, um die Prozesse dieser Komponente auf dem Core-Gerät auszuführen. Wenn Sie einen Parameter im -`runWith`Objekt weglassen, verwendet die AWS IoT Greengrass Core-Software die Standardwerte, die Sie für die Greengrass-Kernkomponente konfigurieren.

  Sie können eine der folgenden Optionen angeben:

  - `posixUser` – Der POSIX-Systembenutzer und optional die Gruppe, die zum Ausführen dieser Komponente auf Linux-Core-Geräten verwendet werden sollen. Der Benutzer und die Gruppe, falls angegeben, müssen auf jedem Linux-Core-Gerät vorhanden sein. Geben Sie den Benutzer und die Gruppe durch einen Doppelpunkt (`:`) getrennt im folgenden Format an: `user:group`. Die Gruppe ist optional. Wenn Sie keine Gruppe angeben, verwendet die AWS IoT Greengrass -Core-Software die primäre Gruppe für den Benutzer. Weitere Informationen finden Sie unter Konfigurieren Sie den Benutzer, der die Komponenten ausführt.
  - `windowsUser` – Der Windows-Benutzer, der zum Ausführen dieser Komponente auf Windows-Core-Geräten verwendet werden soll. Der Benutzer muss auf jedem Windows-Core-Gerät vorhanden sein und sein Name und sein Passwort müssen in der Credentials Manager-Instance des LocalSystem Kontos gespeichert werden. Weitere Informationen finden Sie unter Konfigurieren Sie den Benutzer, der die Komponenten ausführt.

    Diese Funktion ist für v2.5.0 und höher der Greengrass-Kernkomponente verfügbar.
  - `systemResourceLimits` – Die Systemressourcenlimits, die auf die Prozesse dieser Komponente angewendet werden sollen. Sie können Systemressourcenlimits auf generische und nicht containerisierte Lambda-Komponenten anwenden. Weitere Informationen finden Sie unter Konfigurieren Sie die Systemressourcenlimits für Komponenten.

Sie können eine der folgenden Optionen angeben:

- `cpus` – Die maximale CPU-Zeit, die die Prozesse dieser Komponente auf dem Core-Gerät verwenden können. Die gesamte CPU-Zeit eines Core-Geräts entspricht der Anzahl der CPU-Kerne des Geräts. Auf einem Core-Gerät mit 4 CPU-Kernen können Sie diesen Wert beispielsweise auf setzen, 2 um die Prozesse dieser Komponente auf eine Auslastung von 50 Prozent jedes CPU-Kerns zu beschränken. Auf einem Gerät mit 1 CPU-Kern können Sie diesen Wert auf setzen, `0.25` um die Prozesse dieser Komponente auf eine CPU-Auslastung von 25 Prozent zu beschränken. Wenn Sie diesen Wert auf eine Zahl festlegen, die größer als die Anzahl der CPU-Kerne ist, schränkt die AWS IoT Greengrass Core-Software die CPU-Auslastung der Komponente nicht ein.
- `memory` – Die maximale Menge an RAM (in Kilobyte), die die Prozesse dieser Komponente auf dem Core-Gerät verwenden können.

Diese Funktion ist für v2.4.0 und höher der Greengrass-Kernkomponente verfügbar. unterstützt diese Funktion derzeit AWS IoT Greengrass nicht auf Windows-Kerngeräten.

Example Beispiel für ein grundlegendes Konfigurationsupdate

Das folgende `components` Beispielobjekt gibt an, eine Komponente, `com.example.PythonRuntime`, bereitzustellen, die einen Konfigurationsparameter namens erwartet`pythonVersion`.

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
    "com.example.PythonRuntime": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"pythonVersion\":\"3.7\"}"
      }
    }
  }
}
```

Example Beispiel für eine Konfigurationsaktualisierung mit Reset- und Zusammenführungsaktualisierungen

Betrachten Sie ein Beispiel für eine Komponente des industriellen Dashboards, `com.example.IndustrialDashboard`, die die folgende Standardkonfiguration hat.

```
{
  "name": null,
  "mode": "REQUEST",
  "network": {
    "useHttps": true,
    "port": {
      "http": 80,
      "https": 443
    },
  },
  "tags": []
}
```

Das folgende Konfigurationsupdate gibt die folgenden Anweisungen an:

1. Setzen Sie die HTTPS-Einstellung auf ihren Standardwert (`true`) zurück.
2. Setzen Sie die Liste der industriellen Tags auf eine leere Liste zurück.
3. Führen Sie eine Liste von industriellen Tags zusammen, die Temperatur- und Druckdatenströme für zwei Boiler identifizieren.

```
{
  "reset": [
    "/network/useHttps",
    "/tags"
  ],
  "merge": {
    "tags": [
      "/boiler/1/temperature",
      "/boiler/1/pressure",
      "/boiler/2/temperature",
      "/boiler/2/pressure"
    ]
  }
```

```
}
```

Das folgende `components` Beispielobjekt gibt an, diese Komponente und Konfiguration des industriellen Dashboards bereitzustellen.

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "reset": [
          "/network/useHttps",
          "/tags"
        ],
        "merge": "{\"tags\":[\"/boiler/1/temperature\",\"/boiler/1/pressure\",\"/
boiler/2/temperature\",\"/boiler/2/pressure\"]}"
      }
    }
  }
}
```

5. (Optional) Definieren Sie Bereitstellungsrichtlinien für die Bereitstellung. Sie können konfigurieren, wann -Core-Geräte eine Bereitstellung sicher anwenden können oder was zu tun ist, wenn ein -Core-Gerät die Bereitstellung nicht anwenden kann. Fügen Sie dazu ein `deploymentPolicies` Objekt zu hinzu `deployment.json`und führen Sie dann einen der folgenden Schritte aus:

   1. (Optional) Geben Sie die Richtlinie zur Komponentenaktualisierung an (`componentUpdatePolicy`). Diese Richtlinie definiert, ob die Bereitstellung es Komponenten ermöglicht, ein Update aufschieben, bis sie zum Aktualisieren bereit sind. Komponenten müssen beispielsweise möglicherweise Ressourcen bereinigen oder kritische Aktionen abschließen, bevor sie neu gestartet werden können, um ein Update anzuwenden. Diese Richtlinie definiert auch die Zeit, die Komponenten haben, um auf eine Aktualisierungsbenachrichtigung zu reagieren.

      Diese Richtlinie ist ein Objekt mit den folgenden Parametern:

- `action` – (Optional) Gibt an, ob Komponenten benachrichtigt werden sollen und darauf gewartet werden soll, dass sie gemeldet werden, wenn sie für die Aktualisierung bereit sind. Wählen Sie aus den folgenden Optionen aus:
  - `NOTIFY_COMPONENTS` – Die Bereitstellung benachrichtigt jede Komponente, bevor diese Komponente gestoppt und aktualisiert wird. Komponenten können die SubscribeToComponentUpdates IPC-Operation verwenden, um diese Benachrichtigungen zu erhalten.
  - `SKIP_NOTIFY_COMPONENTS` – Die Bereitstellung benachrichtigt die Komponenten nicht und wartet nicht darauf, dass sie sicher aktualisiert werden können.

  Standardeinstellung: `NOTIFY_COMPONENTS`.
- `timeoutInSeconds` Die Zeit in Sekunden, die jede Komponente benötigt, um mit der DeferComponentUpdate IPC-Operation auf eine Aktualisierungsbenachrichtigung zu reagieren. Wenn die Komponente nicht innerhalb dieser Zeit reagiert, wird die Bereitstellung auf dem Core-Gerät fortgesetzt.

  Der Standardwert ist 60 Sekunden.

2. (Optional) Geben Sie die Richtlinie zur Konfigurationsvalidierung an (`configurationValidationPolicy`). Diese Richtlinie definiert, wie lange jede Komponente hat, um eine Konfigurationsaktualisierung aus einer Bereitstellung zu validieren. Komponenten können die SubscribeToValidateConfigurationUpdates IPC-Operation verwenden, um Benachrichtigungen für ihre eigenen Konfigurationsaktualisierungen zu abonnieren. Anschließend können Komponenten die SendConfigurationValidityReport IPC-Operation verwenden, um der AWS IoT Greengrass Core-Software mitzuteilen, ob das Konfigurationsupdate gültig ist. Wenn das Konfigurationsupdate nicht gültig ist, schlägt die Bereitstellung fehl.

   Diese Richtlinie ist ein Objekt mit dem folgenden Parameter:

   - `timeoutInSeconds` (Optional) Die Zeit in Sekunden, die jede Komponente zum Validieren einer Konfigurationsaktualisierung benötigt. Wenn die Komponente nicht innerhalb dieser Zeit reagiert, wird die Bereitstellung auf dem Core-Gerät fortgesetzt.

     Der Standardwert ist 30 Sekunden.

3. (Optional) Geben Sie die Richtlinie zur Fehlerbehandlung an (`failureHandlingPolicy`). Diese Richtlinie ist eine Zeichenfolge, die definiert, ob Geräte zurückgesetzt werden sollen, wenn die Bereitstellung fehlschlägt. Wählen Sie aus den folgenden Optionen aus:

- `ROLLBACK` – Wenn die Bereitstellung auf einem Core-Gerät fehlschlägt, setzt die AWS IoT Greengrass Core-Software dieses Core-Gerät auf seine vorherige Konfiguration zurück.
- `DO_NOTHING` – Wenn die Bereitstellung auf einem Core-Gerät fehlschlägt, behält die AWS IoT Greengrass Core-Software die neue Konfiguration bei. Dies kann zu fehlerhaften Komponenten führen, wenn die neue Konfiguration nicht gültig ist.

Standardeinstellung: `ROLLBACK`.

Ihre Bereitstellung in `deployment.json` könnte dem folgenden Beispiel ähneln:

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "reset": [
          "/network/useHttps",
          "/tags"
        ],
        "merge": "{\"tags\":[\"/boiler/1/temperature\",\"/boiler/1/pressure\",\"/
boiler/2/temperature\",\"/boiler/2/pressure\"]}"
      }
    }
  },
  "deploymentPolicies": {
    "componentUpdatePolicy": {
      "action": "NOTIFY_COMPONENTS",
      "timeoutInSeconds": 30
    },
    "configurationValidationPolicy": {
      "timeoutInSeconds": 60
    },
    "failureHandlingPolicy": "ROLLBACK"
  }
}
```

6. (Optional) Definieren Sie, wie die Bereitstellung angehalten wird, ein Rollout durchführt oder eine Zeitüberschreitung auftritt. AWS IoT Greengrass verwendet AWS IoT Core Aufträge, um Bereitstellungen an Core-Geräte zu senden, sodass diese Optionen mit den

Konfigurationsoptionen für AWS IoT Core Aufträge identisch sind. Weitere Informationen finden Sie unter Auftrags-Rollout und Abbruchkonfiguration im AWS IoT Entwicklerhandbuch für .

Um die Auftragsoptionen zu definieren, fügen Sie ein `iotJobConfiguration` Objekt zu hinzu`deployment.json`. Definieren Sie dann die zu konfigurierenden Optionen.

Ihre Bereitstellung in `deployment.json` könnte dem folgenden Beispiel ähneln:

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "reset": [
          "/network/useHttps",
          "/tags"
        ],
        "merge": "{\"tags\":[\"/boiler/1/temperature\",\"/boiler/1/pressure\",\"/
boiler/2/temperature\",\"/boiler/2/pressure\"]}"
      }
    }
  },
  "deploymentPolicies": {
    "componentUpdatePolicy": {
      "action": "NOTIFY_COMPONENTS",
      "timeoutInSeconds": 30
    },
    "configurationValidationPolicy": {
      "timeoutInSeconds": 60
    },
    "failureHandlingPolicy": "ROLLBACK"
  },
  "iotJobConfiguration": {
    "abortConfig": {
      "criteriaList": [
        {
          "action": "CANCEL",
          "failureType": "ALL",
          "minNumberOfExecutedThings": 100,
          "thresholdPercentage": 5
        }
```

```
      ]
    },
    "jobExecutionsRolloutConfig": {
      "exponentialRate": {
        "baseRatePerMinute": 5,
        "incrementFactor": 2,
        "rateIncreaseCriteria": {
          "numberOfNotifiedThings": 10,
          "numberOfSucceededThings": 5
        }
      },
      "maximumPerMinute": 50
    },
    "timeoutConfig":  {
      "inProgressTimeoutInMinutes": 5
    }
  }
}
```

7. (Optional) Fügen Sie Tags (`tags`) für die Bereitstellung hinzu. Weitere Informationen finden Sie unter Markieren Ihrer AWS IoT Greengrass Version 2-Ressourcen mit Tags.
8. Führen Sie den folgenden Befehl aus, um die Bereitstellung aus zu erstellen`deployment.json`.

   ```
   aws greengrassv2 create-deployment --cli-input-json file://deployment.json
   ```

   Die Antwort enthält eine `deploymentId`, die diese Bereitstellung identifiziert. Sie können die Bereitstellungs-ID verwenden, um den Status der Bereitstellung zu überprüfen. Weitere Informationen finden Sie unter Prüfen des Bereitstellungsstatus.

Komponentenkonfigurationen aktualisieren

Komponentenkonfigurationen sind JSON-Objekte, die die Parameter für jede Komponente definieren. Das Rezept jeder Komponente definiert ihre Standardkonfiguration, die Sie ändern, wenn Sie Komponenten auf Kerngeräten bereitstellen.

Wenn Sie eine Bereitstellung erstellen, können Sie das Konfigurationsupdate angeben, das für jede Komponente angewendet werden soll. Bei Konfigurationsupdates handelt es sich um Patch-Operationen. Das bedeutet, dass das Update die Komponentenkonfiguration ändert, die auf dem Kerngerät vorhanden ist. Wenn das Kerngerät nicht über die Komponente verfügt, ändert das Konfigurationsupdate die Standardkonfiguration für diese Bereitstellung und wendet sie an.

Das Konfigurationsupdate definiert das Zurücksetzen von Updates und das Zusammenführen von Updates. Beim Zurücksetzen von Updates wird definiert, welche Konfigurationswerte auf ihre Standardwerte zurückgesetzt oder entfernt werden sollen. Merge-Updates definieren die neuen Konfigurationswerte, die für die Komponente festgelegt werden sollen. Wenn Sie ein Konfigurationsupdate bereitstellen, führt die AWS IoT Greengrass Core-Software das Reset-Update vor dem Merge-Update aus.

Komponenten können die von Ihnen bereitgestellten Konfigurationsupdates validieren. Die Komponente abonniert den Empfang einer Benachrichtigung, wenn eine Bereitstellung ihre Konfiguration ändert, und sie kann eine Konfiguration ablehnen, die sie nicht unterstützt. Weitere Informationen finden Sie unter Interagieren mit der Komponentenkonfiguration.

Themen

Updates zurücksetzen

Beim Zurücksetzen von Updates wird definiert, welche Konfigurationswerte auf dem Kerngerät auf ihre Standardwerte zurückgesetzt werden sollen. Wenn ein Konfigurationswert keinen Standardwert hat, entfernt das Reset-Update diesen Wert aus der Konfiguration der Komponente. Dies kann Ihnen helfen, eine Komponente zu reparieren, die aufgrund einer ungültigen Konfiguration kaputt geht.

Verwenden Sie eine Liste von JSON-Zeigern, um zu definieren, welche Konfigurationswerte zurückgesetzt werden sollen. JSON-Zeiger beginnen mit einem Schrägstrich. / Um einen Wert in einer verschachtelten Komponentenkonfiguration zu identifizieren, verwenden Sie Schrägstriche (/), um die Schlüssel für jede Ebene in der Konfiguration voneinander zu trennen. Weitere Informationen finden Sie in der JSON-Zeigerspezifikation.

> **Note**
>
> Sie können nur eine gesamte Liste auf ihre Standardwerte zurücksetzen. Sie können das Zurücksetzen von Aktualisierungen nicht verwenden, um ein einzelnes Element in einer Liste zurückzusetzen.

Um die gesamte Konfiguration einer Komponente auf ihre Standardwerte zurückzusetzen, geben Sie eine einzelne leere Zeichenfolge als Reset-Update an.

```
"reset": [""]
```

Updates zusammenführen

Merge-Updates definieren die Konfigurationswerte, die in die Komponentenkonfiguration auf dem Core eingefügt werden sollen. Das Merge-Update ist ein JSON-Objekt, das die AWS IoT Greengrass Core-Software zusammenführt, nachdem sie die Werte in den Pfaden zurückgesetzt hat, die Sie im Reset-Update angegeben haben. Wenn Sie die AWS SDKs AWS CLI oder verwenden, müssen Sie dieses JSON-Objekt als Zeichenfolge serialisieren.

Sie können ein Schlüssel-Wert-Paar zusammenführen, das in der Standardkonfiguration der Komponente nicht vorhanden ist. Sie können auch ein Schlüssel-Wert-Paar zusammenführen, das einen anderen Typ hat als der Wert mit demselben Schlüssel. Der neue Wert ersetzt den alten Wert. Das bedeutet, dass Sie die Struktur des Konfigurationsobjekts ändern können.

Sie können Nullwerte und leere Zeichenketten, Listen und Objekte zusammenführen.

> **Note**
>
> Sie können das Zusammenführen von Aktualisierungen nicht dazu verwenden, ein Element in eine Liste einzufügen oder an sie anzuhängen. Sie können eine gesamte Liste ersetzen oder ein Objekt definieren, bei dem jedes Element einen eindeutigen Schlüssel hat.
> AWS IoT Greengrass verwendet JSON für Konfigurationswerte. JSON spezifiziert einen Zahlentyp, unterscheidet aber nicht zwischen Ganzzahlen und Gleitkommazahlen. Aus diesem Grund können Konfigurationswerte in Fließkommazahlen umgewandelt werden. AWS IoT Greengrass Um sicherzustellen, dass Ihre Komponente den richtigen Datentyp verwendet, empfehlen wir, numerische Konfigurationswerte als Zeichenfolgen zu definieren. Lassen Sie Ihre Komponente sie dann als Ganzzahlen oder Gleitkommazahlen analysieren. Dadurch wird sichergestellt, dass Ihre Konfigurationswerte in der Konfiguration und auf Ihrem Kerngerät denselben Typ haben.

Verwenden Sie Rezeptvariablen bei Merge-Updates

Diese Funktion ist für Version 2.6.0 und höher der Greengrass Nucleus-Komponente verfügbar.

Wenn Sie die ComponentConfiguration Konfigurationsoption Interpolate von Greengrass Nucleus auf einstellen`true`, können Sie in Merge-Updates andere Rezeptvariablen als die *`component_dependency_name`*`:configuration:`*`json_pointer`* Rezeptvariable verwenden. Sie können die `{iot:thingName}` Rezeptvariable beispielsweise in einem Merge-Update verwenden, um den Ding-Namen des AWS IoT Kerngeräts in einen Komponentenkonfigurationswert aufzunehmen, z. B. in eine IPC-Autorisierungsrichtlinie (Interprocess Communication).

Beispiele

Das folgende Beispiel zeigt Konfigurationsupdates für eine Dashboard-Komponente mit der folgenden Standardkonfiguration. Diese Beispielkomponente zeigt Informationen über Industrieanlagen an.

```
{
  "name": null,
  "mode": "REQUEST",
  "network": {
    "useHttps": true,
    "port": {
      "http": 80,
      "https": 443
    },
  },
  "tags": []
}
```

Rezept für industrielle Armaturenbrettkomponenten

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IndustrialDashboard",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Displays information about industrial equipment.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "name": null,
      "mode": "REQUEST",
      "network": {
        "useHttps": true,
```

```
        "port": {
          "http": 80,
          "https": 443
        },
      },
      "tags": []
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/industrial_dashboard.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "run": "py -3 -u {artifacts:path}/industrial_dashboard.py"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.IndustrialDashboard
ComponentVersion: '1.0.0'
ComponentDescription: Displays information about industrial equipment.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    name: null
    mode: REQUEST
    network:
      useHttps: true
      port:
```

```
        http: 80
        https: 443
    tags: []
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      run: |
        python3 -u {artifacts:path}/industrial_dashboard.py
  - Platform:
      os: windows
    Lifecycle:
      run: |
        py -3 -u {artifacts:path}/industrial_dashboard.py
```

Example Beispiel 1: Update zusammenführen

Sie erstellen eine Einrichtung, die das folgende Konfigurationsupdate anwendet, das ein Merge-Update, aber kein Reset-Update spezifiziert. Dieses Konfigurationsupdate weist die Komponente an, das Dashboard auf dem HTTP-Port 8080 mit Daten von zwei Boilern anzuzeigen.

Console

Konfiguration zum Zusammenführen

```
{
  "name": "Factory 2A",
  "network": {
    "useHttps": false,
    "port": {
      "http": 8080
    }
  },
  "tags": [
    "/boiler/1/temperature",
    "/boiler/1/pressure",
    "/boiler/2/temperature",
    "/boiler/2/pressure"
  ]
}
```

AWS CLI

Der folgende Befehl erstellt eine Bereitstellung auf einem Core-Gerät.

```
aws greengrassv2 create-deployment --cli-input-json file://dashboard-deployment.json
```

Die `dashboard-deployment.json` Datei enthält das folgende JSON-Dokument.

```
{
  "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "deploymentName": "Deployment for MyGreengrassCore",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"name\":\"Factory 2A\",\"network\":{\"useHttps\":false,\"port
\":{\"http\":8080}},\"tags\":[\"/boiler/1/temperature\",\"/boiler/1/pressure\",\"/
boiler/2/temperature\",\"/boiler/2/pressure\"]}"
      }
    }
  }
}
```

Greengrass CLI

Der folgende Greengrass-CLI-Befehl erstellt eine lokale Bereitstellung auf einem Core-Gerät.

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.IndustrialDashboard=1.0.0" \
  --update-config dashboard-configuration.json
```

Die `dashboard-configuration.json` Datei enthält das folgende JSON-Dokument.

```
{
  "com.example.IndustrialDashboard": {
    "MERGE": {
      "name": "Factory 2A",
      "network": {
        "useHttps": false,
```

```
        "port": {
          "http": 8080
        }
      },
      "tags": [
        "/boiler/1/temperature",
        "/boiler/1/pressure",
        "/boiler/2/temperature",
        "/boiler/2/pressure"
      ]
    }
  }
}
```

Nach diesem Update hat die Dashboard-Komponente die folgende Konfiguration.

```
{
  "name": "Factory 2A",
  "mode": "REQUEST",
  "network": {
    "useHttps": false,
    "port": {
      "http": 8080,
      "https": 443
    }
  },
  "tags": [
    "/boiler/1/temperature",
    "/boiler/1/pressure",
    "/boiler/2/temperature",
    "/boiler/2/pressure"
  ]
}
```

Example Beispiel 2: Updates zurücksetzen und zusammenführen

Anschließend erstellen Sie eine Einrichtung, die das folgende Konfigurationsupdate anwendet, das ein Reset-Update und ein Merge-Update spezifiziert. Diese Updates spezifizieren, dass das Dashboard auf dem Standard-HTTPS-Port mit Daten von verschiedenen Kesseln angezeigt wird. Diese Updates ändern die Konfiguration, die sich aus den Konfigurationsupdates im vorherigen Beispiel ergibt.

Console

Pfade zurücksetzen

```
[
  "/network/useHttps",
  "/tags"
]
```

Konfiguration zum Zusammenführen

```
{
  "tags": [
    "/boiler/3/temperature",
    "/boiler/3/pressure",
    "/boiler/4/temperature",
    "/boiler/4/pressure"
  ]
}
```

AWS CLI

Der folgende Befehl erstellt eine Bereitstellung auf einem Core-Gerät.

```
aws greengrassv2 create-deployment --cli-input-json file://dashboard-
deployment2.json
```

Die `dashboard-deployment2.json` Datei enthält das folgende JSON-Dokument.

```
{
  "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "deploymentName": "Deployment for MyGreengrassCore",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "reset": [
          "/network/useHttps",
          "/tags"
        ],
```

```
        "merge": "{\"tags\":[\"/boiler/3/temperature\",\"/boiler/3/pressure\",\"/
boiler/4/temperature\",\"/boiler/4/pressure\"]}"
      }
    }
  }
}
```

Greengrass CLI

Der folgende Greengrass-CLI-Befehl erstellt eine lokale Bereitstellung auf einem Core-Gerät.

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.IndustrialDashboard=1.0.0" \
  --update-config dashboard-configuration2.json
```

Die `dashboard-configuration2.json` Datei enthält das folgende JSON-Dokument.

```
{
  "com.example.IndustrialDashboard": {
    "RESET": [
      "/network/useHttps",
      "/tags"
    ],
    "MERGE": {
      "tags": [
        "/boiler/3/temperature",
        "/boiler/3/pressure",
        "/boiler/4/temperature",
        "/boiler/4/pressure"
      ]
    }
  }
}
```

Nach diesem Update hat die Dashboard-Komponente die folgende Konfiguration.

```
{
  "name": "Factory 2A",
  "mode": "REQUEST",
```

```
  "network": {
    "useHttps": true,
    "port": {
      "http": 8080,
      "https": 443
    }
  },
  "tags": [
    "/boiler/3/temperature",
    "/boiler/3/pressure",
    "/boiler/4/temperature",
    "/boiler/4/pressure",
  ]
}
```

Erstellen von Unterbereitstellungen

> **Note**
>
> Die Unterbereitstellungsfunktion ist auf Greengrass-Kernen der Version 2.9.0 und höher verfügbar. Es ist nicht möglich, eine Konfiguration für eine Unterbereitstellung mit früheren Komponentenversionen des Greengrass-Kerns bereitzustellen.

Eine Unterbereitstellung ist eine Bereitstellung, die auf eine kleinere Teilmenge von Geräten innerhalb einer übergeordneten Bereitstellung abzielt. Sie können Unterbereitstellungen verwenden, um eine Konfiguration für eine kleinere Teilmenge von Geräten bereitzustellen. Sie können auch Unterbereitstellungen erstellen, um eine erfolglose übergeordnete Bereitstellung zu wiederholen, wenn ein oder mehrere Geräte in dieser übergeordneten Bereitstellung fehlschlagen. Mit dieser Funktion können Sie Geräte auswählen, die in dieser übergeordneten Bereitstellung fehlgeschlagen sind, und eine Unterbereitstellung erstellen, um Konfigurationen zu testen, bis die Unterbereitstellung erfolgreich ist. Sobald die Unterbereitstellung erfolgreich ist, können Sie diese Konfiguration erneut für die übergeordnete Bereitstellung bereitstellen.

Führen Sie die Schritte in diesem Abschnitt aus, um eine Unterbereitstellung zu erstellen und ihren Status zu überprüfen. Weitere Informationen zum Erstellen von Bereitstellungen finden Sie unter Erstellen von Bereitstellungen.

So erstellen Sie eine Unterbereitstellung (AWS CLI)

1. Führen Sie den folgenden Befehl aus, um die neuesten Bereitstellungen für eine Objektgruppe abzurufen. Ersetzen Sie den ARN im Befehl durch den ARN der abzufragenden Objektgruppe. Legen Sie den Wert `--history-filter` auf fest**`LATEST_ONLY`**, um die neueste Bereitstellung dieser Objektgruppe anzuzeigen.

   ```
   aws greengrassv2 list-deployments --target-arn arn:aws:iot:region:account-id:thinggroup/thingGroupName --history-filter LATEST_ONLY
   ```

2. Kopieren Sie die `deploymentId` aus der Antwort in den list-deployments Befehl , um sie im nächsten Schritt zu verwenden.
3. Führen Sie den folgenden Befehl aus, um den Status einer Bereitstellung abzurufen. Ersetzen Sie durch *`deploymentId`* die ID der abzufragenden Bereitstellung.

   ```
   aws greengrassv2 get-deployment --deployment-id deploymentId
   ```

4. Kopieren Sie die `iotJobId` aus der Antwort auf den get-deployment Befehl , um sie im folgenden Schritt zu verwenden.
5. Führen Sie den folgenden Befehl aus, um die Liste der Auftragsausführungen für den angegebenen Auftrag abzurufen. Ersetzen Sie *`jobID`* durch die `iotJobId` aus dem vorherigen Schritt. Ersetzen Sie den *`Status`* durch den Status, nach dem Sie filtern möchten. Sie können Ergebnisse mit den folgenden Status filtern:

   - `QUEUED`
   - `IN_PROGRESS`
   - `SUCCEEDED`
   - `FAILED`
   - `TIMED_OUT`
   - `REJECTED`
   - `REMOVED`
   - `CANCELED`

   ```
   aws iot list-job-executions-for-job --job-id jobID --status status
   ```

6. Erstellen Sie eine neue AWS IoT Objektgruppe oder verwenden Sie eine vorhandene Objektgruppe für Ihre Unterbereitstellung. Fügen Sie dann dieser Objektgruppe ein AWS IoT -Objekt hinzu. Sie verwenden Objektgruppen, um Flotten von Greengrass-Core-Geräten zu verwalten. Wenn Sie Softwarekomponenten auf Ihren Geräten bereitstellen, können Sie entweder einzelne Geräte oder Gerätegruppen anvisieren. Sie können ein Gerät zu einer Objektgruppe mit einer aktiven Greengrass-Bereitstellung hinzufügen. Nach dem Hinzufügen können Sie die Softwarekomponenten dieser Objektgruppe auf diesem Gerät bereitstellen.

   Gehen Sie wie folgt vor, um eine neue Objektgruppe zu erstellen und Ihre Geräte hinzuzufügen:

   a. Erstellen Sie eine -AWS IoTObjektgruppe. Ersetzen Sie *MyGreengrassCoreGroup* durch den Namen für die neue Objektgruppe. Sie können keinen Doppelpunkt (:) in einem Objektgruppennamen verwenden.

      > **Note**
      >
      > Wenn eine Objektgruppe für eine Unterbereitstellung mit einem verwendet wird`parentTargetArn`, kann sie nicht mit einer anderen übergeordneten Flotte wiederverwendet werden. Wenn eine Objektgruppe bereits zum Erstellen einer Unterbereitstellung für eine andere Flotte verwendet wurde, gibt die API einen Fehler zurück.

      ```
      aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
      ```

      Wenn die Anforderung erfolgreich ist, sieht die Antwort ähnlich wie im folgenden Beispiel aus:

      ```
      {
        "thingGroupName": "MyGreengrassCoreGroup",
        "thingGroupArn": "arn:aws:iot:us-
      west-2:123456789012:thinggroup/MyGreengrassCoreGroup",
        "thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"
      }
      ```

   b. Fügen Sie Ihrer Objektgruppe einen bereitgestellten Greengrass-Kern hinzu. Führen Sie den folgenden Befehl mit diesen Parametern aus:

- Ersetzen Sie durch *MyGreengrassCore* den Namen Ihres bereitgestellten Greengrass-Kerns.
- Ersetzen Sie *MyGreengrassCoreGroup* durch den Namen Ihrer Objektgruppe.

```
aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-
name MyGreengrassCoreGroup
```

Der Befehl hat keine Ausgabe, wenn die Anforderung erfolgreich ist.

7. Erstellen Sie eine Datei mit dem Namen `deployment.json`und kopieren Sie dann das folgende JSON-Objekt in die Datei . Ersetzen Sie *targetArn* durch den ARN der AWS IoT Objektgruppe, auf die die Unterbereitstellung ausgerichtet werden soll. Ein Unterbereitstellungsziel kann nur eine Objektgruppe sein. Objektgruppen-ARNs haben das folgende Format:

   - Objektgruppe – `arn:aws:iot:`*region*`:`*account-id*`:thinggroup/`*thingGroupName*

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

8. Führen Sie den folgenden Befehl erneut aus, um die Details der ursprünglichen Bereitstellung abzurufen. Zu diesen Details gehören Metadaten, Komponenten und Auftragskonfiguration. Ersetzen Sie *deploymentId* durch die ID von Step 1. Sie können diese Bereitstellungskonfiguration verwenden, um Ihre Unterbereitstellung zu konfigurieren und nach Bedarf Änderungen vorzunehmen.

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

Die Antwort enthält die Details der Bereitstellung. Kopieren Sie eines der folgenden Schlüssel-Wert-Paare aus der Antwort des get-deployment Befehls in `deployment.json`. Sie können diese Werte für die Unterbereitstellung ändern. Weitere Informationen zu den Details dieses Befehls finden Sie unter GetDeployment.

- `components` – Die Komponenten der Bereitstellung. Um eine Komponente zu deinstallieren, entfernen Sie sie aus diesem Objekt.
- `deploymentName` – Der Name der Bereitstellung.

- `deploymentPolicies` – Die Richtlinien der Bereitstellung.
- `iotJobConfiguration` – Die Auftragskonfiguration der Bereitstellung.
- `parentTargetArn` – Das Ziel der übergeordneten Bereitstellung.
- `tags` – Die Tags der Bereitstellung.

9. Führen Sie den folgenden Befehl aus, um die Unterbereitstellung aus zu erstellen`deployment.json`. Ersetzen Sie *`subdeploymentName`* durch einen Namen für die Unterbereitstellung.

```
aws greengrassv2 create-deployment --deployment-name subdeploymentName --cli-input-
json file://deployment.json
```

Die Antwort enthält eine `deploymentId`, die diese Unterbereitstellung identifiziert. Sie können die Bereitstellungs-ID verwenden, um den Status der Bereitstellung zu überprüfen. Weitere Informationen finden Sie unter Überprüfen des Bereitstellungsstatus.

10. Wenn die Unterbereitstellung erfolgreich ist, können Sie ihre Konfiguration verwenden, um den übergeordneten Bereitstellungsdienst zu überarbeiten. Kopieren Sie die `deployment.json`, die Sie im vorherigen Schritt verwendet haben. Ersetzen Sie die `targetArn` in der JSON-Datei durch den ARN der übergeordneten Bereitstellung und führen Sie den folgenden Befehl aus, um die übergeordnete Bereitstellung mit dieser neuen Konfiguration zu erstellen.

> **Note**
>
> Wenn Sie eine neue Bereitstellungsrevision der übergeordneten Flotte erstellen, werden alle Bereitstellungsrevisionen und Unterbereitstellungen für diese übergeordnete Bereitstellung ersetzt. Weitere Informationen finden Sie unter Bereitstellungen überarbeiten.

```
aws greengrassv2 create-deployment --cli-input-json file://deployment.json
```

Die Antwort enthält eine `deploymentId`, die diese Bereitstellung identifiziert. Sie können die Bereitstellungs-ID verwenden, um den Status der Bereitstellung zu überprüfen. Weitere Informationen finden Sie unter Prüfen des Bereitstellungsstatus.

Überarbeiten von Bereitstellungen

Jedes Zielobjekt oder jede Objektgruppe kann jeweils eine aktive Bereitstellung haben. Wenn Sie eine Bereitstellung für ein Ziel erstellen, das bereits über eine Bereitstellung verfügt, ersetzen die Softwarekomponenten in der neuen Bereitstellung die Komponenten der vorherigen Bereitstellung. Wenn die neue Bereitstellung keine Komponente definiert, die die vorherige Bereitstellung definiert hat, entfernt die AWS IoT Greengrass -Core-Software diese Komponente von den Ziel-Core-Geräten. Sie können eine vorhandene Bereitstellung überarbeiten, sodass Sie die Komponenten, die auf Core-Geräten ausgeführt werden, nicht von einer früheren Bereitstellung zu einem Ziel entfernen.

Um eine Bereitstellung zu überarbeiten, erstellen Sie eine Bereitstellung, die mit denselben Komponenten und Konfigurationen beginnt, die in einer früheren Bereitstellung vorhanden waren. Sie verwenden die -CreateDeploymentOperation, die der gleichen Operation entspricht, die Sie zum Erstellen von Bereitstellungen verwenden.

So überarbeiten Sie eine Bereitstellung (AWS CLI)

1. Führen Sie den folgenden Befehl aus, um die Bereitstellungen für das Bereitstellungsziel aufzulisten. Ersetzen Sie *`targetArn`* durch den ARN des AWS IoT Zielobjekts oder der Objektgruppe.

   ```
   aws greengrassv2 list-deployments --target-arn targetArn
   ```

   Die Antwort enthält eine Liste mit der neuesten Bereitstellung für das Ziel. Kopieren Sie die `deploymentId` aus der Antwort, um sie im nächsten Schritt zu verwenden.

   > **Note**
   >
   > Sie können auch eine andere Bereitstellung als die neueste Revision für das Ziel überarbeiten. Geben Sie das `--history-filter ALL` Argument an, um alle Bereitstellungen für das Ziel aufzulisten. Kopieren Sie dann die ID der Bereitstellung, die Sie überarbeiten möchten.

2. Führen Sie den folgenden Befehl aus, um die Details der Bereitstellung abzurufen. Zu diesen Details gehören Metadaten, Komponenten und Auftragskonfiguration. Ersetzen Sie *`deploymentId`* durch die ID aus dem vorherigen Schritt.

   ```
   aws greengrassv2 get-deployment --deployment-id deploymentId
   ```

Die Antwort enthält die Details der Bereitstellung.

3. Erstellen Sie eine Datei namens `deployment.json` und kopieren Sie die Antwort des vorherigen Befehls in die Datei.
4. Entfernen Sie die folgenden Schlüssel-Wert-Paare aus dem JSON-Objekt in `deployment.json`:

   - `deploymentId`
   - `revisionId`
   - `iotJobId`
   - `iotJobArn`
   - `creationTimestamp`
   - `isLatestForTarget`
   - `deploymentStatus`

   Der CreateDeployment Vorgang erwartet eine Nutzlast mit der folgenden Struktur.

   ```
   {
     "targetArn": "String",
     "components": Map of components,
     "deploymentPolicies": DeploymentPolicies,
     "iotJobConfiguration": DeploymentIoTJobConfiguration,
     "tags": Map of tags
   }
   ```

5. Führen Sie in `deployment.json` eine der folgenden Aufgaben durch:

   - Ändern Sie den Namen der Bereitstellung (`deploymentName`).
   - Ändern Sie die Komponenten der Bereitstellung (`components`).
   - Ändern Sie die Richtlinien der Bereitstellung (`deploymentPolicies`).
   - Ändern Sie die Auftragskonfiguration der Bereitstellung (`iotJobConfiguration`).
   - Ändern Sie die Tags der Bereitstellung (`tags`).

   Weitere Informationen zum Definieren dieser Bereitstellungsdetails finden Sie unter Erstellen von Bereitstellungen.

6. Führen Sie den folgenden Befehl aus, um die Bereitstellung aus zu erstellen`deployment.json`.

```
aws greengrassv2 create-deployment --cli-input-json file://deployment.json
```

Die Antwort enthält eine `deploymentId`, die diese Bereitstellung identifiziert. Sie können die Bereitstellungs-ID verwenden, um den Status der Bereitstellung zu überprüfen. Weitere Informationen finden Sie unter Prüfen des Bereitstellungsstatus.

Abbrechen von Bereitstellungen

Sie können eine aktive Bereitstellung abbrechen, um zu verhindern, dass die zugehörigen Softwarekomponenten aufAWS IoT Greengrass Kerngeräten installiert werden. Wenn Sie eine Bereitstellung stornieren, die auf eine Dinggruppe abzielt, erhalten Kerngeräte, die Sie der Gruppe hinzufügen, diese kontinuierliche Bereitstellung nicht. Wenn die Bereitstellung bereits auf einem Kerngerät ausgeführt wird, werden Sie die Komponenten auf diesem Gerät nicht ändern, wenn Sie die Bereitstellung abbrechen. Sie müssen eine neue Bereitstellung erstellen oder die Bereitstellung überarbeiten, um die Komponenten zu ändern, die auf den Kerngeräten ausgeführt werden, die die stornierte Bereitstellung erhalten haben.

Um eine Bereitstellung abzubrechen (AWS CLI)

1. Führen Sie den folgenden Befehl aus, um die ID der letzten Deployment-Revision für ein Ziel zu ermitteln. Die neueste Revision ist die einzige Bereitstellung, die für ein Ziel aktiv sein kann, da vorherige Bereitstellungen storniert werden, wenn Sie eine neue Revision erstellen. *Erbrechen `targetArn`* -Zielobjekt-AWS IoT oder der Objektgruppe.

   ```
   aws greengrassv2 list-deployments --target-arn targetArn
   ```

   Die Antwort enthält eine Liste mit dem letzten Deployment für das Ziel. Kopieren Sie die`deploymentId` aus der Antwort, um sie im nächsten Schritt zu verwenden.

2. Abbrechen der Bereitstellung mithilfe des folgenden Befehls. *Erbrechen Sie deploymentID* durch die ID aus dem vorherigen Schritt.

   ```
   aws greengrassv2 cancel-deployment --deployment-id deploymentId
   ```

   Wenn die Operation erfolgreich ist, ändert sich der Bereitstellungsstatus in`CANCELED`.

Prüfen des Bereitstellungsstatus

Sie können den Status einer Bereitstellung überprüfen, die Sie erstellt habenAWS IoT Greengrass. Sie können auch den Status derAWS IoT Jobs überprüfen, die die Bereitstellung auf jedem Kerngerät bereitstellen. Während ein Deployment aktiv ist, lautet der Status desAWS IoT Jobs`IN_PROGRESS`. Nachdem Sie eine neue Version eines Deployments erstellt haben, ändert sich der Status desAWS IoT Jobs der vorherigen Version in`CANCELLED`.

Themen

Prüfen des Bereitstellungsstatus

Sie können den Status einer Bereitstellung überprüfen, die Sie anhand ihres Ziels oder ihrer ID identifizieren.

Um den Bereitstellungsstatus nach Ziel zu überprüfen (AWS CLI)

- Führen Sie den folgenden Befehl aus, um den Status der letzten Bereitstellung für ein Ziel abzurufen. Ersetzen *`targetArn`* durch den Amazon-Ressourcennamen (ARN) des Dings oderAWS IoT der Dinggruppe, auf die die Bereitstellung abzielt.

  ```
  aws greengrassv2 list-deployments --target-arn targetArn
  ```

  Die Antwort enthält eine Liste mit dem letzten Deployment für das Ziel. Dieses Bereitstellungsobjekt enthält den Status der Bereitstellung.

Um den Bereitstellungsstatus anhand der ID (AWS CLI) zu überprüfen

- Führen Sie den folgenden Befehl aus, um den Status einer Bereitstellung abzurufen. Ersetzen Sie *`deploymentId`* durch die ID des abzufragenden Deployments.

  ```
  aws greengrassv2 get-deployment --deployment-id deploymentId
  ```

  Die Antwort enthält den Status der Bereitstellung.

Überprüfen Sie den Status der Gerätebereitstellung

Sie können den Status eines Bereitstellungsauftrag überprüfen, die für ein einzelnes Kerngerät gilt. Sie können auch den Status eines Bereitstellungsauftrags für eine Dinggruppenbereitstellung überprüfen.

Um den Status von Bereitstellungsaufträgen für ein Kerngerät zu überprüfen (AWS CLI)

- Führen Sie den folgenden Befehl aus, um den Status aller Bereitstellungsaufträge für ein Kerngerät abzurufen. Ersetzen Sie *coreDeviceName*durch den Namen des abzufragenden Kerngeräts.

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

  Die Antwort enthält die Liste der Bereitstellungsaufträge für das Kerngerät. Sie können den Job für einen Einsatz anhand des Jobs`deploymentId` oder identifizieren`targetArn`. Jeder Bereitstellungsauftrag enthält den Status des Jobs auf dem Kerngerät.

Um den Bereitstellungsstatus für eine Dinggruppe zu überprüfen (AWS CLI)

1. Führen Sie den folgenden Befehl aus, um die ID einer vorhandenen Bereitstellung abzurufen. Ersetzen *targetArn* durch ARN der ARN der ARN der Zielgruppe.

   ```
   aws greengrassv2 list-deployments --target-arn targetArn
   ```

   Die Antwort enthält eine Liste mit dem letzten Deployment für das Ziel. Kopieren Sie die`deploymentId` aus der Antwort, um sie im nächsten Schritt zu verwenden.

   > **Note**
   >
   > Sie können auch ein anderes Deployment als das letzte Deployment für das Ziel auflisten. Geben Sie das`--history-filter ALL` Argument an, um alle Deployments für das Ziel aufzulisten. Kopieren Sie dann die ID der Bereitstellung, deren Status Sie überprüfen möchten.

2. Führen Sie den folgenden Befehl aus, um die Bereitstellungsdetails abzurufen. Ersetzen Sie *DeploymentID* durch die ID aus dem vorherigen Schritt.

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

Die Antwort enthält Informationen zur Bereitstellung. Kopieren Sie das`iotJobId` aus der Antwort, um es im folgenden Schritt zu verwenden.

3. Führen Sie den folgenden Befehl aus, um die Jobausführung eines Kerngeräts für die Bereitstellung zu beschreiben. Ersetzen Sie *iotJobId*und *coreDeviceThingName* durch die Job-ID aus dem vorherigen Schritt und das Kerngerät, für das Sie den Status überprüfen möchten.

```
aws iot describe-job-execution --job-id iotJobId --thing-name coreDeviceThingName
```

Die Antwort enthält den Status der Ausführung des Bereitstellungsauftrags des Kerngeräts sowie Details zum Status. Das`detailsMap` enthält die folgenden Informationen:

- `detailed-deployment-status`— Der Bereitstellungsstatus. Es gibt zwei der folgenden Werte:
  - `SUCCESSFUL`— Der Einsatz war erfolgreich.
  - `FAILED_NO_STATE_CHANGE`— Die Bereitstellung ist fehlgeschlagen, während sich das Kerngerät auf die Implementierung vorbereitet hat.
  - `FAILED_ROLLBACK_NOT_REQUESTED`— Die Bereitstellung ist fehlgeschlagen, und in der Bereitstellung wurde nicht angegeben, zu einer vorherigen funktionierenden Konfiguration zurückzukehren, sodass das Kerngerät möglicherweise nicht richtig funktioniert.
  - `FAILED_ROLLBACK_COMPLETE`— Die Bereitstellung ist fehlgeschlagen, und das Kerngerät wurde erfolgreich auf eine vorherige funktionierende Konfiguration zurückgesetzt.
  - `FAILED_UNABLE_TO_ROLLBACK`— Die Bereitstellung ist fehlgeschlagen, und das Kerngerät konnte nicht auf eine vorherige funktionierende Konfiguration zurückgesetzt werden, sodass das Kerngerät möglicherweise nicht richtig funktioniert.

  Wenn die Bereitstellung fehlgeschlagen ist, überprüfen Sie den`deployment-failure-cause` Wert und die Protokolldateien des Kerngeräts, um das Problem zu identifizieren. Weitere Informationen zum Zugriff auf die Protokolldateien des Kerngeräts finden Sie unterÜberwachen von AWS IoT Greengrass Protokollen.
- `deployment-failure-cause`— Eine Fehlermeldung, die zusätzliche Informationen darüber enthält, warum die Auftragsausführung fehlgeschlagen ist.

Die Antwort sieht in etwa so aus:


```
{
  "execution": {
    "jobId": "2cc2698a-5175-48bb-adf2-1dd345606ebd",
    "status": "FAILED",
    "statusDetails": {
      "detailsMap": {
        "deployment-failure-cause": "No local or cloud component version
 satisfies the requirements. Check whether the version constraints conflict and
 that the component exists in your AWS-Konto with a version that matches the
 version constraints. If the version constraints conflict, revise deployments
 to resolve the conflict. Component com.example.HelloWorld version constraints:
 LOCAL_DEPLOYMENT requires =1.0.0, thinggroup/MyGreengrassCoreGroup requires
 =1.0.1.",
        "detailed-deployment-status": "FAILED_NO_STATE_CHANGE"
      }
    },
    "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
    "queuedAt": "2022-02-15T14:45:53.098000-08:00",
    "startedAt": "2022-02-15T14:46:05.670000-08:00",
    "lastUpdatedAt": "2022-02-15T14:46:20.892000-08:00",
    "executionNumber": 1,
    "versionNumber": 3
  }
}
```

Protokollieren und Überwachen in AWS IoT Greengrass

Die Überwachung ist ein wichtiger Teil der Wahrung 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 Überwachungsziele?
- Welche Ressourcen möchten Sie überwachen?
- Wie oft werden diese Ressourcen überwacht?
- Welche Überwachungs-Tools möchten Sie verwenden?
- Wer soll die Überwachungsaufgaben ausführen?
- Wer soll benachrichtigt werden, wenn Fehler auftreten?

Themen

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 Überwachungstools verwenden, um Probleme zu überwachen AWS IoT Greengrass und zu melden:

- Amazon CloudWatch Logs – Überwachen, Speichern und Zugriff auf Ihre Protokolldateien von AWS CloudTrail oder anderen Quellen. Weitere Informationen finden Sie unter Überwachen von Protokolldateien im Amazon- CloudWatch Benutzerhandbuch.
- AWS CloudTrail Protokollüberwachung – Teilen Sie Protokolldateien zwischen Konten, überwachen Sie CloudTrail Protokolldateien in Echtzeit, indem Sie sie an - CloudWatch Protokolle senden, schreiben Sie Anwendungen zur Protokollverarbeitung in Java und überprüfen Sie, ob sich Ihre Protokolldateien nach der Bereitstellung durch nicht geändert haben CloudTrail. Weitere Informationen finden Sie unter Arbeiten mit CloudTrail Protokolldateien im AWS CloudTrail - Benutzerhandbuch.
- Greengrass-Systemzustandstelemetrie – Abonnieren Sie den Empfang von Telemetriedaten, die vom Greengrass-Kern gesendet werden. Weitere Informationen finden Sie unter the section called “Erfassen von Telemetriedaten zum Systemstatus”.
- Gerätezustandsbenachrichtigungen Erstellen Sie Ereignisse mit Amazon EventBridge , um Statusaktualisierungen zu Bereitstellungen und Komponenten zu erhalten. Weitere Informationen finden Sie unter Erhalten Sie Benachrichtigungen über den Bereitstellungs- und Komponentenstatus.
- Flottenstatusservice – Verwenden Sie die Flottenstatus-API-Operationen, um den Status von -Core-Geräten und deren Greengrass-Komponenten zu überprüfen. Sie können auch Flottenstatusinformationen in der AWS IoT Greengrass Konsole anzeigen. Weitere Informationen finden Sie unter Überprüfen Sie den Status des Greengrass Core-Geräts.

Überwachen von 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 Logs und in das lokale Dateisystem des Core-Geräts schreiben. Greengrass-Komponenten, die auf dem Core-Gerät ausgeführt werden, können auch Protokolle in CloudWatch Protokolle 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.

Standardmäßig schreibt die AWS IoT Greengrass Core-Software Protokolle nur in das lokale Dateisystem. Sie können Dateisystemprotokolle in Echtzeit anzeigen, sodass Sie Greengrass-Komponenten debuggen können, die Sie entwickeln und bereitstellen. Sie können ein Core-Gerät auch so konfigurieren, dass Protokolle in CloudWatch -Protokolle geschrieben werden, sodass Sie Probleme mit dem Core-Gerät beheben können, ohne auf das lokale Dateisystem zugreifen

zu müssen. Weitere Informationen finden Sie unter Aktivieren der Protokollierung in - CloudWatch Protokollen.

Themen

Zugriff auf Dateisystemprotokolle

Die AWS IoT Greengrass Core-Software speichert Protokolle im */greengrass/v2*/logs Ordner auf einem Core-Gerät, wobei der Pfad zum AWS IoT Greengrass Stammordner */greengrass/v2* ist. Der Ordner „Protokolle“ hat die folgende Struktur.

```
/greengrass/v2
### logs
    ### greengrass.log
    ### greengrass_2021_09_14_15_0.log
    ### ComponentName.log
    ### ComponentName_2021_09_14_15_0.log
    ### main.log
```

- `greengrass.log` – Die Protokolldatei der AWS IoT Greengrass -Core-Software. Verwenden Sie diese Protokolldatei, um Echtzeitinformationen zu Komponenten und Bereitstellungen anzuzeigen. Diese Protokolldatei enthält Protokolle für den Greengrass-Kern, der das Kernverzeichnis der AWS IoT Greengrass-Core-Software ist, sowie Plugin-Komponenten wie Log Manager und Secret Manager.
- *ComponentName*`.log` – Greengrass-Komponenten-Protokolldateien. Verwenden Sie Komponentenprotokolldateien, um Echtzeitinformationen zu einer Greengrass-Komponente anzuzeigen, die auf dem Core-Gerät ausgeführt wird. Generische Komponenten und Lambda-Komponenten schreiben die Standardausgabe (stdout) und den Standardfehler (stderr) in diese Protokolldateien.

- `main.log` – Die Protokolldatei für den `main` Service, der den Komponentenlebenszyklus verarbeitet. Diese Protokolldatei ist immer leer.

Weitere Informationen zu den Unterschieden zwischen Plugin-, generischen und Lambda-Komponenten finden Sie unter Komponententypen.

Die folgenden Hinweise gelten für die Verwendung von Dateisystemprotokollen:

- Berechtigungen für Root-Benutzer

  Sie müssen über Root-Berechtigungen zum Lesen von AWS IoT Greengrass-Protokollen im Dateisystem verfügen.

- Rotation der Protokolldatei

  Die AWS IoT Greengrass Core-Software rotiert Protokolldateien stündlich oder wenn sie eine Dateigrößenbeschränkung überschreiten. Rotierte Protokolldateien enthalten einen Zeitstempel in ihrem Dateinamen. Beispielsweise könnte eine rotierte AWS IoT Greengrass Core-Softwareprotokolldatei den Namen haben`greengrass_2021_09_14_15_0.log`. Die Standarddateigrößenbeschränkung beträgt 1 024 KB (1 MB). Sie können die Dateigrößenbeschränkung für die Greengrass-Kernkomponente konfigurieren.

- Löschen von Protokolldateien

  Die AWS IoT Greengrass Core-Software bereinigt frühere Protokolldateien, wenn die Größe von AWS IoT Greengrass Core-Softwareprotokolldateien oder Greengrass-Komponentenprotokolldateien, einschließlich rotierter Protokolldateien, ein Festplattenspeicherlimit überschreitet. Das standardmäßige Speicherplatzlimit für das AWS IoT Greengrass Core-Softwareprotokoll und jedes Komponentenprotokoll beträgt 10.240 KB (10 MB). Sie können das Speicherplatzlimit für das AWS IoT Greengrass Core-Softwareprotokoll für die Greengrass-Kernkomponente oder die Log Manager-Komponente konfigurieren. Sie können das Speicherplatzlimit für die Protokollfestplatten jeder Komponente für die Protokollmanagerkomponente konfigurieren.

So zeigen Sie die AWS IoT Greengrass-Core-Software-Protokolldatei an

- Führen Sie den folgenden Befehl aus, um die Protokolldatei in Echtzeit anzuzeigen. Ersetzen Sie durch */greengrass/v2* den Pfad zum AWS IoT Greengrass Stammordner.

Linux or Unix

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

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\logs\com.example.HelloWorld.log
```

Der `type` Befehl schreibt den Inhalt der Datei in das Terminal. Führen Sie diesen Befehl mehrmals aus, um Änderungen in der Datei zu beobachten.

PowerShell

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

So zeigen Sie die Protokolldatei für eine Komponente an

- Führen Sie den folgenden Befehl aus, um die Protokolldatei in Echtzeit anzuzeigen. Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Pfad zum AWS IoT Greengrass Stammordner und ersetzen Sie *com.example durchHelloWorld* den Namen der Komponente.

  Linux or Unix

  ```
  sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
  ```

  PowerShell

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

Sie können auch den `logs` Befehl der Greengrass-CLI verwenden, um Greengrass-Protokolle auf einem Core-Gerät zu analysieren. Um den `logs` Befehl zu verwenden, müssen Sie den Greengrass-Kern für die Ausgabe von Protokolldateien im JSON-Format konfigurieren. Weitere Informationen finden Sie unter Greengrass-Befehlszeilenschnittstelle und Protokolle.

CloudWatch Zugriffsprotokolle

Sie können die Log Manager-Komponente bereitstellen, um das Core-Gerät so zu konfigurieren, dass es in CloudWatch Protokolle schreibt. Weitere Informationen finden Sie unter Aktivieren der Protokollierung in - CloudWatch Protokollen. Anschließend können Sie Protokolle auf der Seite Protokolle der Amazon- CloudWatch Konsole oder mithilfe der CloudWatch Protokoll-API anzeigen.

Protokollgruppenname

```
/aws/greengrass/componentType/region/componentName
```

Der Name der Protokollgruppe verwendet die folgenden Variablen:

- `componentType` – Der Typ der Komponente, der einer der folgenden sein kann:
  - `GreengrassSystemComponent` – Diese Protokollgruppe enthält Protokolle für die Kern- und Plugin-Komponenten, die in derselben JVM wie der Greengrass-Kern ausgeführt werden. Die Komponente ist Teil des Greengrass-Kerns .
  - `UserComponent` – Diese Protokollgruppe enthält Protokolle für generische Komponenten, Lambda-Komponenten und andere Anwendungen auf dem Gerät. Die Komponente ist nicht Teil des Greengrass-Kerns.

  Weitere Informationen finden Sie unter Komponententypen.
- `region` – Die AWS Region, die das Core-Gerät verwendet.
- `componentName` – Der Name der Komponente. Für Systemprotokolle ist dieser Wert `System`.

Name des Protokollstreams

```
/date/thing/thingName
```

Der Name des Protokollstreams verwendet die folgenden Variablen:

- `date` – Das Datum des Protokolls, z. B. `2020/12/15`. Die Log-Manager-Komponente verwendet das `yyyy/MM/dd` Format .
- `thingName` – Der Name des Core-Geräts.

> **Note**
>
> Wenn ein Objektname einen Doppelpunkt (`:`) enthält, ersetzt der Protokollmanager den Doppelpunkt durch ein Pluszeichen (`+`).

Die folgenden Überlegungen gelten, wenn Sie die Log Manager-Komponente zum Schreiben in CloudWatch Protokolle verwenden:

- Protokollverzögerungen

> **Note**
>
> Wir empfehlen Ihnen, auf Log Manager Version 2.3.0 zu aktualisieren, um Protokollverzögerungen für rotierte und aktive Protokolldateien zu reduzieren. Wenn Sie auf Log Manager 2.3.0 aktualisieren, empfehlen wir Ihnen, auch auf Greengrass-Kern 2.9.1 zu aktualisieren.

  Die Log Manager-Komponente Version 2.2.8 (und früher) verarbeitet und lädt Protokolle nur aus rotierten Protokolldateien hoch. Standardmäßig rotiert die AWS IoT Greengrass Core-Software Protokolldateien stündlich oder nach 1 024 KB. Daher lädt die Protokollmanagerkomponente Protokolle erst hoch, nachdem die -AWS IoT GreengrassCore-Software oder eine Greengrass-Komponente Protokolle mit mehr als 1.024 KB geschrieben hat. Sie können eine niedrigere Größenbeschränkung für Protokolldateien konfigurieren, damit Protokolldateien häufiger rotiert werden. Dies führt dazu, dass die Protokollmanagerkomponente Protokolle häufiger in CloudWatch Protokolle hochlädt.

  Die Protokollmanager-Komponente Version 2.3.0 (und höher) verarbeitet und lädt alle Protokolle hoch. Wenn Sie ein neues Protokoll schreiben, verarbeitet und lädt Log Manager Version 2.3.0 (und höher) diese aktive Protokolldatei direkt hoch, anstatt darauf zu warten, dass sie rotiert wird. Das bedeutet, dass Sie das neue Protokoll in höchstens 5 Minuten anzeigen können.

  Die Log Manager-Komponente lädt regelmäßig neue Protokolle hoch. Standardmäßig lädt die Log Manager-Komponente alle 5 Minuten neue Protokolle hoch. Sie können ein niedrigeres Upload-Intervall konfigurieren, sodass die Protokollmanagerkomponente Protokolle häufiger in CloudWatch Protokolle hochlädt, indem Sie die konfigurieren`periodicUploadIntervalSec`. Weitere Informationen zur Konfiguration dieses regelmäßigen Intervalls finden Sie unter Konfiguration .

  Protokolle können nahezu in Echtzeit aus demselben Greengrass-Dateisystem hochgeladen werden. Wenn Sie Protokolle in Echtzeit beobachten müssen, sollten Sie die Verwendung von Dateisystemprotokollen in Betracht ziehen.

> **Note**
>
> Wenn Sie verschiedene Dateisysteme verwenden, um Protokolle zu schreiben, kehrt Log Manager zum Verhalten in den Versionen 2.2.8 und früher des Protokollmanagers zurück. Informationen zum Zugriff auf Dateisystemprotokolle finden Sie unter Zugriff auf Dateisystemprotokolle.

- Zeitverzerrung

  Die Log-Manager-Komponente verwendet den Standard-Signaturprozess von Signature Version 4, um API-Anforderungen an CloudWatch Logs zu erstellen. Wenn die Systemzeit auf einem Core-Gerät um mehr als 15 Minuten nicht synchron ist, lehnt CloudWatch Logs die Anforderungen ab. Weitere Informationen finden Sie unter Signaturprozess mit Signaturversion 4 in derAllgemeine AWS-Referenz .

Zugriff auf System-Serviceprotokolle

Wenn Sie die AWS IoT Greengrass -Core-Software als Systemservice konfigurieren, können Sie Systemserviceprotokolle anzeigen, um Probleme zu beheben, z. B. wenn die Software nicht gestartet werden kann.

So zeigen Sie System-Serviceprotokolle (CLI) an

1. Führen Sie den folgenden Befehl aus, um die Serviceprotokolle des AWS IoT Greengrass -Core-Softwaresystems anzuzeigen.

   Linux or Unix (systemd)

   ```
   sudo journalctl -u greengrass.service
   ```

   Windows Command Prompt (CMD)

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

   PowerShell

   ```
   gc C:\greengrass\v2\logs\greengrass.wrapper.log
   ```

2. Auf Windows-Geräten erstellt die AWS IoT Greengrass -Core-Software eine separate Protokolldatei für Systemdienstfehler. Führen Sie den folgenden Befehl aus, um die Systemservice-Fehlerprotokolle anzuzeigen.

   Windows Command Prompt (CMD)

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

   PowerShell

   ```
   gc C:\greengrass\v2\logs\greengrass.err.log
   ```

Auf Windows-Geräten können Sie auch die Event Viewer-Anwendung verwenden, um Systemserviceprotokolle anzuzeigen.

So zeigen Sie Windows-Serviceprotokolle an (Event Viewer)

1. Öffnen Sie die Event Viewer-Anwendung.
2. Wählen Sie Windows Logs aus, um es zu erweitern.
3. Wählen Sie Anwendung, um Anwendungsserviceprotokolle anzuzeigen.
4. Suchen und öffnen Sie Ereignisprotokolle, deren Quelle istgreengrass.

Aktivieren der Protokollierung in - CloudWatch Protokollen

Sie können die Log Manager-Komponente bereitstellen, um ein Core-Gerät so zu konfigurieren, dass Protokolle in CloudWatch Logs geschrieben werden. Sie können CloudWatch Protokolle für AWS IoT Greengrass -Core-Softwareprotokolle und CloudWatch Protokolle für bestimmte Greengrass-Komponenten aktivieren.

> **Note**
>
> Die Token-Austauschrolle des Greengrass-Core-Geräts muss dem Core-Gerät erlauben, in CloudWatch Protokolle zu schreiben, wie in der folgenden Beispiel-IAM-Richtlinie gezeigt. Wenn Sie die AWS IoT Greengrass Core-Software mit automatischer Ressourcenbereitstellung installiert haben, verfügt Ihr Core-Gerät über diese Berechtigungen.
>
> ```
> {
> ```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:logs:*:*:*"
    }
  ]
}
```

Um ein Core-Gerät für das Schreiben von AWS IoT Greengrass Core-Softwareprotokollen in CloudWatch Logs zu konfigurieren, erstellen Sie eine Bereitstellung, die ein Konfigurationsupdate angibt, das `true` für die `aws.greengrass.LogManager` Komponente `uploadToCloudWatch` auf festlegt. AWS IoT Greengrass Core-Softwareprotokolle enthalten Protokolle für den Greengrass-Kern und die Plugin-Komponenten .

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true"
    }
  }
}
```

Um ein Core-Gerät so zu konfigurieren, dass es die Protokolle einer Greengrass-Komponente in CloudWatch Protokolle schreibt, erstellen Sie eine Bereitstellung, die ein Konfigurationsupdate angibt, das die Komponente zur Liste der Komponentenprotokollierungskonfigurationen hinzufügt. Wenn Sie dieser Liste eine Komponente hinzufügen, schreibt die Protokollmanagerkomponente ihre Protokolle in CloudWatch Protokolle. Komponentenprotokolle enthalten Protokolle für generische Komponenten und Lambda-Komponenten.

```
{
  "logsUploaderConfiguration": {
    "componentLogsConfigurationMap": {
```

```
      "com.example.HelloWorld": {

      }
    }
  }
}
```

Wenn Sie die Log Manager-Komponente bereitstellen, können Sie auch Speicherplatzlimits konfigurieren und festlegen, ob das Core-Gerät Protokolldateien löscht, nachdem sie in CloudWatch Logs geschrieben wurden. Weitere Informationen finden Sie unter Konfigurieren der Protokollierung für AWS IoT Greengrass.

Konfigurieren der Protokollierung für AWS IoT Greengrass

Sie können die folgenden Optionen konfigurieren, um die Protokollierung für Greengrass-Core-Geräte anzupassen. Um diese Optionen zu konfigurieren, erstellen Sie eine Bereitstellung, die eine Konfigurationsaktualisierung für die Greengrass-Kern- oder Protokollmanagerkomponenten angibt.

- Schreiben von Protokollen in CloudWatch Protokolle

  Zur Remote-Fehlerbehebung bei -Core-Geräten können Sie -Core-Geräte so konfigurieren, dass -AWS IoT GreengrassCore-Software und Komponentenprotokolle in CloudWatch -Protokolle geschrieben werden. Stellen Sie dazu die Log Manager-Komponente bereit und konfigurieren Sie sie. Weitere Informationen finden Sie unter Aktivieren der Protokollierung in - CloudWatch Protokollen.
- Löschen hochgeladener Protokolldateien

  Um den Speicherplatz zu reduzieren, können Sie -Core-Geräte so konfigurieren, dass Protokolldateien gelöscht werden, nachdem die Protokolldateien in CloudWatch -Protokolle geschrieben wurden. Weitere Informationen finden Sie im `deleteLogFileAfterCloudUpload` Parameter der Protokollmanagerkomponente, den Sie für -AWS IoT GreengrassCore-Softwareprotokolle und Komponentenprotokolle angeben können.
- Speicherplatzlimits für Protokolle

  Um die Festplattenspeichernutzung zu begrenzen, können Sie den maximalen Festplattenspeicher für jedes Protokoll, einschließlich seiner rotierten Protokolldateien, auf einem Core-Gerät konfigurieren. Sie können beispielsweise den maximalen kombinierten Festplattenspeicher für `greengrass.log` und rotierte `greengrass.log` Dateien konfigurieren. Weitere Informationen finden Sie unter `logging.totalLogsSizeKB` dem Parameter der Greengrass-Kernkomponente

und dem `diskSpaceLimit` Parameter der Protokollmanagerkomponente, den Sie für AWS IoT Greengrass -Core-Softwareprotokolle und Komponentenprotokolle angeben können.

- Größenbeschränkung der Protokolldatei

  Sie können die maximale Dateigröße für jede Protokolldatei konfigurieren. Nachdem eine Protokolldatei diese Dateigrößenbeschränkung überschreitet, erstellt die AWS IoT Greengrass Core-Software eine neue Protokolldatei. Die Log Manager-Komponente Version 2.28 (und früher) schreibt nur rotierte Protokolldateien in CloudWatch Logs, sodass Sie eine niedrigere Dateigrößenbeschränkung angeben können, um Protokolle häufiger in CloudWatch Logs zu schreiben. Die Log Manager-Komponente Version 2.3.0 (und höher) verarbeitet und lädt alle Protokolle hoch, anstatt darauf zu warten, dass sie rotiert werden. Weitere Informationen finden Sie unter dem Parameter für die Begrenzung der Protokolldateigröße der Greengrass-Kernkomponente (`logging.fileSizeKB`).

- Minimale Protokollstufen

  Sie können die minimale Protokollebene konfigurieren, die die Greengrass-Kernkomponente in Dateisystemprotokolle schreibt. Sie können beispielsweise `DEBUG` Ebenenprotokolle angeben, um die Fehlerbehebung zu unterstützen, oder Sie können `ERROR` Ebenenprotokolle angeben, um die Anzahl der Protokolle zu reduzieren, die ein Core-Gerät erstellt. Weitere Informationen finden Sie im Protokollebenenparameter der Greengrass-Kernkomponente (`logging.level`).

  Sie können auch die minimale Protokollebene konfigurieren, die die Protokollmanagerkomponente in CloudWatch Protokolle schreibt. Sie können beispielsweise eine höhere Protokollebene angeben, um die Protokollierungskosten zu senken. Weitere Informationen finden Sie im `minimumLogLevel` Parameter der Protokollmanagerkomponente, den Sie für -AWS IoT GreengrassCore-Softwareprotokolle und Komponentenprotokolle angeben können.

- Intervall zum Überprüfen, ob Protokolle in CloudWatch Protokolle geschrieben werden

  Um zu erhöhen oder zu verringern, wie oft die Protokollmanager-Komponente Protokolle in CloudWatch Protokolle schreibt, können Sie das Intervall konfigurieren, in dem sie nach neuen Protokolldateien sucht, die geschrieben werden sollen. Sie können beispielsweise ein niedrigeres Intervall angeben, um Protokolle in - CloudWatch Protokollen früher anzuzeigen als im Standardintervall von 5 Minuten. Sie können ein höheres Intervall angeben, um die Kosten zu senken, da die Protokollmanager-Komponente Protokolldateien in weniger Anforderungen stapelt. Weitere Informationen finden Sie im Upload-Intervallparameter der Protokollmanagerkomponente (`periodicUploadIntervalSec`).

- Protokollformat

Sie können wählen, ob die AWS IoT Greengrass Core-Software Protokolle im Text- oder JSON-Format schreibt. Wählen Sie Textformat, wenn Sie Protokolle lesen, oder JSON-Format, wenn Sie eine Anwendung zum Lesen oder Analysieren von Protokollen verwenden. Weitere Informationen finden Sie im Protokollformatparameter der Greengrass-Kernkomponente (`logging.format`).

- Lokaler Dateisystem-Protokollordner

  Sie können den Protokollordner von */greengrass/v2*`/logs` in einen anderen Ordner auf dem Core-Gerät ändern. Weitere Informationen finden Sie im Ausgabeverzeichnisparameter der Greengrass-Kernkomponente (`logging.outputDirectory`).

AWS CloudTrail-Protokolle

AWS IoT Greengrass lässt sich in integrieren, einen ServiceAWS CloudTrail, der die Aktionen eines Benutzers, einer Rolle oder AWS-Service in aufzeichnetAWS IoT Greengrass. Weitere Informationen finden Sie unter Protokollieren von AWS IoT Greengrass V2 API-Aufrufen mit AWS CloudTrail.

Protokollieren von AWS IoT Greengrass V2 API-Aufrufen mit AWS CloudTrail

AWS IoT Greengrass V2 ist in integriert, einem Service AWS CloudTrail, der die Aktionen eines Benutzers, einer Rolle oder eines - AWS Services in aufzeichnet AWS IoT Greengrass Version 2. CloudTrail erfasst alle API-Aufrufe für AWS IoT Greengrass als Ereignisse. Zu den erfassten Aufrufen gehören Aufrufe von der AWS IoT Greengrass Konsole und Codeaufrufe der AWS IoT Greengrass API-Operationen.

Wenn Sie einen Trail erstellen, können Sie die kontinuierliche Bereitstellung von CloudTrail Ereignissen an einen 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 Anfrage AWS 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 V2 Informationen in CloudTrail

CloudTrail wird auf Ihrem aktiviert AWS-Konto , wenn Sie das Konto erstellen. Wenn eine Aktivität in auftritt AWS IoT Greengrass, wird diese Aktivität in einem - CloudTrail Ereignis zusammen mit anderen - AWS Serviceereignissen 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 .

Erstellen Sie für eine fortlaufende Aufzeichnung der Ereignisse in Ihrem AWS-Konto, einschließlich Ereignissen für AWS IoT Greengrass, einen Trail. Ein Trail ermöglicht CloudTrail die Bereitstellung von Protokolldateien an einen S3-Bucket. Wenn Sie einen Trail in der Konsole erstellen, gilt der Trail standardmäßig für alle AWS-Region s. Der Trail protokolliert Ereignisse aus allen Regionen in der - AWS Partition und stellt die Protokolldateien in dem von Ihnen angegebenen S3-Bucket bereit. Darüber hinaus können Sie andere - AWS Services 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 Greengrass V2 Aktionen werden von protokolliert CloudTrail und sind in der AWS IoT Greengrass V2 API-Referenz zu dokumentiert. Aufrufe der `CancelDeployment` Aktionen `CreateComponentVersion`, `CreateDeployment` und erzeugen beispielsweise Einträge in den CloudTrail Protokolldateien.

Jeder Ereignis- oder Protokolleintrag enthält Informationen zu dem Benutzer, der die Anforderung generiert hat. Die Identitätsinformationen unterstützen Sie bei der Ermittlung der folgenden Punkte:

- Ob die Anforderung mit Root- oder AWS Identity and Access Management (IAM)-Benutzeranmeldeinformationen ausgeführt wurde.
- Gibt an, ob die Anforderung mit temporären Sicherheitsanmeldeinformationen für eine Rolle oder einen Verbundbenutzer gesendet wurde.
- Ob die Anforderung von einem anderen - AWS Service gestellt wurde.

Weitere Informationen finden Sie unter CloudTrail -Element userIdentity.

AWS IoT Greengrass -Datenereignisse in CloudTrail

Datenereignisse liefern Informationen über die Ressourcenoperationen, die für oder in einer Ressource ausgeführt werden (z. B. das Abrufen einer Komponentenversion oder die Konfiguration einer Bereitstellung). Sie werden auch als Vorgänge auf Datenebene bezeichnet. Datenereignisse sind oft Aktivitäten mit hohem Volume. Standardmäßig protokolliert CloudTrail keine Datenereignisse. Der CloudTrail Ereignisverlauf zeichnet keine Datenereignisse auf.

Für Datenereignisse werden zusätzliche Gebühren fällig. Weitere Informationen zu CloudTrail Preisen finden Sie unter -AWS CloudTrail Preise.

Sie können Datenereignisse für die AWS IoT Greengrass Ressourcentypen mithilfe der CloudTrail Konsole AWS CLI oder API CloudTrail -Operationen protokollieren. Die Tabelle in diesem Abschnitt zeigt die Ressourcentypen, die für verfügbar sind AWS IoT Greengrass.

- Um Datenereignisse mit der CloudTrail Konsole zu protokollieren, erstellen Sie einen Trail oder Ereignisdatenspeicher, um Datenereignisse zu protokollieren, oder aktualisieren Sie einen vorhandenen Trail oder Ereignisdatenspeicher, um Datenereignisse zu protokollieren.

  1. Wählen Sie Datenereignisse aus, um Datenereignisse zu protokollieren.
  2. Wählen Sie in der Liste Datenereignistyp den Ressourcentyp aus, für den Sie Datenereignisse protokollieren möchten.
  3. Wählen Sie die Protokollselektorvorlage aus, die Sie verwenden möchten. Sie können alle Datenereignisse für den Ressourcentyp protokollieren, alle `readOnly` Ereignisse `readOnly`protokollieren, alle `writeOnly` Ereignisse protokollieren oder eine benutzerdefinierte Protokollauswahlvorlage erstellen, um nach den `resources.ARN` Feldern `eventName`, und zu filtern.

- Um Datenereignisse mit der zu protokollieren AWS CLI, konfigurieren Sie den `--advanced-event-selectors` Parameter so, dass das `eventCategory` Feld gleich `Data` und das `resources.type` Feld gleich dem Ressourcentypwert ist (siehe Tabelle). Sie können Bedingungen hinzufügen, um nach den Werten der `resources.ARN` Felder `readOnlyeventName`, und zu filtern.
  - Um einen Trail für die Protokollierung von Datenereignissen zu konfigurieren, führen Sie den put-event-selectors Befehl aus. Weitere Informationen finden Sie unter Protokollieren von Datenereignissen für Trails mit der AWS CLI.

- Um einen Ereignisdatenspeicher für die Protokollierung von Datenereignissen zu konfigurieren, führen Sie den create-event-data-store Befehl aus, um einen neuen Ereignisdatenspeicher für die Protokollierung von Datenereignissen zu erstellen, oder führen Sie den update-event-data-store Befehl aus, um einen vorhandenen Ereignisdatenspeicher zu aktualisieren. Weitere Informationen finden Sie unter Protokollieren von Datenereignissen für Ereignisdatenspeicher mit der AWS CLI.

In der folgenden Tabelle sind die AWS IoT Greengrass Ressourcentypen aufgeführt. In der Spalte Datenereignistyp (Konsole) wird der Wert angezeigt, der aus der Liste Datenereignistyp in der CloudTrail Konsole ausgewählt werden kann. Die Spalte resources.type value zeigt den `resources.type` Wert an, den Sie bei der Konfiguration erweiterter Ereignisselektoren mit der AWS CLI oder CloudTrail APIs angeben würden. Die Spalte Daten-APIs, die in protokolliert CloudTrail wurden, zeigt die API-Aufrufe an, CloudTrail die für den Ressourcentyp protokolliert wurden.

Typ des Datenereignisses (Konsole)	resources.type-Wert	Daten-APIs, die bei protokolliert wurden CloudTrail
IoT-Zertifikat	`AWS::IoT::Certificate`	• VerifyClientDeviceIdentity • VerifyClientDeviceIoTCertificateAssociation
IoT-Greengrass-Komponentenversion	`AWS::GreengrassV2::ComponentVersion`	• ResolveComponentCandidates
IoT-Greengrass-Bereitstellung	`AWS::GreengrassV2::Deployment`	• GetDeploymentConfiguration
IoT-Objekt	`AWS::IoT::Thing`	• ListThingGroupsForCoreDevices • PutCertificateAuthorities • VerifyClientDeviceIoTCertificateAssociation

> **Note**
>
> Greengrass protokolliert keine Ereignisse mit Zugriffsverweigerung.

Sie können erweiterte Ereignisselektoren so konfigurieren`eventName`, dass sie nach den `resources.`ARN Feldern `readOnly`, und filtern, um nur die Ereignisse zu protokollieren, die für Sie wichtig sind.

Fügen Sie einen Filter für hinzu`eventName`, um bestimmte Daten-APIs einzuschließen oder auszuschließen.

Weitere Informationen zu diesen Feldern finden Sie unter AdvancedFieldSelector.

Die folgenden Beispiele zeigen, wie Sie erweiterte Selektoren mithilfe der konfigurieren AWS CLI. Ersetzen Sie *`TrailName`* und *`Region`* durch Ihre eigenen Informationen.

Example – Protokollieren von Datenereignissen für IoT-Objekte

```
aws cloudtrail put-event-selectors --trail-name TrailName --region region \
--advanced-event-selectors \
'[
    {
            "Name": "Log all thing data events",
            "FieldSelectors": [
                { "Field": "eventCategory", "Equals": ["Data"] },
                { "Field": "resources.type", "Equals": ["AWS::IoT::Thing"] }
            ]
        }
]'
```

Example – Filtern nach einer bestimmten IoT-Objekt-API

```
aws cloudtrail put-event-selectors --trail-name TrailName --region region \
--advanced-event-selectors \
'[
    {
            "Name": "Log IoT Greengrass PutCertificateAuthorities API calls",
            "FieldSelectors": [
                { "Field": "eventCategory", "Equals": ["Data"] },
                { "Field": "resources.type", "Equals": ["AWS::IoT::Thing"] },
```

```
                { “Field”: “eventName”, “Equals”: [“PutCertificateAuthorities”] }
            ]
    }
]’
```

Example – Protokollieren aller Greengrass-Datenereignisse

```
aws cloudtrail put-event-selectors --trail-name TrailName --region region \
--advanced-event-selectors \
‘[
    {
        “Name”: “Log all certificate data events”,
        “FieldSelectors”: [
            {
                “Field”: “eventCategory”,
                “Equals”: [
                    “Data”
                ]
            },
            {
                “Field”: “resources.type”,
                “Equals”: [
                    “AWS::IoT::Certificate”
                ]
            }
        ]
    },
    {
        “Name”: “Log all component version data events”,
        “FieldSelectors”: [
            {
                “Field”: “eventCategory”,
                “Equals”: [
                    “Data”
                ]
            },
            {
                “Field”: “resources.type”,
                “Equals”: [
                    “AWS::GreengrassV2::ComponentVersion”
                ]
            }
        ]
```

```
    },
    {
        “Name”: “Log all deployment version”,
        “FieldSelectors”: [
            {
                “Field”: “eventCategory”,
                “Equals”: [
                    “Data”
                ]
            },
            {
                “Field”: “resources.type”,
                “Equals”: [
                    “AWS::GreengrassV2::Deployment”
                ]
            }
        ]
    },
    {
        “Name”: “Log all thing data events”,
        “FieldSelectors”: [
            {
                “Field”: “eventCategory”,
                “Equals”: [
                    “Data”
                ]
            },
            {
                “Field”: “resources.type”,
                “Equals”: [
                    “AWS::IoT::Thing”
                ]
            }
        ]
    }
]’
```

AWS IoT Greengrass -Verwaltungsereignisse in CloudTrail

Verwaltungsereignisse liefern Informationen zu Verwaltungsvorgängen, die für Ressourcen in Ihrem AWS Konto ausgeführt werden. Sie werden auch als Vorgänge auf Steuerebene bezeichnet. Standardmäßig CloudTrail protokolliert Verwaltungsereignisse.

AWS IoT Greengrass protokolliert alle Operationen auf AWS IoT Greengrass Steuerebene als Verwaltungsereignisse. Eine Liste der Operationen auf AWS IoT Greengrass Steuerebene, die in AWS IoT Greengrass protokolliert CloudTrail, finden Sie in der APIAWS IoT Greengrass -Referenz, Version 2.

Grundlegendes zu AWS IoT Greengrass V2 Protokolldateieinträgen

Ein Trail ist eine Konfiguration, die die Bereitstellung von Ereignissen als Protokolldateien an einen von Ihnen angegebenen S3-Bucket ermöglicht. CloudTrail Protokolldateien enthalten einen oder mehrere Protokolleinträge. Ein Ereignis stellt eine einzelne Anforderung aus einer beliebigen Quelle dar. Sie 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 `CreateDeployment` Aktion demonstriert.


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Administrator",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Administrator"
    },
    "eventTime": "2021-01-06T02:38:05Z",
    "eventSource": "greengrass.amazonaws.com",
    "eventName": "CreateDeployment",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "203.0.113.0",
    "userAgent": "aws-cli/2.1.9 Python/3.7.9 Windows/10 exe/AMD64 prompt/off command/
greengrassv2.create-deployment",
    "requestParameters": {
        "deploymentPolicies": {
            "failureHandlingPolicy": "DO_NOTHING",
            "componentUpdatePolicy": {
                "timeoutInSeconds": 60,
                "action": "NOTIFY_COMPONENTS"
            },
            "configurationValidationPolicy": {
```

```
                "timeoutInSeconds": 60
            }
        },
        "deploymentName": "Deployment for MyGreengrassCoreGroup",
        "components": {
            "aws.greengrass.Cli": {
                "componentVersion": "2.0.3"
            }
        },
        "iotJobConfiguration": {},
        "targetArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
MyGreengrassCoreGroup"
    },
    "responseElements": {
        "iotJobArn": "arn:aws:iot:us-west-2:123456789012:job/fdfeba1d-ac6d-44ef-
ab28-54f684ea578d",
        "iotJobId": "fdfeba1d-ac6d-44ef-ab28-54f684ea578d",
        "deploymentId": "4196dddc-0a21-4c54-a985-66a525f6946e"
    },
    "requestID": "311b9529-4aad-42ac-8408-c06c6fec79a9",
    "eventID": "c0f3aa2c-af22-48c1-8161-bad4a2ab1841",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "123456789012"
}
```


Erfassen von Telemetriedaten zum Systemstatus von -AWS IoT GreengrassCore-Geräten

Systemzustand-Telemetriedaten sind Diagnosedaten, mit denen Sie die Leistung kritischer Vorgänge auf Ihren Greengrass-Kerngeräten überwachen können. Sie können Projekte und Anwendungen erstellen, um Telemetriedaten von Ihren Edge-Geräten abzurufen, zu analysieren, zu transformieren und zu melden. Domainexperten wie Prozessingenieure können diese Anwendungen verwenden, um Einblicke in den Flottenzustand zu erhalten.

Sie können die folgenden Methoden verwenden, um Telemetriedaten von Ihren Greengrass-Core-Geräten zu sammeln:

- Komponente des Telemetrie-Emitters – Die Komponente des Telemetrie-Emitters des Kerns (`aws.greengrass.telemetry.NucleusEmitter`) auf einem Greengrass-Kerngerät veröffentlicht standardmäßig Telemetriedaten zum `$local/greengrass/telemetry` Thema. Sie können die in diesem Thema veröffentlichten Daten verwenden, um lokal auf Ihrem Core-Gerät zu reagieren, auch wenn Ihr Gerät über eine eingeschränkte Konnektivität zur Cloud verfügt. Optional können Sie die Komponente auch so konfigurieren, dass Telemetriedaten in einem AWS IoT Core MQTT-Thema Ihrer Wahl veröffentlicht werden.

  Sie müssen die Kern-Emitterkomponente auf einem Core-Gerät bereitstellen, um Telemetriedaten zu veröffentlichen. Für die Veröffentlichung von Telemetriedaten im lokalen Thema fallen keine Kosten an. Die Verwendung eines MQTT-Themas zum Veröffentlichen von Daten in AWS Cloud der unterliegt jedoch den AWS IoT Core Preisen .

  AWS IoT Greengrass bietet mehrere Community-Komponenten, mit denen Sie Telemetriedaten mithilfe von InfluxDB und Grafana lokal auf Ihrem Core-Gerät analysieren und visualisieren können. Diese Komponenten verwenden Telemetriedaten aus der Kern-Emitterkomponente. Weitere Informationen finden Sie in der README für die Herausgeberkomponente von InfluxDB.
- Telemetrie-Agent – Der Telemetrie-Agent auf Greengrass-Core-Geräten sammelt lokale Telemetriedaten und veröffentlicht sie in Amazon, EventBridge ohne dass Kunden interagieren müssen. Core-Geräte veröffentlichen Telemetriedaten EventBridge nach bestem Wissen und Gewissen in . Beispielsweise können -Core-Geräte Telemetriedaten möglicherweise nicht bereitstellen, wenn sie offline sind.

  Das Telemetrie-Agent-Feature ist standardmäßig für alle Greengrass-Core-Geräte aktiviert. Sie beginnen automatisch mit dem Empfang von Daten, sobald Sie ein Greengrass-Core-Gerät eingerichtet haben. Neben Ihren Datenverbindungskosten AWS IoT Core ist die Datenübertragung vom Core-Gerät zu kostenlos. Dies liegt daran, dass der Agent zu einem AWS reservierten Thema veröffentlicht. Abhängig von Ihrem Anwendungsfall können jedoch Kosten anfallen, wenn Sie die Daten erhalten oder verarbeiten.

  > **Note**
  >
  > Amazon EventBridge ist ein Event-Bus-Service, mit dem Sie Ihre Anwendungen mit Daten aus einer Vielzahl von Quellen verbinden können, z. B. Greengrass-Core-Geräten. Weitere Informationen finden Sie unter Was ist Amazon EventBridge? im Amazon- EventBridge Benutzerhandbuch.

Um sicherzustellen, dass die AWS IoT Greengrass -Core-Software ordnungsgemäß funktioniert, AWS IoT Greengrass verwendet die Daten für Entwicklungs- und Qualitätsverbesserungszwecke. Diese Funktion hilft auch dabei, neue und erweiterte Edge-Funktionen zu entwickeln. AWS IoT Greengrass speichert Telemetriedaten bis zu sieben Tage lang.

In diesem Abschnitt wird beschrieben, wie Sie den Telemetrie-Agenten konfigurieren und verwenden. Informationen zur Konfiguration der Kerntelemetrie-Emitterkomponente finden Sie unter Telemetrie-Emitter.

Themen

Telemetriemetriken

In der folgenden Tabelle werden die Metriken beschrieben, die vom Telemetrie-Agenten veröffentlicht werden.

Name	Beschreibung	
System (System)		
`SystemMemUsage`	Die Speichermenge, die derzeit von allen Anwendungen auf dem Greengrass-Core-Gerät verwendet wird, einschließlich des Betriebssystems.	
`CpuUsage`	Die Menge an CPU, die derzeit von allen Anwendungen auf dem Greengrass-Core-Gerät verwendet wird, einschließlich des Betriebssystems.	

Name	Beschreibung
`TotalNumberOfFDs`	Die Anzahl der vom Betriebssystem des Greengrass-Core-Geräts gespeicherten Dateideskriptoren. Ein Dateideskriptor identifiziert eindeutig eine geöffnete Datei.
Greengrass-Kern	
`NumberOfComponentsRunning`	Die Anzahl der Komponenten, die auf dem Greengrass-Kerngerät ausgeführt werden.
`NumberOfComponentsErrored`	Die Anzahl der Komponenten, die sich auf dem Greengrass-Core-Gerät im Fehlerzustand befinden.
`NumberOfComponentsInstalled`	Die Anzahl der Komponenten, die auf dem Greengrass-Core-Gerät installiert sind.
`NumberOfComponentsStarting`	Die Anzahl der Komponenten, die auf dem Greengrass-Kerngerät beginnen.
`NumberOfComponentsNew`	Die Anzahl der Komponenten, die auf dem Greengrass-Kerngerät neu sind.
`NumberOfComponentsStopping`	Die Anzahl der Komponenten, die auf dem Greengrass-Core-Gerät anhalten.
`NumberOfComponentsFinished`	Die Anzahl der Komponenten, die auf dem Greengrass-Kerngerät fertig sind.

Name	Beschreibung	
`NumberOfComponents Broken`	Die Anzahl der Komponent en, die auf dem Greengrass-Kerngerät unterbrochen sind.	
`NumberOfComponents Stateless`	Die Anzahl der Komponent en, die auf dem Greengrass-Kerngerät zustandslos sind.	
Client-Geräteauthentifizier ung – Diese Funktion erfordert v2.4.0 oder höher der Client-Geräteauthentifizier ungskomponente.		
`VerifyClientDevice Identity.Success`	Gibt an, wie oft die Überprüfu ng, dass die Client-Ge räteidentität erfolgreich war, erfolgreich war.	
`VerifyClientDevice Identity.Failure`	Gibt an, wie oft die Überprüfu ng, dass die Identität des Client-Geräts fehlgeschlagen ist, fehlgeschlagen ist.	
`AuthorizeClientDev iceActions.Success`	Gibt an, wie oft das Client-Ge rät autorisiert ist, angeforderte Aktionen durchzuführen.	
`AuthorizeClientDev iceActions.Failure`	Gibt an, wie oft das Client-Gerät nicht berechtigt ist, angeforderte Aktionen durchzuführen.	
`GetClientDeviceAut hToken.Success`	Gibt an, wie oft das Client-Ge rät erfolgreich authentifiziert wurde.	

Name	Beschreibung
`GetClientDeviceAuthToken.Failure`	Gibt an, wie oft das Client-Gerät nicht authentifiziert werden kann.
`SubscribeToCertificateUpdates.Success`	Die Anzahl der erfolgreichen Abonnements für Zertifikatsaktualisierungen.
`SubscribeToCertificateUpdates.Failure`	Die Anzahl der erfolglosen Versuche, Zertifikatsaktualisierungen zu abonnieren.
`ServiceError`	Die Anzahl der unbehandelten internen Fehler in der gesamten Client-Geräte-Authentifizierung.
Stream Manager – Diese Funktion erfordert v2.7.0 oder höher der Greengrass-Kernkomponente.	
`BytesAppended`	Die Anzahl der Bytes der an den Stream-Manager angehängten Daten.
`BytesUploadedToIoTAnalytics`	Die Anzahl der Bytes an Daten, die der Stream-Manager in Kanäle in exportiertAWS IoT Analytics.
`BytesUploadedToKinesis`	Die Anzahl der Bytes an Daten, die Stream Manager in Streams in Amazon Kinesis Data Streams exportiert.

Name	Beschreibung	
`BytesUploadedToIoTSiteWise`	Die Anzahl der Bytes an Daten, die der Stream-Manager in Komponenteneigenschaften in exportiertAWS IoT SiteWise.	
`BytesUploadedToS3`	Die Anzahl der Bytes an Daten, die Stream Manager in Objekte in Amazon S3 exportiert.	

Konfigurieren von Telemetrie-Agent-Einstellungen

Der Telemetrie-Agent verwendet die folgenden Standardeinstellungen:

- Der Telemetrie-Agent aggregiert Telemetriedaten stündlich.
- Der Telemetrie-Agent veröffentlicht alle 24 Stunden eine Telemetrienachricht.

Der Telemetrie-Agent veröffentlicht Daten mithilfe des MQTT-Protokolls mit einem Quality of Service (QoS)-Level von 0, was bedeutet, dass er die Zustellung nicht bestätigt oder erneut versucht, Veröffentlichungsversuche durchzuführen. Telemetrienachrichten teilen sich eine MQTT-Verbindung mit anderen Nachrichten für Abonnements, die für bestimmt sindAWS IoT Core.

Neben Ihren Datenverbindungskosten AWS IoT Core ist die Datenübertragung vom Kern zu kostenlos. Dies liegt daran, dass der Agent zu einem AWS reservierten Thema veröffentlicht. Abhängig von Ihrem Anwendungsfall können jedoch Kosten anfallen, wenn Sie die Daten erhalten oder verarbeiten.

Sie können die Telemetrie-Agent-Funktion für jedes Greengrass-Kerngerät aktivieren oder deaktivieren. Sie können auch die Intervalle konfigurieren, in denen das Core-Gerät Daten aggregiert und veröffentlicht. Um Telemetrie zu konfigurieren, passen Sie den Telemetrie-Konfigurationsparameter an, wenn Sie die Greengrass-Kernkomponente bereitstellen.

Abonnieren von Telemetriedaten in EventBridge

Sie können Regeln in Amazon erstellen EventBridge , die definieren, wie Telemetriedaten verarbeitet werden, die vom Telemetrieagenten auf dem Greengrass-Kerngerät veröffentlicht werden. Wenn die Daten EventBridge empfängt, werden die in Ihren Regeln definierten Zielaktionen aufgerufen. Sie können beispielsweise Ereignisregeln erstellen, die Benachrichtigungen senden, Ereignisinformationen speichern, Korrekturmaßnahmen ergreifen oder andere Ereignisse aufrufen.

Telemetrieereignisse

Telemetrieereignisse verwenden das folgende Format.

```
{
  "version": "0",
  "id": "a09d303e-2f6e-3d3c-a693-8e33f4fe3955",
  "detail-type": "Greengrass Telemetry Data",
  "source": "aws.greengrass",
  "account": "123456789012",
  "time": "2020-11-30T20:45:53Z",
  "region": "us-east-1",
  "resources": [],
  "detail": {
    "ThingName": "MyGreengrassCore",
    "Schema": "2020-07-30",
    "ADP": [
      {
        "TS": 1602186483234,
        "NS": "SystemMetrics",
        "M": [
          {
            "N": "TotalNumberOfFDs",
            "Sum": 6447.0,
            "U": "Count"
          },
          {
            "N": "CpuUsage",
            "Sum": 15.458333333333332,
            "U": "Percent"
          },
          {
            "N": "SystemMemUsage",
            "Sum": 10201.0,
            "U": "Megabytes"
```

```
        }
      ]
    },
    {
      "TS": 1602186483234,
      "NS": "GreengrassComponents",
      "M": [
        {
          "N": "NumberOfComponentsStopping",
          "Sum": 0.0,
          "U": "Count"
        },
        {
          "N": "NumberOfComponentsStarting",
          "Sum": 0.0,
          "U": "Count"
        },
        {
          "N": "NumberOfComponentsBroken",
          "Sum": 0.0,
          "U": "Count"
        },
        {
          "N": "NumberOfComponentsFinished",
          "Sum": 1.0,
          "U": "Count"
        },
        {
          "N": "NumberOfComponentsInstalled",
          "Sum": 0.0,
          "U": "Count"
        },
        {
          "N": "NumberOfComponentsRunning",
          "Sum": 7.0,
          "U": "Count"
        },
        {
          "N": "NumberOfComponentsNew",
          "Sum": 0.0,
          "U": "Count"
        },
        {
          "N": "NumberOfComponentsErrored",
```

```
            "Sum": 0.0,
            "U": "Count"
          },
          {
            "N": "NumberOfComponentsStateless",
            "Sum": 0.0,
            "U": "Count"
          }
        ]
      },
      {
        "TS": 1602186483234,
        "NS": "aws.greengrass.ClientDeviceAuth",
        "M": [
          {
            "N": "VerifyClientDeviceIdentity.Success",
            "Sum": 3.0,
            "U": "Count"
          },
          {
            "N": "VerifyClientDeviceIdentity.Failure",
            "Sum": 1.0,
            "U": "Count"
          },
          {
            "N": "AuthorizeClientDeviceActions.Success",
            "Sum": 20.0,
            "U": "Count"
          },
          {
            "N": "AuthorizeClientDeviceActions.Failure",
            "Sum": 5.0,
            "U": "Count"
          },
          {
            "N": "GetClientDeviceAuthToken.Success",
            "Sum": 5.0,
            "U": "Count"
          },
          {
            "N": "GetClientDeviceAuthToken.Failure",
            "Sum": 2.0,
            "U": "Count"
          },
```

```
          {
            "N": "SubscribeToCertificateUpdates.Success",
            "Sum": 10.0,
            "U": "Count"
          },
          {
            "N": "SubscribeToCertificateUpdates.Failure",
            "Sum": 1.0,
            "U": "Count"
          },
          {
            "N": "ServiceError",
            "Sum": 3.0,
            "U": "Count"
          }
        ]
      },
      {
        "TS": 1602186483234,
        "NS": "aws.greengrass.StreamManager",
        "M": [
          {
            "N": "BytesAppended",
            "Sum": 157745524.0,
            "U": "Bytes"
          },
          {
            "N": "BytesUploadedToIoTAnalytics",
            "Sum": 149012.0,
            "U": "Bytes"
          },
          {
            "N": "BytesUploadedToKinesis",
            "Sum": 12192.0,
            "U": "Bytes"
          },
          {
            "N": "BytesUploadedToIoTSiteWise",
            "Sum": 13321.0,
            "U": "Bytes"
          },
          {
            "N": "BytesUploadedToS3",
            "Sum": 12213.0,
```

```
            "U": "Bytes"
          }
        ]
      }
    ]
  }
}
```

Das ADPArray enthält eine Liste aggregierter Datenpunkte mit den folgenden Eigenschaften:

`TS`

Der Zeitstempel der Datenerfassung.

`NS`

Der Metrik-Namespace.

`M`

Die Liste der Metriken. Eine Metrik enthält die folgenden Eigenschaften:

`N`

Name der Metrik.

`Sum`

Die Summe der Metrikwerte in diesem Telemetrieereignis.

`U`

Die Einheit des Metrikwerts.

Weitere Informationen zu den einzelnen Metriken finden Sie unter Telemetriemetriken.

Voraussetzungen für das Erstellen EventBridge von Regeln

Bevor Sie eine - EventBridge Regel für erstellenAWS IoT Greengrass, sollten Sie Folgendes tun:

- Machen Sie sich mit Ereignissen, Regeln und Zielen in vertraut EventBridge.
- Erstellen und konfigurieren Sie die Ziele, die von Ihren EventBridge Regeln aufgerufen werden. Regeln können viele Arten von Zielen aufrufen, z. B. Amazon Kinesis-Streams, -AWS LambdaFunktionen, Amazon SNS-Themen und Amazon SQSWarteschlangen.

Ihre EventBridge Regel und die zugehörigen Ziele müssen sich in der AWS-Region befinden, in der Sie Ihre Greengrass-Ressourcen erstellt haben. Weitere Informationen finden Sie unter Service-Endpunkte und -Kontingente im Allgemeine AWS-Referenz.

Weitere Informationen finden Sie unter Was ist Amazon EventBridge? und Erste Schritte mit Amazon EventBridge im Amazon- EventBridge Benutzerhandbuch.

Erstellen einer Ereignisregel zum Abrufen von Telemetriedaten (Konsole)

Führen Sie die folgenden Schritte aus, um mit eine EventBridge Regel AWS Management Console zu erstellen, die vom Greengrass-Kerngerät veröffentlichte Telemetriedaten empfängt. Auf diese Weise können Webserver, E-Mail-Adressen und andere Themenabonnenten auf das Ereignis reagieren. Weitere Informationen finden Sie unter Erstellen einer EventBridge Regel, die bei einem Ereignis aus einer -AWSRessource ausgelöst wird im Amazon EventBridge -Benutzerhandbuch.

1. Öffnen Sie die Amazon EventBridge-Konsole und wählen Sie Regel erstellen aus.
2. Geben Sie unter Name and description (Name und Beschreibung) einen Namen und eine Beschreibung für die Regel ein.
3. Konfigurieren Sie unter Define pattern (Muster definieren) das Regelmuster.
   a. Wählen Sie Event pattern (Ereignismuster) aus.
   b. Wählen Sie Pre-defined pattern by service (Vordefiniertes Muster nach Service)aus.
   c. Wählen Sie für Service provider (Serviceanbieter) die Option AWS aus.
   d. Wählen Sie für Service-Name die Option Greengrassaus.
   e. Wählen Sie für Ereignistyp die Option Greengrass-Telemetriedaten aus.
4. Behalten Sie unter Select event bus (Ereignisbus auswählen) die standardmäßigen Ereignisbusoptionen bei.
5. Konfigurieren Sie unter Select targets (Ziele auswählen) das Ziel. Im folgenden Beispiel wird eine Amazon SQS-Warteschlange verwendet, Sie können jedoch andere Zieltypen konfigurieren.
   a. Wählen Sie für Ziel die Option SQS-Warteschlange aus.
   b. Wählen Sie für Warteschlange* Ihre Zielwarteschlange aus.
6. Definieren Sie unter Tags - optional Tags für die Regel oder lassen Sie die Felder leer.
7. Wählen Sie Erstellen.

Erstellen einer Ereignisregel zum Abrufen von Telemetriedaten (CLI)

Führen Sie die folgenden Schritte aus, um mit eine EventBridge Regel AWS CLI zu erstellen, die Telemetriedaten empfängt, die von Greengrass-Core-Geräten veröffentlicht wurden. Auf diese Weise können Webserver, E-Mail-Adressen und andere Themenabonnenten auf das Ereignis reagieren.

1. Erstellen Sie die -Regel.

   - Ersetzen Sie *Objektname* durch den Objektnamen des Core-Geräts.

   Linux or Unix

   ```
   aws events put-rule \
     --name MyGreengrassTelemetryEventRule \
     --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"ThingName
   \": [\"thing-name\"]}}"
   ```

   Windows Command Prompt (CMD)

   ```
   aws events put-rule ^
     --name MyGreengrassTelemetryEventRule ^
     --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"ThingName
   \": [\"thing-name\"]}}"
   ```

   PowerShell

   ```
   aws events put-rule `
     --name MyGreengrassTelemetryEventRule `
     --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"ThingName
   \": [\"thing-name\"]}}"
   ```

   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 andere Zieltypen konfigurieren.

   - Ersetzen Sie *queue-arn* durch den ARN Ihrer Amazon SQS-Warteschlange.

Linux or Unix

```
aws events put-targets \
  --rule MyGreengrassTelemetryEventRule \
  --targets "Id"="1","Arn"="queue-arn"
```

Windows Command Prompt (CMD)

```
aws events put-targets ^
  --rule MyGreengrassTelemetryEventRule ^
  --targets "Id"="1","Arn"="queue-arn"
```

PowerShell

```
aws events put-targets `
  --rule MyGreengrassTelemetryEventRule `
  --targets "Id"="1","Arn"="queue-arn"
```

> **Note**
>
> Damit Amazon Ihre Zielwarteschlange EventBridge aufrufen kann, müssen Sie Ihrem Thema eine ressourcenbasierte Richtlinie hinzufügen. Weitere Informationen finden Sie unter Amazon SQS-Berechtigungen im Amazon- EventBridge Benutzerhandbuch.

Weitere Informationen finden Sie unter Ereignisse und Ereignismuster in EventBridge im Amazon-EventBridge Benutzerhandbuch.

Erhalten Sie Benachrichtigungen über den Bereitstellungs- und Komponentenstatus

Mit den EventBridge Amazon-Veranstaltungsregeln erhalten Sie Benachrichtigungen über Statusänderungen für Ihre Greengrass-Bereitstellungen, die von Ihren Geräten empfangen wurden, und über installierte Komponenten auf Ihrem Gerät. EventBridge liefert einen Stream von Systemereignissen nahezu in Echtzeit, der Änderungen an AWS Ressourcen beschreibt. AWS

IoT Greengrass sendet diese Ereignisse nach EventBridge bestem Wissen und Gewissen an. Das bedeutet, dass AWS IoT Greengrass versucht wird, alle Ereignisse an zu senden, EventBridge aber in einigen seltenen Fällen kann es vorkommen, dass ein Ereignis nicht zugestellt wird. Darüber hinaus AWS IoT Greengrass können mehrere Kopien eines bestimmten Ereignisses gesendet werden, was bedeutet, dass Ihre Event-Listener die Ereignisse möglicherweise nicht in der Reihenfolge empfangen, in der sie aufgetreten sind.

> **Note**
>
> Amazon EventBridge ist ein Event-Bus-Service, mit dem Sie Ihre Anwendungen mit Daten aus einer Vielzahl von Quellen verbinden können, wie z. B. Greengrass-Kerngeräten sowie Bereitstellungs- und Komponentenbenachrichtigungen. Weitere Informationen finden Sie unter Was ist Amazon EventBridge? im EventBridge Amazon-Benutzerhandbuch.

Themen

Ereignis zur Änderung des Bereitstellungsstatus

AWS IoT Greengrass löst ein Ereignis aus, wenn eine Bereitstellung in die folgenden Zustände übergeht: `FAILED`, `SUCCEEDED`, `COMPLETED`, `REJECTED` und `CANCELED`. Sie können eine EventBridge Regel erstellen, die für alle Zustandsübergänge oder Übergänge in von Ihnen angegebene Zustände ausgeführt wird. Wenn eine Bereitstellung in einen Zustand übergeht, der eine Regel initiiert, EventBridge ruft sie die in der Regel definierten Zielaktionen auf. Auf diese Weise können Sie Benachrichtigungen senden, Ereignisinformationen erfassen, Korrekturmaßnahmen ergreifen oder andere Ereignisse als Reaktion auf eine Statusänderung initiieren. Sie können beispielsweise Regeln für die folgenden Anwendungsfälle erstellen:

- Initiiert Operationen nach der Bereitstellung, z. B. das Herunterladen von Ressourcen und die Benachrichtigung des Personals.
- Senden Sie Benachrichtigungen bei erfolgreicher oder fehlgeschlagener Bereitstellung.
- Veröffentlichen Sie benutzerdefinierte Metriken zu Bereitstellungsereignissen.

Das Ereignis für eine Änderung des Bereitstellungsstatus verwendet das folgende Format:

```
{
    "version":"0",
    "id":" cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
    "detail-type":"Greengrass V2 Effective Deployment Status Change",
    "source":"aws.greengrass",
    "account":"123456789012",
    "region":"us-west-2",
    "time":"2018-03-22T00:38:11Z",
    "resources":["arn:aws:greengrass:us-
east-1:123456789012:coreDevices:MyGreengrassCore"],
    "detail":{
        "deploymentId": "4f38f1a7-3dd0-42a1-af48-EXAMPLE09681",
        "coreDeviceExecutionStatus": "FAILED|SUCCEEDED|COMPLETED|REJECTED|CANCELED",
        "statusDetails": {
          "errorStack": ["DEPLOYMENT_FAILURE", "ARTIFACT_DOWNLOAD_ERROR", "S3_ERROR",
 "S3_ACCESS_DENIED", "S3_HEAD_OBJECT_ACCESS_DENIED"],
          "errorTypes": ["DEPENDENCY_ERROR", "PERMISSION_ERROR"],
        },
        "reason": "S3_HEAD_OBJECT_ACCESS_DENIED: FAILED_NO_STATE_CHANGE: Failed to
 download artifact name: 's3://pentest27/nucleus/281/aws.greengrass.nucleus.zip' for
 component aws.greengrass.Nucleus-2.8.1, reason: S3 HeadObject returns 403 Access
 Denied. Ensure the IAM role associated with the core device has a policy granting
 s3:GetObject. null (Service: S3, Status Code: 403, Request ID: HR94ZNT2161DAR58,
 Extended Request ID: wTX4DDI+qigQt3uzwl9rlnQiYlBgwvPm/KJFWeFAn9t1mnGXTms/
luLCYANgq08RIH+x2H+hEKc=)"
     }
}
```

Sie können Regeln und Ereignisse erstellen, die Sie über den Status einer Bereitstellung auf dem Laufenden halten. Ein Ereignis wird ausgelöst, wenn eine Bereitstellung entweder als`FAILED`,, `SUCCEEDED` `COMPLETED``REJECTED`, oder abgeschlossen wird`CANCELED`. Wenn die Bereitstellung auf dem Kerngerät fehlgeschlagen ist, erhalten Sie eine ausführliche Antwort, in der erklärt wird,

warum die Bereitstellung fehlgeschlagen ist. Weitere Informationen zu Fehlercodes bei der Bereitstellung finden Sie unterDetaillierte Bereitstellungsfehlercodes.

Bereitstellungsstatus

- `FAILED`. Die Bereitstellung ist fehlgeschlagen.
- `SUCCEEDED`. Die Bereitstellung, die auf eine Dinggruppe abzielt, wurde erfolgreich abgeschlossen.
- `COMPLETED`. Die Bereitstellung, die auf eine Sache abzielte, wurde erfolgreich abgeschlossen.
- `REJECTED`. Der Einsatz wurde abgelehnt. Weitere Informationen finden Sie in dem entsprechenden `statusDetails` Feld.
- `CANCELED`. Die Bereitstellung wurde vom Benutzer abgebrochen.

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.

Eine vollständige Liste der Fehlercodes in `errorStacks` und `errorTypes` finden Sie unter Detaillierte Bereitstellungsfehlercodes undDetaillierte Komponenten-Statuscodes.

Ereignis zur Änderung des Komponentenstatus

Für AWS IoT Greengrass Versionen 2.12.2 und früher gibt Greengrass ein Ereignis aus, wenn eine Komponente in die folgenden Zustände übergeht: und. `ERRORED` `BROKEN` Für Greengrass Nucleus-Versionen 2.12.3 und höher gibt Greengrass ein Ereignis aus, wenn eine Komponente in die folgenden Zustände übergeht:`ERRORED`,, und. `BROKEN` `RUNNING` `FINISHED` Greengrass gibt auch ein Ereignis aus, wenn eine Bereitstellung abgeschlossen ist. Sie können eine EventBridge Regel erstellen, die für alle von Ihnen angegebenen Zustandsübergänge oder Übergänge zu Zuständen gilt. Wenn eine installierte Komponente in einen Zustand übergeht, der eine Regel EventBridge auslöst, werden die in der Regel definierten Zielaktionen aufgerufen. Auf diese Weise können Sie Benachrichtigungen senden, Ereignisinformationen erfassen, Korrekturmaßnahmen ergreifen oder andere Ereignisse als Reaktion auf eine Statusänderung initiieren.

Das Ereignis für eine Änderung des Komponentenstatus verwendet die folgenden Formate:

Greengrass nucleus v2.12.2 and earlier

<title>Status der Komponente: `ERRORED` oder `BROKEN`</title>

```
{
```

```
    "version":"0",
    "id":" cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
    "detail-type":"Greengrass V2 Installed Component Status Change",
    "source":"aws.greengrass",
    "account":"123456789012",
    "region":"us-west-2",
    "time":"2018-03-22T00:38:11Z",
    "resources":["arn:aws:greengrass:us-
east-1:123456789012:coreDevices:MyGreengrassCore"],
    "detail": {
       "components": [
          {
            "componentName": "MyComponent",
            "componentVersion": "1.0.0",
            "root": true,
            "lifecycleState": "ERRORED|BROKEN",
            "lifecycleStatusCodes": ["STARTUP_ERROR"],
            "lifecycleStateDetails": "An error occurred during startup. The startup
 script exited with code 1."
          }
        ]
    }
}
```

Greengrass nucleus v2.12.3 and later

<title>Status der Komponente: ERRORED oder BROKEN</title>

```
{
    "version":"0",
    "id":" cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
    "detail-type":"Greengrass V2 Installed Component Status Change",
    "source":"aws.greengrass",
    "account":"123456789012",
    "region":"us-west-2",
    "time":"2018-03-22T00:38:11Z",
    "resources":["arn:aws:greengrass:us-
east-1:123456789012:coreDevices:MyGreengrassCore"],
    "detail": {
       "components": [
          {
            "componentName": "MyComponent",
            "componentVersion": "1.0.0",
            "root": true,
```

```
            "lifecycleState": "ERRORED|BROKEN",
            "lifecycleStatusCodes": ["STARTUP_ERROR"],
            "lifecycleStateDetails": "An error occurred during startup. The startup
 script exited with code 1."
          }
        ]
    }
}
```

<title>Status der Komponente: RUNNING oder FINISHED</title>

```
{
    "version":"0",
    "id":" cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
    "detail-type":"Greengrass V2 Installed Component Status Change",
    "source":"aws.greengrass",
    "account":"123456789012",
    "region":"us-west-2",
    "time":"2018-03-22T00:38:11Z",
    "resources":["arn:aws:greengrass:us-
east-1:123456789012:coreDevices:MyGreengrassCore"],
    "detail": {
       "components": [
          {
            "componentName": "MyComponent",
            "componentVersion": "1.0.0",
            "root": true,
            "lifecycleState": "RUNNING|FINISHED",
            "lifecycleStateDetails": null
          }
        ]
    }
}
```

Sie können Regeln und Ereignisse erstellen, die Sie über den Status einer installierten Komponente auf dem Laufenden halten. Ein Ereignis wird ausgelöst, wenn sich der Status einer Komponente auf dem Gerät ändert. Sie erhalten eine ausführliche Antwort, in der erklärt wird, warum eine Komponente fehlerhaft oder defekt ist. Sie erhalten außerdem einen Statuscode, der einen Grund für den Fehler angibt. Weitere Informationen zu den Statuscodes von Komponenten finden Sie unterDetaillierte Komponenten-Statuscodes.

Voraussetzungen für die Erstellung von EventBridge Regeln

Bevor Sie eine EventBridge Regel für erstellen AWS IoT Greengrass, gehen Sie wie folgt vor:

- Machen Sie sich mit Ereignissen, Regeln und Zielen in vertraut EventBridge.
- Erstellen und konfigurieren Sie die Ziele, die durch Ihre EventBridge Regeln aufgerufen werden. Regeln können viele Arten von Zielen aufrufen, einschließlich:
  - Amazon-Simple-Notification-Service (Amazon-SNS)
  - AWS Lambda Funktionen
  - Amazon Kinesis Video Streams
  - Amazon-Simple-Queue-Service-(Amazon-SQS)-Warteschlangen

Weitere Informationen finden Sie unter Was ist Amazon EventBridge? und Erste Schritte mit Amazon EventBridge im EventBridge Amazon-Benutzerhandbuch.

Benachrichtigungen zum Gerätestatus konfigurieren (Konsole)

Gehen Sie wie folgt vor, um eine EventBridge Regel zu erstellen, die ein Amazon SNS SNS-Thema veröffentlicht, wenn sich der Bereitstellungsstatus für eine Gruppe ändert. Auf diese Weise können Webserver, E-Mail-Adressen und andere Themenabonnenten auf das Ereignis reagieren. Weitere Informationen finden Sie im EventBridge Amazon-Benutzerhandbuch unter Erstellen einer EventBridge Regel, die bei einem Ereignis von einer AWS Ressource ausgelöst wird.

1. Öffnen Sie die EventBridgeAmazon-Konsole.
2. Wählen Sie im Navigationsbereich Rules aus.
3. Wählen Sie Regel erstellen aus.
4. Geben Sie einen Namen und eine Beschreibung für die Regel ein.

   Eine Regel darf nicht denselben Namen wie eine andere Regel in derselben Region und auf demselben Event Bus haben.
5. Wählen Sie als Event bus (Event Bus) den Event Bus aus, den Sie dieser Regel zuordnen möchten. Wenn Sie möchten, dass diese Regel mit Ereignissen aus Ihrem eigenen Konto übereinstimmt, wählen Sie AWS -Standard-Event-Bus aus. Wenn ein AWS Service in Ihrem Konto ein Ereignis ausgibt, wird es immer an den Standard-Event-Bus Ihres Kontos weitergeleitet.

6. Bei Rule type (Regeltyp) wählen Sie Rule with an event pattern (Regel mit einem Ereignismuster) aus.
7. Wählen Sie Weiter aus.
8. Wählen Sie unter Event source (Ereignisquelle) AWS events (Ereignisse) aus.
9. Wählen Sie für Event-Pattern die Option AWS Services aus.
10. Wählen Sie für den AWS Service Greengrass.
11. Wählen Sie unter Ereignistyp eine der folgenden Optionen aus:

    - Wählen Sie für Bereitstellungsereignisse Greengrass V2 Effective Deployment Status Change.
    - Wählen Sie für Komponentenereignisse die Option Greengrass V2 Installed Component Status Change.
12. Wählen Sie Weiter aus.
13. Bei Target types (Zieltypen) wählen Sie AWS -Service aus.
14. Für Wählen Sie ein Ziel aus konfigurieren Sie Ihr Ziel. In diesem Beispiel wird ein Amazon SNS SNS-Thema verwendet, Sie können jedoch auch andere Zieltypen für das Senden von Benachrichtigungen konfigurieren.

    a. Wählen Sie in Target (Ziel) die Option SNS topic (SNS-Thema) aus.
    b. Wählen Sie unter Thema das Zielthema aus.
    c. Wählen Sie Weiter.
15. Wählen Sie Weiter.
16. Überprüfen Sie die Details der Regel und wählen Sie dann Create rule (Regel erstellen) aus.

Benachrichtigungen zum Gerätestatus (CLI) konfigurieren

Gehen Sie wie folgt vor, um eine EventBridge Regel zu erstellen, die ein Amazon SNS SNS-Thema veröffentlicht, wenn es ein Greengrass-Statusänderungsereignis gibt. Auf diese Weise können Webserver, E-Mail-Adressen und andere Themenabonnenten auf das Ereignis reagieren.

1. Erstellen Sie die -Regel.

    - Für Ereignisse zur Änderung des Bereitstellungsstatus.

```
aws events put-rule \
  --name TestRule \
```

```
  --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail-type\":
 [\"Greengrass V2 Effective Deployment Status Change\"]}"
```

- Für Ereignisse zur Änderung des Komponentenstatus.

```
aws events put-rule \
  --name TestRule \
  --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail-type\":
 [\"Greengrass V2 Installed Component Status Change\"]}"
```

Eigenschaften, die aus dem Muster weggelassen werden, werden ignoriert.

2. Fügen Sie das Thema als Regelziel hinzu.

   - Ersetzen Sie *topic-arn* durch den ARN Ihres Amazon SNS SNS-Themas.

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

> **Note**
>
> Damit Amazon EventBridge Ihr Zielthema aufrufen kann, müssen Sie Ihrem Thema eine ressourcenbasierte Richtlinie hinzufügen. Weitere Informationen finden Sie unter Amazon SNS SNS-Berechtigungen im EventBridge Amazon-Benutzerhandbuch.

Weitere Informationen finden Sie unter Ereignisse und Ereignismuster EventBridge im EventBridge Amazon-Benutzerhandbuch.

Benachrichtigungen zum Gerätestatus konfigurieren (AWS CloudFormation)

Verwenden Sie AWS CloudFormation Vorlagen, um EventBridge Regeln zu erstellen, die Benachrichtigungen über Statusänderungen für Ihre Greengrass-Gruppenbereitstellungen senden. Weitere Informationen finden Sie in der Referenz zum EventBridge Amazon-Ressourcentyp im AWS CloudFormation Benutzerhandbuch.

Weitere Informationen finden Sie auch unter

- Überprüfen Sie den Status der Gerätebereitstellung
- Was ist Amazon EventBridge? im EventBridge Amazon-Benutzerhandbuch

Überprüfen Sie den Status des Greengrass Core-Geräts

Greengrass-Core-Geräte melden den Status ihrer Softwarekomponenten an AWS IoT Greengrass. Sie können die Zustandsübersicht jedes Geräts überprüfen und Sie können den Status jeder Komponente auf jedem Gerät überprüfen.

Kerngeräte haben die folgenden Integritätsstatus:

- `HEALTHY`— Die AWS IoT Greengrass Core-Software und alle Komponenten laufen problemlos auf dem Core-Gerät.
- `UNHEALTHY`— Die AWS IoT Greengrass Core-Software oder eine Komponente befindet sich auf dem Core-Gerät in einem Fehlerstatus.

> **Note**
>
> AWS IoT Greengrass verlässt sich darauf, dass einzelne Geräte Statusaktualisierungen an die senden AWS Cloud. Wenn die AWS IoT Greengrass Core-Software nicht auf dem Gerät ausgeführt wird oder wenn das Gerät nicht mit dem verbunden ist AWS Cloud, entspricht der gemeldete Status dieses Geräts möglicherweise nicht dem aktuellen Status. Der Statuszeitstempel gibt an, wann der Gerätestatus zuletzt aktualisiert wurde.
> Core-Geräte senden Status-Updates zu den folgenden Zeiten:
>
> - Wenn die AWS IoT Greengrass Core-Software gestartet wird
> - Wenn das Kerngerät eine Bereitstellung von der erhält AWS Cloud
> - Bei Greengrass Nucleus 2.12.2 und früher sendet das Core-Gerät Status-Updates, wenn der Status einer Komponente auf dem Core-Gerät oder `ERRORED` `BROKEN`
> - Für Greengrass Nucleus 2.12.3 und höher sendet das Core-Gerät Status-Updates, wenn der Status einer Komponente auf dem Core-Gerät`ERRORED`,, `BROKEN` oder `RUNNING` `FINISHED`
> - In regelmäßigen Intervallen, die Sie konfigurieren können. Die Standardeinstellung ist 24 Stunden

Bei AWS IoT Greengrass Core v2.7.0 und höher sendet das Core-Gerät Statusaktualisierungen, wenn die lokale Bereitstellung und die Cloud-Bereitstellung erfolgt

Themen

Überprüfen Sie den Zustand eines Kerngeräts

Sie können den Status einzelner Kerngeräte überprüfen.

Um den Status eines Kerngeräts zu überprüfen (AWS CLI)

- Führen Sie den folgenden Befehl aus, um den Status eines Geräts abzurufen. *coreDeviceName*Ersetzen Sie ihn durch den Namen des abzufragenden Kerngeräts.

  ```
  aws greengrassv2 get-core-device --core-device-thing-name coreDeviceName
  ```

  Die Antwort enthält Informationen über das Kerngerät, einschließlich seines Status.

Überprüfen Sie den Zustand einer Kerngerätegruppe

Sie können den Status einer Gruppe von Kerngeräten (einer Dinggruppe) überprüfen.

Um den Status einer Gruppe von Geräten zu überprüfen (AWS CLI)

- Führen Sie den folgenden Befehl aus, um den Status mehrerer Kerngeräte abzurufen. Ersetzen Sie den ARN im Befehl durch den ARN der abzufragenden Dinggruppe.

  ```
  aws greengrassv2 list-core-devices --thing-group-arn "arn:aws:iot:region:account-id:thinggroup/thingGroupName"
  ```

  Die Antwort enthält die Liste der Kerngeräte in der Dinggruppe. Jeder Eintrag in der Liste enthält den Status des Kerngeräts.

Überprüfen Sie den Status der Komponenten des Kerngeräts

Sie können den Status, z. B. den Lebenszyklusstatus, der Softwarekomponenten auf einem Kerngerät überprüfen. Weitere Informationen zu den Lebenszyklusstatus von Komponenten finden Sie unterEntwickeln von AWS IoT Greengrass Komponenten.

Um den Status von Komponenten auf einem Kerngerät zu überprüfen (AWS CLI)

- Führen Sie den folgenden Befehl aus, um den Status der Komponenten auf einem Kerngerät abzurufen. *coreDeviceName*Ersetzen Sie ihn durch den Namen des abzufragenden Kerngeräts.

  ```
  aws greengrassv2 list-installed-components --core-device-thing-name coreDeviceName
  ```

  Die Antwort enthält die Liste der Komponenten, die auf dem Kerngerät ausgeführt werden. Jeder Eintrag in der Liste enthält den Lebenszyklusstatus der Komponente, einschließlich, wie aktuell der Status der Daten ist und wann das Greengrass-Core-Gerät zuletzt eine Nachricht mit einer bestimmten Komponente an die Cloud gesendet hat. Die Antwort wird auch die letzte Bereitstellungsquelle enthalten, die die Komponente auf das Greengrass-Core-Gerät gebracht hat.

  > **Note**
  >
  > Dieser Befehl ruft eine paginierte Liste der Komponenten ab, die auf einem Greengrass-Core-Gerät ausgeführt werden. Standardmäßig enthält diese Liste keine Komponenten, die als Abhängigkeiten von anderen Komponenten bereitgestellt werden. Sie können Abhängigkeiten in die Antwort einbeziehen, indem Sie den `topologyFilter` Parameter auf setzen`ALL`.

Ausführen von -AWS LambdaFunktionen

> **Note**
>
> AWS IoT Greengrass unterstützt diese Funktion derzeit nicht auf Windows-Core-Geräten.

Sie können AWS Lambda Funktionen als Komponenten importieren, die auf -AWS IoT GreengrassCore-Geräten ausgeführt werden. Sie können dies in den folgenden Fällen tun:

- Sie haben Anwendungscode in Lambda-Funktionen, die Sie auf -Core-Geräten bereitstellen möchten.
- Sie haben AWS IoT Greengrass V1-Anwendungen, die Sie auf -AWS IoT Greengrass V2Core-Geräten ausführen möchten. Weitere Informationen finden Sie unter Schritt 2: Erstellen und Bereitstellen von AWS IoT Greengrass V2 Komponenten zur Migration von AWS IoT Greengrass V1 Anwendungen.

Lambda-Funktionen umfassen Abhängigkeiten von den folgenden Komponenten. Sie müssen diese Komponenten nicht als Abhängigkeiten definieren, wenn Sie die Funktion importieren. Wenn Sie die Lambda-Funktionskomponente bereitstellen, enthält die Bereitstellung diese Abhängigkeiten von Lambda-Komponenten.

- Die Lambda-Launcher-Komponente (`aws.greengrass.LambdaLauncher`) verarbeitet Prozesse und Umgebungskonfigurationen.
- Die Lambda-Manager-Komponente (`aws.greengrass.LambdaManager`) kümmert sich um die Kommunikation und Skalierung zwischen Prozessen.
- Die Lambda-Laufzeitkomponente (`aws.greengrass.LambdaRuntimes`) stellt Artefakte für jede unterstützte Lambda-Laufzeit bereit.

Themen

- Importieren einer Lambda-Funktion als Komponente (AWS CLI)

Voraussetzungen

Ihre Core-Geräte und Lambda-Funktionen müssen die folgenden Anforderungen erfüllen, damit Sie die Funktionen auf der AWS IoT Greengrass Core-Software ausführen können:

- Ihr Core-Gerät muss die Anforderungen für die Ausführung von Lambda-Funktionen erfüllen. Wenn Sie möchten, dass das Core-Gerät containerisierte Lambda-Funktionen ausführt, muss das Gerät die Anforderungen dafür erfüllen. Weitere Informationen finden Sie unter Anforderungen an die Lambda-Funktion.
- Sie müssen die Programmiersprachen installieren, die die Lambda-Funktion auf Ihren -Core-Geräten verwendet.

> **Tip**
>
> Sie können eine Komponente erstellen, die die Programmiersprache installiert, und diese Komponente dann als Abhängigkeit Ihrer Lambda-Funktionskomponente angeben. Greengrass unterstützt alle von Lambda unterstützten Versionen von Python-, Node.js- und Java-Laufzeiten. Greengrass wendet keine zusätzlichen Einschränkungen auf veraltete Lambda-Laufzeitversionen an. Sie können Lambda-Funktionen ausführen, die diese veralteten Laufzeiten auf verwendenAWS IoT Greengrass, aber Sie können sie nicht in erstellenAWS Lambda. Weitere Informationen zur AWS IoT Greengrass Unterstützung von Lambda-Laufzeiten finden Sie unter Ausführen von -AWS LambdaFunktionen.

Konfigurieren des Lebenszyklus von Lambda-Funktionen

Der Lebenszyklus der Greengrass-Lambda-Funktion bestimmt, wann eine Funktion gestartet wird und wie sie Container erstellt und verwendet. Der Lebenszyklus bestimmt auch, wie die AWS IoT Greengrass Core-Software Variablen und Vorverarbeitungslogik beibehält, die sich außerhalb des Funktionshandlers befinden.

AWS IoT Greengrass unterstützt On-Demand-Lebenszyklen (Standard) und langlebige Lebenszyklen:

- On-Demand-Funktionen beginnen, wenn sie aufgerufen werden, und werden beendet, wenn keine Aufgaben mehr ausgeführt werden müssen. Jeder Aufruf der Funktion erstellt einen separaten Container, auch Sandbox genannt, um Aufrufe zu verarbeiten, es sei denn, ein vorhandener Container steht zur Wiederverwendung zur Verfügung. Jeder der Container kann Daten verarbeiten, die Sie an die Funktion senden.

  Mehrere Aufrufe einer On-Demand-Funktion können gleichzeitig ausgeführt werden.

  Variablen und Vorverarbeitungslogik, die Sie außerhalb des Funktionshandlers definieren, werden nicht beibehalten, wenn neue Container erstellt werden.
- Langlebige (oder angeheftete) Funktionen beginnen, wenn die AWS IoT Greengrass Core-Software gestartet wird und in einem einzigen Container ausgeführt wird. Derselbe Container verarbeitet alle Daten, die Sie an die Funktion senden.

  Mehrere Aufrufe werden in die Warteschlange gestellt, bis die AWS IoT Greengrass Core-Software frühere Aufrufe ausführt.

  Variablen und Vorverarbeitungslogik, die Sie außerhalb des Funktionshandlers definieren, werden für jeden Aufruf des Handlers beibehalten.

  Verwenden Sie langlebige Lambda-Funktionen, wenn Sie ohne anfängliche Eingabe mit der Arbeit beginnen müssen. Beispielsweise kann eine langlebige Funktion ein Machine-Learning-Modell laden und mit der Verarbeitung beginnen, damit es bereit ist, wenn die Funktion Gerätedaten empfängt.

  > **Note**
  >
  > Langlebige Funktionen haben Timeouts, die jedem Aufruf ihres Handlers zugeordnet sind. Wenn Sie Code aufrufen möchten, der auf unbestimmte Zeit ausgeführt wird, müssen Sie ihn außerhalb des Handlers starten. Stellen Sie sicher, dass außerhalb des Handlers kein blockierender Code vorhanden ist, der die Initialisierung der Funktion verhindern könnte. Diese Funktionen werden ausgeführt, es sei denn, die AWS IoT Greengrass -Core-Software wird angehalten, z. B. während einer Bereitstellung oder eines Neustarts. Diese Funktionen werden nicht ausgeführt, wenn die Funktion auf eine unerkannte Ausnahme stößt, ihre Speicherlimits überschreitet oder in einen Fehlerstatus übergeht, z. B. ein Handler-Timeout.

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

Konfigurieren der Containerisierung von Lambda-Funktionen

Standardmäßig werden Lambda-Funktionen innerhalb eines -AWS IoT GreengrassContainers ausgeführt. Greengrass-Container bieten Isolation zwischen Ihren Funktionen und dem Host. Diese Isolierung erhöht die Sicherheit sowohl für den Host als auch für die Funktionen im Container.

Wir empfehlen Ihnen, 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 darüber, wie Sie den Zugriff auf -Ressourcen einschränken.

In den folgenden Fällen können Sie eine Lambda-Funktion ohne Containerisierung ausführen:

- Sie möchten AWS IoT Greengrass auf einem Gerät ausführen, das den Containermodus nicht unterstützt. Ein Beispiel wäre, wenn Sie eine spezielle Linux-Distribution verwenden möchten oder eine frühere Kernelversion haben möchten, die veraltet 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 mit Pfaden, die zum Zeitpunkt der Bereitstellung nicht bestimmt werden können oder deren Pfade sich nach der Bereitstellung ändern können. Ein Beispiel für diese Ressource wäre ein Pluggable-Gerät.
- Sie haben eine frühere Anwendung, die als Prozess geschrieben wurde, und es treten Probleme auf, wenn Sie sie in einem Greengrass-Container 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

Containerisierung	Hinweise
	keinen Zugriff auf den bereitgestellten Code anderer Lambda-Funktionen, auch wenn sie mit derselben Systemgruppe ausgeführt werden. Mit anderen Worten, Ihre Lambda-Funktionen werden mit erhöhter Isolation ausgeführt. • Da die AWS IoT Greengrass Core-Software alle untergeordneten Prozesse im selben Container wie die Lambda-Funktion ausführt, werden die untergeordneten Prozesse beendet, wenn die Lambda-Funktion beendet wird.

Kein Container	• Die folgenden Funktionen sind für nicht containerisierte Lambda-Funktionen nicht verfügbar:
	  • Speicherlimits für Lambda-Funktionen.
	  • Lokale Geräte- und Volume-Ressourcen Sie müssen auf diese Ressourcen über ihre Dateipfade auf dem Core-Gerä t zugreifen und nicht als Lambda-Fu nktionsressourcen.
	• Wenn Ihre nicht containerisierte Lambda-Fu nktion auf eine Machine-Learning-Ressource zugreift, müssen Sie einen Ressource nbesitzer identifizieren und Zugriffsberechtigu ngen für die Ressource festlegen, nicht für die Lambda-Funktion.
	• Nicht containerisierte Lambda-Funktionen haben schreibgeschützten Zugriff auf den bereitgestellten Code anderer Lambda-Fu nktionen, die mit derselben Systemgruppe ausgeführt werden.

Wenn Sie die Containerisierung für eine Lambda-Funktion bei der Bereitstellung ändern, funktioniert die Funktion möglicherweise nicht wie erwartet. Wenn die Lambda-Funktion lokale Ressourcen verwendet, die mit der neuen Containerisierungseinstellung 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 die Speicherlimits der Funktion verworfen. Sie müssen direkt auf das Dateisystem zugreifen, anstatt verknüpfte lokale Ressourcen zu verwenden. Sie müssen alle angefügten Ressourcen entfernen, bevor Sie die Lambda-Funktion bereitstellen.
- 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 das Standardspeicherlimit von 16 MB akzeptieren. Sie können diese Einstellungen für jede Lambda-Funktion konfigurieren, wenn Sie sie bereitstellen.

Um die Containerisierungseinstellungen für eine Lambda-Funktionskomponente zu ändern, legen Sie den Wert des `containerMode` Konfigurationsparameters bei der Bereitstellung der Komponente auf eine der folgenden Optionen fest.

- `NoContainer` – Die Komponente wird nicht in einer isolierten Laufzeitumgebung ausgeführt.
- `GreengrassContainer` – Die Komponente wird in einer isolierten Laufzeitumgebung innerhalb des AWS IoT Greengrass Containers ausgeführt.

Weitere Informationen zum Bereitstellen und Konfigurieren von Komponenten finden Sie unter Bereitstellen von AWS IoT Greengrass Komponenten auf Geräten und Komponentenkonfigurationen aktualisieren.

Importieren einer Lambda-Funktion als Komponente (Konsole)

Wenn Sie die AWS IoT Greengrass Konsole verwenden, um eine Lambda-Funktionskomponente zu erstellen, importieren Sie eine vorhandene AWS Lambda Funktion und konfigurieren sie dann so, dass eine Komponente erstellt wird, die auf Ihrem Greengrass-Gerät ausgeführt wird.

Bevor Sie beginnen, überprüfen Sie die Anforderungen zum Ausführen von Lambda-Funktionen auf Greengrass-Geräten.

Aufgaben

Schritt 1: Auswählen einer zu importierenden Lambda-Funktion

1. Wählen Sie im Navigationsmenü der AWS IoT GreengrassKonsole Komponenten aus.
2. Wählen Sie auf der Seite Komponenten die Option Komponente erstellen aus.
3. Wählen Sie auf der Seite Komponente erstellen unter Komponenteninformationen die Option Lambda-Funktion importieren aus.
4. Suchen Sie in der Lambda-Funktion nach der Lambda-Funktion, die Sie importieren möchten, und wählen Sie sie aus.

   AWS IoT Greengrass erstellt die Komponente mit dem Namen der Lambda-Funktion.
5. Wählen Sie in Version der Lambda-Funktion die zu importierende Version aus. Sie können keine Lambda-Aliase wie auswählen`$LATEST`.

   AWS IoT Greengrass erstellt die Komponente mit der Version der Lambda-Funktion als gültige semantische Version. Wenn Ihre Funktionsversion beispielsweise `3` lautet, wird die Komponentenversion `3.0.0`.

Schritt 2: Konfigurieren von Lambda-Funktionsparametern

Konfigurieren Sie auf der Seite Komponente erstellen unter Konfiguration der Lambda-Funktion die folgenden Parameter, die zum Ausführen der Lambda-Funktion verwendet werden sollen.

1. (Optional) Fügen Sie die Liste der Ereignisquellen hinzu, für die die Lambda-Funktion Arbeitsnachrichten abonniert. Sie können Ereignisquellen angeben, um diese Funktion für lokale Veröffentlichungs-/Abonnementnachrichten und AWS IoT Core MQTT-Nachrichten zu abonnieren. Die Lambda-Funktion wird aufgerufen, wenn sie eine Nachricht von einer Ereignisquelle erhält.

> **Note**
>
> Um diese Funktion für Nachrichten von anderen Lambda-Funktionen oder -Komponenten zu abonnieren, stellen Sie die Legacy-Abonnement-Routerkomponente bereit, wenn Sie diese Lambda-Funktionskomponente bereitstellen. Wenn Sie die Legacy-Abonnement-Routerkomponente bereitstellen, geben Sie die Abonnements an, die die Lambda-Funktion verwendet.

Gehen Sie unter Ereignisquellen wie folgt vor, um eine Ereignisquelle hinzuzufügen:

a. Geben Sie für jede Ereignisquelle, die Sie hinzufügen, die folgenden Optionen an:

- Thema – Das Thema, das Nachrichten abonniert werden soll.
- Typ – Der Typ der Ereignisquelle. Wählen Sie aus den folgenden Optionen aus:
  - Lokales Veröffentlichen/Abonnieren – Abonnieren Sie lokale Veröffentlichungs-/Abonnementnachrichten.

    Wenn Sie den Greengrass-Kern v2.6.0 oder höher und Lambda Manager v2.2.5 oder höher verwenden, können Sie MQTT-Themen-Platzhalter (+ und #) im Thema verwenden, wenn Sie diesen Typ angeben.
  - AWS IoT Core MQTT – Abonnieren Sie AWS IoT Core MQTT-Nachrichten.

    Sie können MQTT-Themen-Platzhalter (+ und #) im Thema verwenden, wenn Sie diesen Typ angeben.

b. Um eine weitere Ereignisquelle hinzuzufügen, wählen Sie Ereignisquelle hinzufügen und wiederholen Sie den vorherigen Schritt. Um eine Ereignisquelle zu entfernen, wählen Sie Entfernen neben der Ereignisquelle aus, die Sie entfernen möchten.

2. Geben Sie für Timeout (Sekunden) die maximale Zeit in Sekunden ein, die eine nicht angeheftete Lambda-Funktion ausgeführt werden kann, bevor eine Zeitüberschreitung auftritt. Der Standardwert ist 3 Sekunden.
3. Wählen Sie für Angeheftet aus, ob die Lambda-Funktionskomponente angeheftet ist. Der Standardwert ist True .

- Eine angeheftete (oder langlebige) Lambda-Funktion startet, wenn AWS IoT Greengrass startet und weiter in einem eigenen Container ausgeführt wird.

- Eine nicht angeheftete (oder On-Demand) Lambda-Funktion startet nur, wenn sie ein Arbeitselement empfängt, und wird beendet, nachdem sie für eine bestimmte maximale Leerlaufzeit inaktiv bleibt. Wenn die Funktion mehrere Arbeitselemente enthält, erstellt die AWS IoT Greengrass Core-Software mehrere Instances der Funktion.

4. (Optional) Legen Sie unter Zusätzliche Parameter die folgenden Lambda-Funktionsparameter fest.

   - Status-Timeout (Sekunden) – Das Intervall in Sekunden, in dem die Lambda-Funktionskomponente Statusaktualisierungen an die Lambda-Manager-Komponente sendet. Dieser Parameter gilt nur für angeheftete Funktionen. Standardmäßig ist ein Zeitraum von 60 Sekunden festgelegt.
   - Maximale Warteschlangengröße – Die maximale Größe der Nachrichtenwarteschlange für die Lambda-Funktionskomponente. Die AWS IoT Greengrass Core-Software speichert Nachrichten in einer FIFO-Warteschlange (First-In, First-Out), bis sie die Lambda-Funktion ausführen kann, um jede Nachricht zu verarbeiten. Der Standardwert ist 1 000 Nachrichten.
   - Maximale Anzahl von Instances – Die maximale Anzahl von Instances, die eine nicht angeheftete Lambda-Funktion gleichzeitig ausführen kann. Der Standardwert ist 100 Instances.
   - Maximale Leerlaufzeit (Sekunden) – Die maximale Zeit in Sekunden, die eine nicht angeheftete Lambda-Funktion im Leerlauf verbringen kann, bevor die AWS IoT Greengrass Core-Software ihren Prozess beendet. Standardmäßig ist ein Zeitraum von 60 Sekunden festgelegt.
   - Codierungstyp – Der Typ der Nutzlast, die die Lambda-Funktion unterstützt. Wählen Sie aus den folgenden Optionen aus:
     - JSON
     - Binary

     Der Standardwert ist JSON.

5. (Optional) Geben Sie die Liste der Befehlszeilenargumente an, die bei der Ausführung an die Lambda-Funktion übergeben werden sollen.

   a. Wählen Sie unter Zusätzliche Parameter die Option Argument hinzufügen aus.

   b. Geben Sie für jedes Argument, das Sie hinzufügen, das Argument ein, das Sie an die Funktion übergeben möchten.

c. Um ein Argument zu entfernen, wählen Sie neben dem Argument, das Sie entfernen möchten, Entfernen aus.

6. (Optional) Geben Sie die Umgebungsvariablen an, die der Lambda-Funktion bei der Ausführung zur Verfügung stehen. Mit Umgebungsvariablen können Sie Konfigurationseinstellungen speichern und aktualisieren, ohne den Funktionscode ändern zu müssen.

   a. Wählen Sie unter Zusätzliche Parameter, Umgebungsvariablen die Option Umgebungsvariable hinzufügen aus.
   b. Geben Sie für jede Umgebungsvariable, die Sie hinzufügen, die folgenden Optionen an:
      - Schlüssel – Der Variablenname.
      - Wert – Der Standardwert für diese Variable.
   c. Um eine Umgebungsvariable zu entfernen, wählen Sie Entfernen neben der Umgebungsvariablen aus, die Sie entfernen möchten.

Schritt 3: (Optional) Geben Sie unterstützte Plattformen für die Lambda-Funktion an

Alle -Core-Geräte haben Attribute für Betriebssystem und Architektur. Wenn Sie die Lambda-Funktionskomponente bereitstellen, vergleicht die AWS IoT Greengrass-Core-Software die von Ihnen angegebenen Plattformwerte mit den Plattformattributen auf dem Core-Gerät, um festzustellen, ob die Lambda-Funktion auf diesem Gerät unterstützt wird.

> **Note**
>
> Sie können auch benutzerdefinierte Plattformattribute angeben, wenn Sie die Greengrass-Kernkomponente auf einem Core-Gerät bereitstellen. Weitere Informationen finden Sie unter dem Plattformüberschreibungsparameter der Greengrass-Kernkomponente .

Gehen Sie unter Lambda-Funktionskonfiguration, Zusätzliche Parameter, Plattformen wie folgt vor, um die Plattformen anzugeben, die diese Lambda-Funktion unterstützt.

1. Geben Sie für jede Plattform die folgenden Optionen an:

   - Betriebssystem – Der Name des Betriebssystems für die Plattform. Derzeit wird als einziger Wert unterstütz `linux`.

- Architektur – Die Prozessorarchitektur für die Plattform. Unterstützte Werte sind:
  - `amd64`
  - `arm`
  - `aarch64`
  - `x86`

2. Um eine weitere Plattform hinzuzufügen, wählen Sie Plattform hinzufügen und wiederholen Sie den vorherigen Schritt. Um eine unterstützte Plattform zu entfernen, wählen Sie neben der Plattform, die Sie entfernen möchten, Entfernen aus.

Schritt 4: (Optional) Angeben von Komponentenabhängigkeiten für die Lambda-Funktion

Komponentenabhängigkeiten identifizieren zusätzliche von AWSbereitgestellte Komponenten oder benutzerdefinierte Komponenten, die Ihre Funktion verwendet. Wenn Sie die Lambda-Funktionskomponente bereitstellen, enthält die Bereitstellung diese Abhängigkeiten für die Ausführung Ihrer Funktion.

> **Important**
>
> Um eine Lambda-Funktion zu importieren, die Sie für die Ausführung auf AWS IoT Greengrass V1 erstellt haben, müssen Sie individuelle Komponentenabhängigkeiten für die Funktionen definieren, die Ihre Funktion verwendet, z. B. Secrets, lokale Schatten und Stream-Manager. Definieren Sie diese Komponenten als harte Abhängigkeiten, sodass Ihre Lambda-Funktionskomponente neu gestartet wird, wenn sich die Abhängigkeit ändert. Weitere Informationen finden Sie unter Importieren von V1-Lambda-Funktionen.

Führen Sie unter Lambda-Funktionskonfiguration, Zusätzliche Parameter, Komponentenabhängigkeiten die folgenden Schritte aus, um die Komponentenabhängigkeiten für Ihre Lambda-Funktion anzugeben.

1. Wählen Sie Abhängigkeit hinzufügen aus.
2. Geben Sie für jede Komponentenabhängigkeit, die Sie hinzufügen, die folgenden Optionen an:

   - Komponentenname – Der Komponentenname. Geben Sie beispielsweise ein, **`aws.greengrass.StreamManager`** um die Stream-Manager-Komponente einzuschließen.

- Versionsanforderung – Die semantische Versionseinschränkung im npm-Stil, die die kompatiblen Versionen dieser Komponentenabhängigkeit identifiziert. Sie können eine einzelne Version oder einen Bereich von Versionen angeben. Geben Sie beispielsweise ein, **`^1.0.0`** um anzugeben, dass diese Lambda-Funktion von einer beliebigen Version in der ersten Hauptversion der Stream-Manager-Komponente abhängt. Weitere Informationen zu semantischen Versionseinschränkungen finden Sie im npm-Semver-Rechner .
- Typ – Der Typ der Abhängigkeit. Wählen Sie aus den folgenden Optionen aus:
  - Hard – Die Lambda-Funktionskomponente wird neu gestartet, wenn sich die Abhängigkeit ändert. Dies ist die Standardauswahl.
  - Soft – Die Lambda-Funktionskomponente wird nicht neu gestartet, wenn sich die Abhängigkeit ändert.

3. Um eine Komponentenabhängigkeit zu entfernen, wählen Sie Entfernen neben der Komponentenabhängigkeit

Schritt 5: (Optional) Ausführen der Lambda-Funktion in einem Container

Standardmäßig werden Lambda-Funktionen in einer isolierten Laufzeitumgebung innerhalb der AWS IoT Greengrass Core-Software ausgeführt. Sie können die Lambda-Funktion auch als Prozess ohne Isolierung ausführen (d. h. im Modus Kein Container).

Wählen Sie unter Linux-Prozesskonfiguration für Isolationsmodus eine der folgenden Optionen aus, um die Containerisierung für Ihre Lambda-Funktion auszuwählen:

- Greengrass-Container – Die Lambda-Funktion wird in einem Container ausgeführt. Dies ist die Standardauswahl.
- Kein Container – Die Lambda-Funktion wird als Prozess ohne Isolierung ausgeführt.

Wenn Sie die Lambda-Funktion in einem Container ausführen, führen Sie die folgenden Schritte aus, um die Prozesskonfiguration für die Lambda-Funktion zu konfigurieren.

1. Konfigurieren Sie die Speichermenge und die Systemressourcen, wie Volumes und Geräte, um sie dem Container zur Verfügung zu stellen.

   Gehen Sie unter Containerparameter wie folgt vor.

   a. Geben Sie für Speichergröße die Speichergröße ein, die Sie dem Container zuweisen möchten. Sie können die Speichergröße in MB oder kB angeben.

b. Wählen Sie unter Schreibgeschützter Sys-Ordner aus, ob der Container Informationen aus dem `/sys` Geräteordner lesen kann oder nicht. Der Standardwert ist False .

2. (Optional) Konfigurieren Sie die lokalen Volumes, auf die die containerisierte Lambda-Funktion zugreifen kann. Wenn Sie ein Volume definieren, mountet die AWS IoT Greengrass Core-Software die Quelldateien in das Ziel innerhalb des Containers.

   a. Wählen Sie unter Volumes die Option Volume hinzufügen aus.

   b. Geben Sie für jedes Volume, das Sie hinzufügen, die folgenden Optionen an:

      - Physisches Volume – Der Pfad zum Quellordner auf dem Core-Gerät.
      - Logisches Volume – Der Pfad zum Zielordner im Container.
      - Berechtigung – (Optional) Die Berechtigung für den Zugriff auf den Quellordner aus dem Container. Wählen Sie aus den folgenden Optionen aus:
        - Schreibgeschützt – Die Lambda-Funktion hat schreibgeschützten Zugriff auf den Quellordner. Dies ist die Standardauswahl.
        - Lese-/Schreibzugriff – Die Lambda-Funktion hat Lese-/Schreibzugriff auf den Quellordner.
      - Gruppenbesitzer hinzufügen – (Optional) Gibt an, ob die Systemgruppe, die die Lambda-Funktionskomponente ausführt, als Eigentümer des Quellordners hinzugefügt werden soll oder nicht. Der Standardwert ist False .

   c. Um ein Volume zu entfernen, wählen Sie Entfernen neben dem Volume aus, das Sie entfernen möchten.

3. (Optional) Konfigurieren Sie die lokalen Systemgeräte, auf die die containerisierte Lambda-Funktion zugreifen kann.

   a. Wählen Sie unter Geräte die Option Gerät hinzufügen aus.

   b. Geben Sie für jedes Gerät, das Sie hinzufügen, die folgenden Optionen an:

      - Mount-Pfad – Der Pfad zum Systemgerät auf dem Core-Gerät.
      - Berechtigung – (Optional) Die Berechtigung für den Zugriff auf das Systemgerät vom Container aus. Wählen Sie aus den folgenden Optionen aus:
        - Schreibgeschützt – Die Lambda-Funktion hat schreibgeschützten Zugriff auf das Systemgerät. Dies ist die Standardauswahl.
        - Lese-/Schreibzugriff – Die Lambda-Funktion hat Lese-/Schreibzugriff auf den Quellordner.

- Gruppenbesitzer hinzufügen – (Optional) Gibt an, ob die Systemgruppe hinzugefügt werden soll, die die Lambda-Funktionskomponente als Eigentümer des Systemgeräts ausführt. Der Standardwert ist False .

Schritt 6: Erstellen der Lambda-Funktionskomponente

Nachdem Sie die Einstellungen für Ihre Lambda-Funktionskomponente konfiguriert haben, wählen Sie Erstellen, um die Erstellung der neuen Komponente abzuschließen.

Um die Lambda-Funktion auf Ihrem Core-Gerät auszuführen, können Sie dann die neue Komponente auf Ihren Core-Geräten bereitstellen. Weitere Informationen finden Sie unter Bereitstellen von AWS IoT Greengrass Komponenten auf Geräten.

Importieren einer Lambda-Funktion als Komponente (AWS CLI)

Verwenden Sie die -CreateComponentVersionOperation, um Komponenten aus Lambda-Funktionen zu erstellen. Wenn Sie diesen Vorgang aufrufen, geben Sie an, `lambdaFunction` um eine Lambda-Funktion zu importieren.

Aufgaben

- Schritt 1: Definieren der Lambda-Funktionskonfiguration
- Schritt 2: Erstellen der Lambda-Funktionskomponente

Schritt 1: Definieren der Lambda-Funktionskonfiguration

1. Erstellen Sie eine Datei mit dem Namen `lambda-function-component.json`und kopieren Sie dann das folgende JSON-Objekt in die Datei . Ersetzen Sie durch `lambdaArn` den ARN der zu importierenden Lambda-Funktion.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1"
  }
}
```

> **Important**
>
> Sie müssen einen ARN angeben, der die Version der zu importierenden Funktion enthält. Sie können keine Versions-Aliase wie `$LATEST` verwenden.

2. (Optional) Geben Sie den Namen (`componentName`) der Komponente an. Wenn Sie diesen Parameter weglassen, AWS IoT Greengrass erstellt die Komponente mit dem Namen der Lambda-Funktion.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda"
  }
}
```

3. (Optional) Geben Sie die Version (`componentVersion`) für die Komponente an. Wenn Sie diesen Parameter weglassen, AWS IoT Greengrass erstellt die Komponente mit der Version der Lambda-Funktion als gültige semantische Version. Wenn Ihre Funktionsversion beispielsweise `3` lautet, wird die Komponentenversion `3.0.0`.

> **Note**
>
> Jede Komponentenversion, die Sie hochladen, muss eindeutig sein. Stellen Sie sicher, dass Sie die richtige Komponentenversion hochladen, da Sie sie nach dem Hochladen nicht mehr bearbeiten können.
> AWS IoT Greengrass verwendet semantische Versionen für -Komponenten. Semantische Versionen folgen einem größeren Patch-Nummernsystem. Die -Version `1.0.0` stellt beispielsweise die erste Hauptversion für eine Komponente dar. Weitere Informationen finden Sie in der semantischen Versionsspezifikation .

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0"
  }
```

```
}
```

4. (Optional) Geben Sie die Plattformen an, die diese Lambda-Funktion unterstützt. Jede Plattform enthält eine Zuordnung von Attributen, die eine Plattform identifizieren. Alle -Core-Geräte haben Attribute für Betriebssystem (`os`) und Architektur (`architecture`). Die AWS IoT Greengrass -Core-Software fügt möglicherweise weitere Plattformattribute hinzu. Sie können auch benutzerdefinierte Plattformattribute angeben, wenn Sie die Greengrass-Kernkomponente auf einem Core-Gerät bereitstellen. Gehen Sie wie folgt vor:

   a. Fügen Sie der Lambda-Funktion in eine Liste von Plattformen (`componentPlatforms`) hinzu`lambda-function-component.json`.

   ```
   {
     "lambdaFunction": {
       "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
       "componentName": "com.example.HelloWorldLambda",
       "componentVersion": "1.0.0",
       "componentPlatforms": [

       ]
     }
   }
   ```

   b. Fügen Sie der Liste jede unterstützte Plattform hinzu. Jede Plattform verfügt über eine Anzeige`name`, um sie zu identifizieren, und eine Zuordnung von Attributen. Das folgende Beispiel gibt an, dass diese Funktion x86-Geräte unterstützt, auf denen Linux ausgeführt wird.

   ```
   {
     "name": "Linux x86",
     "attributes": {
       "os": "linux",
       "architecture": "x86"
     }
   }
   ```

   Ihr `lambda-function-component.json` könnte ein Dokument ähnlich dem folgenden Beispiel enthalten.

   ```
   {
     "lambdaFunction": {
   ```

```
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ]
  }
}
```

5. (Optional) Geben Sie die Komponentenabhängigkeiten für Ihre Lambda-Funktion an. Wenn Sie die Lambda-Funktionskomponente bereitstellen, enthält die Bereitstellung diese Abhängigkeiten für die Ausführung Ihrer Funktion.

   > **Important**
   >
   > Um eine Lambda-Funktion zu importieren, die Sie für die Ausführung auf AWS IoT Greengrass V1 erstellt haben, müssen Sie individuelle Komponentenabhängigkeiten für die Funktionen definieren, die Ihre Funktion verwendet, z. B. Secrets, lokale Schatten und Stream-Manager. Definieren Sie diese Komponenten als harte Abhängigkeiten, sodass Ihre Lambda-Funktionskomponente neu gestartet wird, wenn sich die Abhängigkeit ändert. Weitere Informationen finden Sie unter Importieren von V1-Lambda-Funktionen.

   Gehen Sie wie folgt vor:

   a. Fügen Sie der Lambda-Funktion in eine Zuordnung von Komponentenabhängigkeiten (`componentDependencies`) hinzu`lambda-function-component.json`.

      ```
      {
        "lambdaFunction": {
          "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
          "componentName": "com.example.HelloWorldLambda",
          "componentVersion": "1.0.0",
          "componentPlatforms": [
      ```

```
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {

    }
  }
}
```

b. Fügen Sie der Zuordnung jede Komponentenabhängigkeit hinzu. Geben Sie den Komponentennamen als Schlüssel und ein Objekt mit den folgenden Parametern an:

- `versionRequirement` – Die semantische Versionseinschränkung im npm-Stil, die die kompatiblen Versionen der Komponentenabhängigkeit identifiziert. Sie können eine einzelne Version oder einen Bereich von Versionen angeben. Weitere Informationen zu semantischen Versionseinschränkungen finden Sie im npm-Semver-Rechner .
- `dependencyType` – (Optional) Der Typ der Abhängigkeit. Wählen Sie eine der folgenden Optionen aus:
  - `SOFT` – Die Lambda-Funktionskomponente wird nicht neu gestartet, wenn sich die Abhängigkeit ändert.
  - `HARD` – Die Lambda-Funktionskomponente wird neu gestartet, wenn die Abhängigkeit den Status ändert.

  Der Standardwert ist `HARD`.

Das folgende Beispiel gibt an, dass diese Lambda-Funktion von jeder Version in der ersten Hauptversion der Stream-Manager-Komponente abhängt. Die Lambda-Funktionskomponente wird neu gestartet, wenn der Stream-Manager neu gestartet oder aktualisiert wird.

```
{
  "aws.greengrass.StreamManager": {
    "versionRequirement": "^1.0.0",
    "dependencyType": "HARD"
```

```
  }
}
```

Ihr `lambda-function-component.json` könnte ein Dokument ähnlich dem folgenden Beispiel enthalten.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    }
  }
}
```

6. (Optional) Konfigurieren Sie die Lambda-Funktionsparameter, die zum Ausführen der Funktion verwendet werden sollen. Sie können Optionen wie Umgebungsvariablen, Nachrichtenereignisquellen, Timeouts und Containereinstellungen konfigurieren. Gehen Sie wie folgt vor:

   a. Fügen Sie das Lambda-Parameterobjekt (`componentLambdaParameters`) zur Lambda-Funktion in hinzu`lambda-function-component.json`.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
```

```
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {

    }
  }
}
```

b. (Optional) Geben Sie die Ereignisquellen an, für die die Lambda-Funktion Arbeitsnachrichten abonniert. Sie können Ereignisquellen angeben, um diese Funktion für lokale Veröffentlichungs-/Abonnementnachrichten und AWS IoT Core MQTT-Nachrichten zu abonnieren. Die Lambda-Funktion wird aufgerufen, wenn sie eine Nachricht von einer Ereignisquelle erhält.

> **Note**
>
> Um diese Funktion für Nachrichten von anderen Lambda-Funktionen oder -Komponenten zu abonnieren, stellen Sie die Legacy-Abonnement-Routerkomponente bereit, wenn Sie diese Lambda-Funktionskomponente bereitstellen. Wenn Sie die Legacy-Abonnement-Routerkomponente bereitstellen, geben Sie die Abonnements an, die die Lambda-Funktion verwendet.

Gehen Sie wie folgt vor:

i. Fügen Sie die Liste der Ereignisquellen (`eventSources`) zu den Lambda-Funktionsparametern hinzu.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [

      ]
    }
  }
}
```

ii. Fügen Sie der Liste jede Ereignisquelle hinzu. Jede Ereignisquelle hat die folgenden Parameter:

- `topic` – Das Thema zum Abonnieren von Nachrichten.
- `type` – Der Typ der Ereignisquelle. Wählen Sie aus den folgenden Optionen aus:
  - `PUB_SUB` — Abonnieren Sie lokale Veröffentlichen/Abonnement-Nachrichten.

    Wenn Sie den Greengrass-Kern v2.6.0 oder höher und Lambda Manager v2.2.5 oder höher verwenden, können Sie MQTT-Themen-Platzhalter (+ und #) in verwenden, `topic` wenn Sie diesen Typ angeben.
  - `IOT_CORE` – Abonnieren Sie AWS IoT Core-MQTT-Nachrichten.

Sie können MQTT-Themen-Platzhalter (+ und #) in der verwenden`topic`, wenn Sie diesen Typ angeben.

Im folgenden Beispiel wird AWS IoT Core MQTT für Themen abonniert, die dem `hello/world/+` Themenfilter entsprechen.

```
{
  "topic": "hello/world/+",
  "type": "IOT_CORE"
}
```

Ihr `lambda-function-component.json` könnte dem folgenden Beispiel ähneln.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ]
```

```
    }
  }
}
```

c. (Optional) Geben Sie einen der folgenden Parameter im Objekt Lambda-Funktionsparameter an:

- `environmentVariables` – Die Zuordnung von Umgebungsvariablen, die der Lambda-Funktion bei der Ausführung zur Verfügung stehen.
- `execArgs` – Die Liste der Argumente, die bei der Ausführung an die Lambda-Funktion übergeben werden sollen.
- `inputPayloadEncodingType` – Die Art der Nutzlast, die die Lambda-Funktion unterstützt. Wählen Sie aus den folgenden Optionen aus:
  - `json`
  - `binary`

  Standard: `json`
- `pinned` – Gibt an, ob die Lambda-Funktion angeheftet ist oder nicht. Der Standardwert ist `true`.
  - Eine angeheftete (oder langlebige) Lambda-Funktion startet, wenn AWS IoT Greengrass startet und weiter in einem eigenen Container ausgeführt wird.
  - Eine nicht angeheftete (oder On-Demand) Lambda-Funktion startet nur, wenn sie ein Arbeitselement empfängt, und wird beendet, nachdem sie für eine bestimmte maximale Leerlaufzeit inaktiv bleibt. Wenn die Funktion mehrere Arbeitselemente enthält, erstellt die AWS IoT Greengrass Core-Software mehrere Instances der Funktion.

  Verwenden Sie `maxIdleTimeInSeconds`, um die maximale Leerlaufzeit für Ihre Funktion festzulegen.
- `timeoutInSeconds` – Die maximale Zeit in Sekunden, die die Lambda-Funktion ausgeführt werden kann, bevor eine Zeitüberschreitung auftritt. Der Standardwert ist 3 Sekunden.
- `statusTimeoutInSeconds` – Das Intervall in Sekunden, in dem die Lambda-Funktionskomponente Statusaktualisierungen an die Lambda-Manager-Komponente sendet. Dieser Parameter gilt nur für angeheftete Funktionen. Standardmäßig ist ein Zeitraum von 60 Sekunden festgelegt.

- `maxIdleTimeInSeconds` – Die maximale Zeit in Sekunden, die eine nicht angeheftete Lambda-Funktion im Leerlauf verbringen kann, bevor die AWS IoT Greengrass Core-Software ihren Prozess beendet. Standardmäßig ist ein Zeitraum von 60 Sekunden festgelegt.
- `maxInstancesCount` – Die maximale Anzahl von Instances, die eine nicht angeheftete Lambda-Funktion gleichzeitig ausführen kann. Der Standardwert ist 100 Instances.
- `maxQueueSize` – Die maximale Größe der Nachrichtenwarteschlange für die Lambda-Funktionskomponente. Die AWS IoT Greengrass Core-Software speichert Nachrichten in einer FIFO-Warteschlange (first-in-first-out), bis sie die Lambda-Funktion ausführen kann, um jede Nachricht zu verarbeiten. Der Standardwert ist 1 000 Nachrichten.

Ihr `lambda-function-component.json` könnte ein Dokument ähnlich dem folgenden Beispiel enthalten.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
```

```
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500
    }
  }
}
```

d. (Optional) Konfigurieren Sie die Container-Einstellungen für die Lambda-Funktion. Standardmäßig werden Lambda-Funktionen in einer isolierten Laufzeitumgebung innerhalb der AWS IoT Greengrass Core-Software ausgeführt. Sie können die Lambda-Funktion auch ohne Isolierung als Prozess ausführen. Wenn Sie die Lambda-Funktion in einem Container ausführen, konfigurieren Sie die Speichergröße des Containers und die Systemressourcen, die für die Lambda-Funktion verfügbar sind. Gehen Sie wie folgt vor:

   i. Fügen Sie das Linux-Prozessparameterobjekt (`linuxProcessParams`) zum Lambda-Parameterobjekt in hinzu`lambda-function-component.json`.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
```

```
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {

      }
    }
  }
}
```

ii. (Optional) Geben Sie an, ob die Lambda-Funktion in einem Container ausgeführt wird oder nicht. Fügen Sie dem Prozessparameterobjekt den `isolationMode` Parameter hinzu und wählen Sie eine der folgenden Optionen aus:

- `GreengrassContainer` – Die Lambda-Funktion wird in einem Container ausgeführt.
- `NoContainer` – Die Lambda-Funktion wird als Prozess ohne Isolierung ausgeführt.

Der Standardwert ist `GreengrassContainer`.

iii. (Optional) Wenn Sie die Lambda-Funktion in einem Container ausführen, können Sie die Speichermenge und die Systemressourcen wie Volumes und Geräte so konfigurieren, dass sie dem Container zur Verfügung stehen. Gehen Sie wie folgt vor:

A. Fügen Sie das Containerparameterobjekt (`containerParams`) zum Linux-Prozessparameterobjekt in hinzu`lambda-function-component.json`.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
```

```
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {
        "containerParams": {

        }
      }
    }
  }
}
```

B. (Optional) Fügen Sie den `memorySizeInKB` Parameter hinzu, um die Speichergröße des Containers anzugeben. Der Standardwert ist 16 384 KB (16 MB).

C. (Optional) Fügen Sie den `mountROSysfs` Parameter hinzu, um anzugeben, ob der Container Informationen aus dem `/sys` Geräteordner lesen kann oder nicht. Der Standardwert ist `false`.

D. (Optional) Konfigurieren Sie die lokalen Volumes, auf die die containerisierte Lambda-Funktion zugreifen kann. Wenn Sie ein Volume definieren, mountet die AWS IoT Greengrass Core-Software die Quelldateien in das Ziel innerhalb des Containers. Gehen Sie wie folgt vor:

   I. Fügen Sie die Liste der Volumes (`volumes`) zu den Containerparametern hinzu.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
```

```
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {
        "containerParams": {
          "memorySizeInKB": 32768,
          "mountROSysfs": true,
          "volumes": [

          ]
        }
      }
    }
  }
}
```

II. Fügen Sie jedes Volume der Liste hinzu. Jedes Volume hat die folgenden Parameter:

- `sourcePath` – Der Pfad zum Quellordner auf dem Core-Gerät.
- `destinationPath` – Der Pfad zum Zielordner im Container.
- `permission` – (Optional) Die Berechtigung für den Zugriff auf den Quellordner aus dem Container. Wählen Sie aus den folgenden Optionen aus:
  - `ro` – Die Lambda-Funktion hat schreibgeschützten Zugriff auf den Quellordner.
  - `rw` – Die Lambda-Funktion hat Lese-/Schreibzugriff auf den Quellordner.

  Der Standardwert ist `ro`.
- `addGroupOwner` – (Optional) Gibt an, ob die Systemgruppe, die die Lambda-Funktionskomponente ausführt, als Eigentümer des Quellordners hinzugefügt werden soll oder nicht. Der Standardwert ist `false`.

Ihr `lambda-function-component.json` könnte ein Dokument ähnlich dem folgenden Beispiel enthalten.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
```

```
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {
        "containerParams": {
          "memorySizeInKB": 32768,
          "mountROSysfs": true,
          "volumes": [
            {
              "sourcePath": "/var/data/src",
              "destinationPath": "/var/data/dest",
              "permission": "rw",
              "addGroupOwner": true
            }
          ]
        }
      }
    }
  }
}
```

E. (Optional) Konfigurieren Sie die lokalen Systemgeräte, auf die die containerisierte Lambda-Funktion zugreifen kann. Gehen Sie wie folgt vor:

   I. Fügen Sie die Liste der Systemgeräte (`devices`) zu den Containerparametern hinzu.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {
        "containerParams": {
```

```
          "memorySizeInKB": 32768,
          "mountROSysfs": true,
          "volumes": [
            {
              "sourcePath": "/var/data/src",
              "destinationPath": "/var/data/dest",
              "permission": "rw",
              "addGroupOwner": true
            }
          ],
          "devices": [

          ]
        }
      }
    }
  }
}
```

II. Fügen Sie jedes Systemgerät der Liste hinzu. Jedes Systemgerät hat die folgenden Parameter:

- `path` – Der Pfad zum Systemgerät auf dem Core-Gerät.
- `permission` – (Optional) Die Berechtigung für den Zugriff auf das Systemgerät vom Container aus. Wählen Sie aus den folgenden Optionen aus:
  - `ro` – Die Lambda-Funktion hat schreibgeschützten Zugriff auf das Systemgerät.
  - `rw` – Die Lambda-Funktion hat Lese-/Schreibzugriff auf das Systemgerät.

  Der Standardwert ist `ro`.
- `addGroupOwner` – (Optional) Gibt an, ob die Systemgruppe, die die Lambda-Funktionskomponente ausführt, als Besitzer des Systemgeräts hinzugefügt werden soll oder nicht. Der Standardwert ist `false`.

Ihr `lambda-function-component.json` könnte ein Dokument ähnlich dem folgenden Beispiel enthalten.

```
{
  "lambdaFunction": {
```

```
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {
        "containerParams": {
          "memorySizeInKB": 32768,
          "mountROSysfs": true,
          "volumes": [
```

```
            {
              "sourcePath": "/var/data/src",
              "destinationPath": "/var/data/dest",
              "permission": "rw",
              "addGroupOwner": true
            }
          ],
          "devices": [
            {
              "path": "/dev/sda3",
              "permission": "rw",
              "addGroupOwner": true
            }
          ]
        }
      }
    }
  }
}
```

7. (Optional) Fügen Sie Tags (`tags`) für die Komponente hinzu. Weitere Informationen finden Sie unter Markieren Ihrer AWS IoT Greengrass Version 2-Ressourcen mit Tags.

Schritt 2: Erstellen der Lambda-Funktionskomponente

1. Führen Sie den folgenden Befehl aus, um die Lambda-Funktionskomponente aus zu erstellen`lambda-function-component.json`.

   ```
   aws greengrassv2 create-component-version --cli-input-json file://lambda-function-component.json
   ```

   Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anforderung erfolgreich ist.

   ```
   {
     "arn":
    "arn:aws:greengrass:region:123456789012:components:com.example.HelloWorldLambda:versions:1
     "componentName": "com.example.HelloWorldLambda",
     "componentVersion": "1.0.0",
     "creationTimestamp": "Mon Dec 15 20:56:34 UTC 2020",
     "status": {
       "componentState": "REQUESTED",
   ```

```
    "message": "NONE",
    "errors": {}
  }
}
```

Kopieren Sie die `arn` aus der Ausgabe, um den Status der Komponente im nächsten Schritt zu überprüfen.

2. Wenn Sie eine Komponente erstellen, lautet ihr Status `REQUESTED`. Anschließend AWS IoT Greengrass überprüft , ob die Komponente bereitstellbar ist. Sie können den folgenden Befehl ausführen, um den Komponentenstatus abzufragen und zu überprüfen, ob Ihre Komponente bereitstellbar ist. Ersetzen Sie durch `arn` den ARN aus dem vorherigen Schritt.

```
aws greengrassv2 describe-component \
  --arn "arn:aws:greengrass:region:account-
id:components:com.example.HelloWorldLambda:versions:1.0.0"
```

Wenn die Komponente validiert wird, gibt die Antwort an, dass der Komponentenstatus ist`DEPLOYABLE`.

```
{
  "arn": "arn:aws:greengrass:region:account-
id:components:com.example.HelloWorldLambda:versions:1.0.0",
  "componentName": "com.example.HelloWorldLambda",
  "componentVersion": "1.0.0",
  "creationTimestamp": "2020-12-15T20:56:34.376000-08:00",
  "publisher": "AWS Lambda",
  "status": {
    "componentState": "DEPLOYABLE",
    "message": "NONE",
    "errors": {}
  },
  "platforms": [
    {
      "name": "Linux x86",
      "attributes": {
        "architecture": "x86",
        "os": "linux"
      }
    }
  ]
```

```
}
```

Nachdem die Komponente ist`DEPLOYABLE`, können Sie die Lambda-Funktion auf Ihren -Core-Geräten bereitstellen. Weitere Informationen finden Sie unter Bereitstellen von AWS IoT Greengrass Komponenten auf Geräten.

Verwenden Sie den AWS IoT Device SDK , um mit dem Greengrass-Kern und anderen Komponenten zu kommunizieren und AWS IoT Core

Komponenten, die auf Ihrem Core-Gerät ausgeführt werden, können die AWS IoT Greengrass Core Interprocess Communication (IPC) -Bibliothek in der verwenden, AWS IoT Device SDK um mit dem AWS IoT Greengrass Nucleus und anderen Greengrass-Komponenten zu kommunizieren. Um benutzerdefinierte Komponenten zu entwickeln und auszuführen, die IPC verwenden, müssen Sie den verwenden, AWS IoT Device SDK um eine Verbindung zum AWS IoT Greengrass Core IPC-Dienst herzustellen und IPC-Operationen durchzuführen.

Die IPC-Schnittstelle unterstützt zwei Arten von Operationen:

- Anfrage/Antwort

  Komponenten senden eine Anfrage an den IPC-Dienst und erhalten eine Antwort, die das Ergebnis der Anfrage enthält.
- Abonnement

  Komponenten senden eine Abonnementanfrage an den IPC-Dienst und erwarten als Antwort einen Strom von Ereignisnachrichten. Komponenten stellen einen Abonnement-Handler bereit, der Ereignismeldungen, Fehler und das Schließen von Streams verarbeitet. Das AWS IoT Device SDK beinhaltet eine Handler-Schnittstelle mit den richtigen Antwort- und Ereignistypen für jeden IPC-Vorgang. Weitere Informationen finden Sie unter Abonnieren Sie IPC-Event-Streams.

Themen

- Interagieren mit dem Komponentenlebenszyklus
- Interagieren mit der Komponentenkonfiguration
- Abrufen von Secret-Werten
- Interagieren mit lokalen Schatten
- Verwalten von lokalen Bereitstellungen und Komponenten
- Authentifizieren und Autorisieren von Client-Geräten

IPC-Client-Versionen

In späteren Versionen der Java- und Python-SDKs wird eine verbesserte Version des IPC-Clients AWS IoT Greengrass bereitgestellt, die als IPC-Client V2 bezeichnet wird. IPC-Client V2:

- Reduziert die Menge an Code, die Sie für die Verwendung von IPC-Operationen schreiben müssen, und trägt dazu bei, häufige Fehler zu vermeiden, die beim IPC-Client V1 auftreten können.
- Ruft Abonnement-Handler-Callbacks in einem separaten Thread auf, sodass Sie jetzt Blockierungscode, einschließlich zusätzlicher IPC-Funktionsaufrufen, in Abonnement-Handler-Callbacks ausführen können. Der IPC-Client V1 verwendet denselben Thread, um mit dem IPC-Server zu kommunizieren und Abonnement-Handler-Callbacks aufzurufen.
- Ermöglicht das Aufrufen von Abonnementvorgängen mithilfe von Lambda-Ausdrücken (Java) oder Funktionen (Python). Für den IPC-Client V1 müssen Sie Abonnement-Handler-Klassen definieren.
- Stellt synchrone und asynchrone Versionen der einzelnen IPC-Operationen bereit. Der IPC-Client V1 stellt nur asynchrone Versionen der einzelnen Operationen bereit.

Wir empfehlen, dass Sie den IPC-Client V2 verwenden, um diese Verbesserungen nutzen zu können. Viele Beispiele in dieser Dokumentation und in einigen Online-Inhalten zeigen jedoch nur, wie der IPC-Client V1 verwendet wird. Anhand der folgenden Beispiele und Tutorials können Sie sich die Beispielkomponenten ansehen, die den IPC-Client V2 verwenden:

- PublishToTopicBeispiele
- SubscribeToTopicBeispiele
- Tutorial: Entwickeln einer Greengrass-Komponente, die Komponentenaktualisierungen verzögert
- Tutorial: Interagieren mit lokalen IoT-Geräten über MQTT

Derzeit unterstützt der AWS IoT Device SDK für C++ v2 nur den IPC-Client V1.

Unterstützte SDKs für die Kommunikation zwischen Prozessen

Die AWS IoT Greengrass IPC-Kernbibliotheken sind in den folgenden Versionen enthalten. AWS IoT Device SDK

SDK	Mindestversion	Verwendung	
AWS IoT Device SDK für Java v2	v1.6.0	Siehe Verwenden Sie AWS IoT Device SDK für Java v2 (IPC-Client V2)	
AWS IoT Device SDK für Python v2	v1.9.0	Siehe Verwendung AWS IoT Device SDK für Python v2 (IPC-Client V2)	
AWS IoT Device SDK für C++ v2	v1.17.0	Siehe Verwenden Sie AWS IoT Device SDK für C++ v2	
AWS IoT Device SDK für v2 JavaScript	v1.12.0	Siehe Verwenden Sie AWS IoT Device SDK für JavaScript v2 (IPC-Client V1)	

Connect zum AWS IoT Greengrass Core IPC-Dienst her

Um die Interprozesskommunikation in Ihrer benutzerdefinierten Komponente zu verwenden, müssen Sie eine Verbindung zu einem IPC-Server-Socket herstellen, auf dem die AWS IoT Greengrass Core-Software ausgeführt wird. Führen Sie die folgenden Aufgaben aus, um das herunterzuladen und AWS IoT Device SDK in der Sprache Ihrer Wahl zu verwenden.

Verwenden Sie AWS IoT Device SDK für Java v2 (IPC-Client V2)

Um den AWS IoT Device SDK für Java v2 (IPC-Client V2) zu verwenden

1. Laden Sie das AWS IoT Device SDK für Java v2 (v1.6.0 oder höher) herunter.

2. Führen Sie einen der folgenden Schritte aus, um Ihren benutzerdefinierten Code in Ihrer Komponente auszuführen:

   - Erstellen Sie Ihre Komponente als JAR-Datei, die die enthält AWS IoT Device SDK, und führen Sie diese JAR-Datei in Ihrem Komponentenrezept aus.
   - Definieren Sie das AWS IoT Device SDK JAR als Komponentenartefakt und fügen Sie dieses Artefakt dem Klassenpfad hinzu, wenn Sie Ihre Anwendung in Ihrem Komponentenrezept ausführen.

3. Verwenden Sie den folgenden Code, um den IPC-Client zu erstellen.

```
try (GreengrassCoreIPCClientV2 ipcClient =
 GreengrassCoreIPCClientV2.builder().build()) {
    // Use client.
} catch (Exception e) {
    LOGGER.log(Level.SEVERE, "Exception occurred when using IPC.", e);
    System.exit(1);
}
```

Verwendung AWS IoT Device SDK für Python v2 (IPC-Client V2)

Um das AWS IoT Device SDK für Python v2 (IPC-Client V2) zu verwenden

1. Laden Sie das AWS IoT Device SDK für Python herunter (v1.9.0 oder höher).
2. Fügen Sie die Installationsschritte des SDK zum Installationslebenszyklus im Rezept Ihrer Komponente hinzu.
3. Stellen Sie eine Verbindung zum AWS IoT Greengrass Core IPC-Dienst her. Verwenden Sie den folgenden Code, um den IPC-Client zu erstellen.

```
from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2

try:
    ipc_client = GreengrassCoreIPCClientV2()
    # Use IPC client.
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

Verwenden Sie AWS IoT Device SDK für C++ v2

Um die AWS IoT Device SDK Version 2 für C++ zu erstellen, muss ein Gerät über die folgenden Tools verfügen:

- C++ 11 oder höher
- CMake 3.1 oder höher
- Einer der folgenden Compiler:
  - GCC 4.8 oder höher
  - Clang 3.9 oder höher
  - MSVC 2015 oder später

Um das AWS IoT Device SDK für C++ v2 zu verwenden

1. Laden Sie das AWS IoT Device SDK für C++ v2 (v1.17.0 oder höher) herunter.
2. Folgen Sie den Installationsanweisungen in der README-Datei, um das AWS IoT Device SDK für C++ v2 aus dem Quellcode zu erstellen.
3. Verlinken Sie in Ihrem C++-Build-Tool die Greengrass IPC-Bibliothek,`AWS::GreengrassIpc-cpp`, die Sie im vorherigen Schritt erstellt haben. Das folgende `CMakeLists.txt` Beispiel verknüpft die Greengrass IPC-Bibliothek mit einem Projekt, das Sie mit CMake erstellen.

```
cmake_minimum_required(VERSION 3.1)
project (greengrassv2_pubsub_subscriber)

file(GLOB MAIN_SRC
        "*.h"
        "*.cpp"
        )
add_executable(${PROJECT_NAME} ${MAIN_SRC})

set_target_properties(${PROJECT_NAME} PROPERTIES
        LINKER_LANGUAGE CXX
        CXX_STANDARD 11)
find_package(aws-crt-cpp PATHS ~/sdk-cpp-workspace/build)
find_package(EventstreamRpc-cpp PATHS ~/sdk-cpp-workspace/build)
find_package(GreengrassIpc-cpp PATHS ~/sdk-cpp-workspace/build)
target_link_libraries(${PROJECT_NAME} AWS::GreengrassIpc-cpp)
```

4. Stellen Sie in Ihrem Komponentencode eine Verbindung zum AWS IoT Greengrass Core IPC-Dienst her, um einen IPC-Client zu erstellen (). `Aws::Greengrass::GreengrassCoreIpcClient` Sie müssen einen IPC-Verbindungslebenszyklus-Handler definieren, der IPC-Verbindungs-, Verbindungstrennungs- und Fehlerereignisse behandelt. Im folgenden Beispiel werden ein IPC-Client und ein IPC-Verbindungslebenszyklus-Handler erstellt, der druckt, wenn der IPC-Client eine Verbindung herstellt oder die Verbindung trennt und auf Fehler stößt.

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() <<
 std::endl;
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
};

int main() {
    // Create the IPC client.
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
```

```
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    // Use the IPC client to create an operation request.

    // Activate the operation request.
    auto activate = operation.Activate(request, nullptr);
    activate.wait();

    // Wait for Greengrass Core to respond to the request.
    auto responseFuture = operation.GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from
 Greengrass Core." << std::endl;
        exit(-1);
    }

    // Check the result of the request.
    auto response = responseFuture.get();
    if (response) {
        std::cout << "Successfully published to topic: " << topic << std::endl;
    } else {
        // An error occurred.
        std::cout << "Failed to publish to topic: " << topic << std::endl;
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            std::cout << "Operation error: " << error->GetMessage().value() <<
 std::endl;
        } else {
            std::cout << "RPC error: " << response.GetRpcError() << std::endl;
        }
        exit(-1);
    }

    return 0;
}
```

5. Um Ihren benutzerdefinierten Code in Ihrer Komponente auszuführen, erstellen Sie Ihren Code als binäres Artefakt und führen Sie das binäre Artefakt in Ihrem Komponentenrezept aus. Legen

Sie die `Execute` Berechtigung des Artefakts auf fest, `OWNER` damit die AWS IoT Greengrass Core-Software das binäre Artefakt ausführen kann.

Der `Manifests` Abschnitt Ihres Komponentenrezepts könnte dem folgenden Beispiel ähneln.

JSON

```
{
  ...
  "Manifests": [
    {
      "Lifecycle": {
        "run": "{artifacts:path}/greengrassv2_pubsub_subscriber"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pubsub_subscriber",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

YAML

```
...
Manifests:
  - Lifecycle:
      run: {artifacts:path}/greengrassv2_pubsub_subscriber
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pubsub_subscriber
        Permission:
          Execute: OWNER
```

Verwenden Sie AWS IoT Device SDK für JavaScript v2 (IPC-Client V1)

Um das AWS IoT Device SDK für JavaScript v2 für die Verwendung mit NodeJS zu erstellen, muss ein Gerät über die folgenden Tools verfügen:

- NodeJS 10.0 oder höher
  - Führen Sie aus`node -v`, um die Node-Version zu überprüfen.
- CMake 3.1 oder höher

Um das AWS IoT Device SDK für JavaScript v2 zu verwenden (IPC-Client V1)

1. Laden Sie das AWS IoT Device SDK für JavaScript v2 herunter (v1.12.10 oder höher).
2. Folgen Sie den Installationsanweisungen in der README-Datei, um das AWS IoT Device SDK für JavaScript Version 2 aus dem Quellcode zu erstellen.
3. Stellen Sie eine Verbindung zum AWS IoT Greengrass Core IPC-Dienst her. Gehen Sie wie folgt vor, um den IPC-Client zu erstellen und eine Verbindung herzustellen.
4. Verwenden Sie den folgenden Code, um den IPC-Client zu erstellen.

   ```
   import * as greengrascoreipc from 'aws-iot-device-sdk-v2';

   let client = greengrascoreipc.createClient();
   ```

5. Verwenden Sie den folgenden Code, um eine Verbindung von Ihrer Komponente zum Greengrass-Kern herzustellen.

   ```
   await client.connect();
   ```

Autorisieren Sie Komponenten zur Ausführung von IPC-Vorgängen

Damit Ihre benutzerdefinierten Komponenten einige IPC-Operationen verwenden können, müssen Sie Autorisierungsrichtlinien definieren, die es der Komponente ermöglichen, den Vorgang auf bestimmten Ressourcen auszuführen. Jede Autorisierungsrichtlinie definiert eine Liste von Vorgängen und eine Liste von Ressourcen, die die Richtlinie zulässt. Der IPC-Dienst zum Veröffentlichen und Abonnieren von Nachrichten definiert beispielsweise Veröffentlichungs- und Abonnementvorgänge für Themenressourcen. Sie können den * Platzhalter verwenden, um den Zugriff auf alle Operationen oder alle Ressourcen zu ermöglichen.

Autorisierungsrichtlinien definieren Sie mit dem `accessControl` Konfigurationsparameter, den Sie im Komponentenrezept oder bei der Bereitstellung der Komponente festlegen können. Das `accessControl` Objekt ordnet IPC-Dienstkennungen Listen von Autorisierungsrichtlinien zu. Sie können mehrere Autorisierungsrichtlinien für jeden IPC-Dienst definieren, um den Zugriff zu kontrollieren. Jede Autorisierungsrichtlinie hat eine Richtlinien-ID, die für alle Komponenten eindeutig sein muss.

> **Tip**
>
> Um eindeutige Richtlinien-IDs zu erstellen, können Sie den Komponentennamen, den IPC-Dienstnamen und einen Zähler kombinieren. Eine Komponente mit dem Namen `com.example.HelloWorld` könnte beispielsweise zwei Autorisierungsrichtlinien für das Publizieren/Abonnieren mit den folgenden IDs definieren:
>
> - `com.example.HelloWorld:pubsub:1`
> - `com.example.HelloWorld:pubsub:2`

Autorisierungsrichtlinien verwenden das folgende Format. Dieses Objekt ist der `accessControl` Konfigurationsparameter.

JSON

```
{
  "IPC service identifier": {
    "policyId": {
      "policyDescription": "description",
      "operations": [
        "operation1",
        "operation2"
      ],
      "resources": [
        "resource1",
        "resource2"
      ]
    }
  }
}
```

YAML

```
IPC service identifier:
  policyId:
    policyDescription: description
    operations:
      - operation1
      - operation2
    resources:
      - resource1
      - resource2
```

Platzhalter in Autorisierungsrichtlinien

Sie können den * Platzhalter im `resources` Element der IPC-Autorisierungsrichtlinien verwenden, um den Zugriff auf mehrere Ressourcen in einer einzigen Autorisierungsrichtlinie zu ermöglichen.

- In allen Versionen von Greengrass Nucleus können Sie ein einzelnes * Zeichen als Ressource angeben, um Zugriff auf alle Ressourcen zu gewähren.
- In Greengrass Nucleus v2.6.0 und höher können Sie den * Charakter in einer Ressource so angeben, dass er einer beliebigen Zeichenkombination entspricht. Sie können beispielsweise angeben, dass der `factory/1/devices/Thermostat*/status` Zugriff auf ein Statusthema für alle Thermostatgeräte in einer Fabrik gewährt werden soll, wobei der Name jedes Geräts mit beginnt. `Thermostat`

Wenn Sie Autorisierungsrichtlinien für den AWS IoT Core MQTT-IPC-Dienst definieren, können Sie auch MQTT-Platzhalter (+und#) verwenden, um mehrere Ressourcen zuzuordnen. Weitere Informationen finden Sie unter MQTT-Platzhalter in MQTT-IPC-Autorisierungsrichtlinien. AWS IoT Core

Rezeptvariablen in Autorisierungsrichtlinien

Wenn Sie Greengrass Nucleus v2.6.0 oder höher verwenden und die interpolateComponentConfigurationKonfigurationsoption von Greengrass Nucleus auf einstellen`true`, können Sie die `{iot:thingName}` Rezeptvariable in Autorisierungsrichtlinien verwenden. Wenn Sie eine Autorisierungsrichtlinie benötigen, die den Namen des Kerngeräts enthält, z. B. für MQTT-Themen oder Geräteschatten, können Sie diese Rezeptvariable verwenden, um eine einzelne Autorisierungsrichtlinie für eine Gruppe von Kerngeräten zu konfigurieren. Beispielsweise

können Sie einer Komponente den Zugriff auf die folgende Ressource für Shadow-IPC-Operationen gewähren.

```
$aws/things/{iot:thingName}/shadow/
```

Sonderzeichen in Autorisierungsrichtlinien

Um ein Literal * oder ein ? Zeichen in einer Autorisierungsrichtlinie anzugeben, müssen Sie eine Escape-Sequenz verwenden. Die folgenden Escape-Sequenzen weisen die AWS IoT Greengrass Core-Software an, den Literalwert anstelle der speziellen Bedeutung des Zeichens zu verwenden. Das * Zeichen ist beispielsweise ein Platzhalter, der einer beliebigen Kombination von Zeichen entspricht.

Wörtliches Zeichen	Escape-Sequenz	Hinweise
*	${*}	
?	${?}	AWS IoT Greengrass unterstützt derzeit nicht den ? Platzhalter, der einem einzelnen Zeichen entspricht.
$	${$}	Verwenden Sie diese Escape-Sequenz, um nach einer Ressource zu suchen, die enthält${. Um beispielsweise einer Ressource mit dem Namen zu entsprechen${resourceName} , müssen Sie Folgendes ${$}{resourceName} angeben: Andernfalls können Sie für die Suche nach einer Ressource$, die Folgendes enthält, ein Literal verwenden $, z. B. um Zugriff auf ein

Wörtliches Zeichen	Escape-Sequenz	Hinweise
		Thema zu gewähren, das mit `$aws` beginnt.

Beispiele für Autorisierungsrichtlinien

Anhand der folgenden Beispiele für Autorisierungsrichtlinien können Sie Autorisierungsrichtlinien für Ihre Komponenten konfigurieren.

Example Beispiel für ein Komponentenrezept mit einer Autorisierungsrichtlinie

Das folgende Beispiel für ein Komponentenrezept enthält ein `accessControl` Objekt, das eine Autorisierungsrichtlinie definiert. Diese Richtlinie autorisiert die `com.example.HelloWorld` Komponente, unter dem `test/topic` Thema zu veröffentlichen.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.HelloWorld:pubsub:1": {
            "policyDescription": "Allows access to publish to test/topic.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "test/topic"
            ]
          }
        }
      }
    }
  },
```

```
  "Manifests": [
    {
      "Lifecycle": {
        "run": "java -jar {artifacts:path}/HelloWorld.jar"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
ComponentDescription: A component that publishes messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        "com.example.HelloWorld:pubsub:1":
          policyDescription: Allows access to publish to test/topic.
          operations:
            - "aws.greengrass#PublishToTopic"
          resources:
            - "test/topic"
Manifests:
  - Lifecycle:
      run: |-
        java -jar {artifacts:path}/HelloWorld.jar
```

Example Beispiel für ein Update der Komponentenkonfiguration mit einer Autorisierungsrichtlinie

Das folgende Beispiel für ein Konfigurationsupdate in einer Bereitstellung gibt an, dass eine Komponente mit einem `accessControl` Objekt konfiguriert werden soll, das eine Autorisierungsrichtlinie definiert. Diese Richtlinie autorisiert die `com.example.HelloWorld` Komponente, unter dem `test/topic` Thema zu veröffentlichen.

Console

Konfiguration zum Zusammenführen

```
{
  "accessControl": {
    "aws.greengrass.ipc.pubsub": {
      "com.example.HelloWorld:pubsub:1": {
        "policyDescription": "Allows access to publish to test/topic.",
        "operations": [
          "aws.greengrass#PublishToTopic"
        ],
        "resources": [
          "test/topic"
        ]
      }
    }
  }
}
```

AWS CLI

Der folgende Befehl erstellt eine Bereitstellung auf einem Core-Gerät.

```
aws greengrassv2 create-deployment --cli-input-json file://hello-world-
deployment.json
```

Die `hello-world-deployment.json` Datei enthält das folgende JSON-Dokument.

```
{
  "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "deploymentName": "Deployment for MyGreengrassCore",
  "components": {
    "com.example.HelloWorld": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"accessControl\":{\"aws.greengrass.ipc.pubsub\":
{\"com.example.HelloWorld:pubsub:1\":{\"policyDescription\":\"Allows access to
 publish to test/topic.\",\"operations\":[\"aws.greengrass#PublishToTopic\"],
\"resources\":[\"test/topic\"]}}}}"
      }
    }
```

```
  }
}
```

Greengrass CLI

Der folgende Greengrass-CLI-Befehl erstellt eine lokale Bereitstellung auf einem Core-Gerät.

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.HelloWorld=1.0.0" \
  --update-config hello-world-configuration.json
```

Die `hello-world-configuration.json` Datei enthält das folgende JSON-Dokument.

```
{
  "com.example.HelloWorld": {
    "MERGE": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.HelloWorld:pubsub:1": {
            "policyDescription": "Allows access to publish to test/topic.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "test/topic"
            ]
          }
        }
      }
    }
  }
}
```

Abonnieren Sie IPC-Event-Streams

Sie können IPC-Operationen verwenden, um Streams von Ereignissen auf einem Greengrass-Core-Gerät zu abonnieren. Um einen Abonnement-Vorgang zu verwenden, definieren Sie einen Abonnement-Handler und erstellen Sie eine Anfrage an den IPC-Dienst. Anschließend führt der

IPC-Client die Funktionen des Abonnement-Handlers jedes Mal aus, wenn das Kerngerät eine Ereignismeldung an Ihre Komponente streamt.

Sie können ein Abonnement schließen, um die Verarbeitung von Ereignismeldungen zu beenden. Rufen Sie dazu `closeStream()` (Java), (Python) oder `close()` `Close()` (C++) für das Objekt für den Abonnementvorgang auf, mit dem Sie das Abonnement geöffnet haben.

Der AWS IoT Greengrass Core IPC-Dienst unterstützt die folgenden Abonnementvorgänge:

- SubscribeToTopic
- SubscribeToIoTCore
- SubscribeToComponentUpdates
- SubscribeToConfigurationUpdate
- SubscribeToValidateConfigurationUpdates

Themen

Definieren Sie Abonnement-Handler

Um einen Abonnement-Handler zu definieren, definieren Sie Callback-Funktionen, die Ereignismeldungen, Fehler und das Schließen von Streams behandeln. Wenn Sie den IPC-Client V1 verwenden, müssen Sie diese Funktionen in einer Klasse definieren. Wenn Sie den IPC-Client V2 verwenden, der in späteren Versionen der Java- und Python-SDKs verfügbar ist, können Sie diese Funktionen definieren, ohne eine Abonnement-Handler-Klasse zu erstellen.

Java

Wenn Sie den IPC-Client V1 verwenden, müssen Sie die generische Schnittstelle implementieren. `software.amazon.awssdk.eventstreamrpc.StreamResponseHandler<`*`StreamEventType`*`>` *`StreamEventType`*ist der Typ der Ereignisnachricht für den Abonnementvorgang. Definieren Sie die folgenden Funktionen, um Ereignismeldungen, Fehler und das Schließen von Streams zu behandeln.

Wenn Sie den IPC-Client V2 verwenden, können Sie diese Funktionen außerhalb einer Abonnement-Handler-Klasse definieren oder Lambda-Ausdrücke verwenden.

```
void onStreamEvent(StreamEventType event)
```

Der Callback, den der IPC-Client aufruft, wenn er eine Ereignisnachricht empfängt, z. B. eine MQTT-Nachricht oder eine Benachrichtigung über ein Komponenten-Update.

```
boolean onStreamError(Throwable error)
```

Der Callback, den der IPC-Client aufruft, wenn ein Stream-Fehler auftritt.

Geben Sie true zurück, um den Abonnement-Stream aufgrund des Fehlers zu schließen, oder geben Sie false zurück, um den Stream geöffnet zu lassen.

```
void onStreamClosed()
```

Der Callback, den der IPC-Client aufruft, wenn der Stream geschlossen wird.

Python

Wenn Sie den IPC-Client V1 verwenden, müssen Sie die Stream-Response-Handler-Klasse erweitern, die dem Abonnementvorgang entspricht. Die AWS IoT Device SDK beinhaltet eine Abonnement-Handler-Klasse für jeden Abonnementvorgang. *StreamEventType*ist der Typ der Ereignisnachricht für den Abonnementvorgang. Definieren Sie die folgenden Funktionen, um Ereignismeldungen, Fehler und das Schließen von Streams zu behandeln.

Wenn Sie den IPC-Client V2 verwenden, können Sie diese Funktionen außerhalb einer Abonnement-Handler-Klasse definieren oder Lambda-Ausdrücke verwenden.

```
def on_stream_event(self, event: StreamEventType) -> None
```

Der Callback, den der IPC-Client aufruft, wenn er eine Ereignisnachricht empfängt, z. B. eine MQTT-Nachricht oder eine Benachrichtigung über ein Komponenten-Update.

```
def on_stream_error(self, error: Exception) -> bool
```

Der Callback, den der IPC-Client aufruft, wenn ein Stream-Fehler auftritt.

Geben Sie true zurück, um den Abonnement-Stream aufgrund des Fehlers zu schließen, oder geben Sie false zurück, um den Stream geöffnet zu lassen.

```
def on_stream_closed(self) -> None
```

Der Callback, den der IPC-Client aufruft, wenn der Stream geschlossen wird.

C++

Implementieren Sie eine Klasse, die von der Stream-Response-Handler-Klasse abgeleitet ist, die dem Abonnementvorgang entspricht. Die AWS IoT Device SDK beinhaltet eine Abonnement-Handler-Basisklasse für jeden Abonnementvorgang. *StreamEventType*ist der Typ der Ereignisnachricht für den Abonnementvorgang. Definieren Sie die folgenden Funktionen, um Ereignismeldungen, Fehler und das Schließen von Streams zu behandeln.

`void OnStreamEvent(StreamEventType *event)`

Der Callback, den der IPC-Client aufruft, wenn er eine Ereignisnachricht empfängt, z. B. eine MQTT-Nachricht oder eine Benachrichtigung über ein Komponenten-Update.

`bool OnStreamError(OperationError *error)`

Der Callback, den der IPC-Client aufruft, wenn ein Stream-Fehler auftritt.

Geben Sie true zurück, um den Abonnement-Stream aufgrund des Fehlers zu schließen, oder geben Sie false zurück, um den Stream geöffnet zu lassen.

`void OnStreamClosed()`

Der Callback, den der IPC-Client aufruft, wenn der Stream geschlossen wird.

JavaScript

Implementieren Sie eine Klasse, die von der Stream-Response-Handler-Klasse abgeleitet ist, die dem Abonnementvorgang entspricht. Die AWS IoT Device SDK beinhaltet eine Abonnement-Handler-Basisklasse für jeden Abonnementvorgang. *StreamEventType*ist der Typ der Ereignisnachricht für den Abonnementvorgang. Definieren Sie die folgenden Funktionen, um Ereignismeldungen, Fehler und das Schließen von Streams zu behandeln.

`on(event: 'ended', listener: StreamingOperationEndedListener)`

Der Callback, den der IPC-Client aufruft, wenn der Stream geschlossen wird.

`on(event: 'streamError', listener: StreamingRpcErrorListener)`

Der Callback, den der IPC-Client aufruft, wenn ein Stream-Fehler auftritt.

Geben Sie true zurück, um den Abonnement-Stream aufgrund des Fehlers zu schließen, oder geben Sie false zurück, um den Stream geöffnet zu lassen.

```
on(event: 'message', listener: (message: InboundMessageType) => void)
```

Der Callback, den der IPC-Client aufruft, wenn er eine Ereignisnachricht empfängt, z. B. eine MQTT-Nachricht oder eine Benachrichtigung über ein Komponenten-Update.

Beispiel für Abonnement-Handler

Das folgende Beispiel zeigt, wie der SubscribeToTopic Vorgang und ein Abonnement-Handler verwendet werden, um lokale Veröffentlichungs-/Abonnementnachrichten zu abonnieren.

Java (IPC client V2)

Example Beispiel: Abonnieren Sie lokale Publish/Subscribe-Nachrichten

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.SubscribeToTopicResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.*;

import java.nio.charset.StandardCharsets;
import java.util.Optional;

public class SubscribeToTopicV2 {

    public static void main(String[] args) {
        String topic = args[0];
        try (GreengrassCoreIPCClientV2 ipcClient =
 GreengrassCoreIPCClientV2.builder().build()) {
            SubscribeToTopicRequest request = new
 SubscribeToTopicRequest().withTopic(topic);
            GreengrassCoreIPCClientV2.StreamingResponse<SubscribeToTopicResponse,
                    SubscribeToTopicResponseHandler> response =
                    ipcClient.subscribeToTopic(request,
 SubscribeToTopicV2::onStreamEvent,
                            Optional.of(SubscribeToTopicV2::onStreamError),
                            Optional.of(SubscribeToTopicV2::onStreamClosed));
            SubscribeToTopicResponseHandler responseHandler =
 response.getHandler();
            System.out.println("Successfully subscribed to topic: " + topic);

            // Keep the main thread alive, or the process will exit.
```

```
            try {
                while (true) {
                    Thread.sleep(10000);
                }
            } catch (InterruptedException e) {
                System.out.println("Subscribe interrupted.");
            }

            // To stop subscribing, close the stream.
            responseHandler.closeStream();
        } catch (Exception e) {
            if (e.getCause() instanceof UnauthorizedError) {
                System.err.println("Unauthorized error while publishing to topic: "
+ topic);
            } else {
                System.err.println("Exception occurred when using IPC.");
            }
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static void onStreamEvent(SubscriptionResponseMessage
subscriptionResponseMessage) {
        try {
            BinaryMessage binaryMessage =
subscriptionResponseMessage.getBinaryMessage();
            String message = new String(binaryMessage.getMessage(),
StandardCharsets.UTF_8);
            String topic = binaryMessage.getContext().getTopic();
            System.out.printf("Received new message on topic %s: %s%n", topic,
message);
        } catch (Exception e) {
            System.err.println("Exception occurred while processing subscription
response " +
                    "message.");
            e.printStackTrace();
        }
    }

    public static boolean onStreamError(Throwable error) {
        System.err.println("Received a stream error.");
        error.printStackTrace();
        return false; // Return true to close stream, false to keep stream open.
```

```
    }

    public static void onStreamClosed() {
        System.out.println("Subscribe to topic stream closed.");
    }
}
```

Python (IPC client V2)

Example Beispiel: Lokale Nachrichten zum Publizieren/Abonnieren abonnieren

```
import sys
import time
import traceback

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import (
    SubscriptionResponseMessage,
    UnauthorizedError
)


def main():
    args = sys.argv[1:]
    topic = args[0]

    try:
        ipc_client = GreengrassCoreIPCClientV2()
        # Subscription operations return a tuple with the response and the
 operation.
        _, operation = ipc_client.subscribe_to_topic(topic=topic,
 on_stream_event=on_stream_event,

 on_stream_error=on_stream_error, on_stream_closed=on_stream_closed)
        print('Successfully subscribed to topic: ' + topic)

        # Keep the main thread alive, or the process will exit.
        try:
            while True:
                time.sleep(10)
        except InterruptedError:
            print('Subscribe interrupted.')

        # To stop subscribing, close the stream.
```

```
        operation.close()
    except UnauthorizedError:
        print('Unauthorized error while subscribing to topic: ' +
              topic, file=sys.stderr)
        traceback.print_exc()
        exit(1)
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)


def on_stream_event(event: SubscriptionResponseMessage) -> None:
    try:
        message = str(event.binary_message.message, 'utf-8')
        topic = event.binary_message.context.topic
        print('Received new message on topic %s: %s' % (topic, message))
    except:
        traceback.print_exc()


def on_stream_error(error: Exception) -> bool:
    print('Received a stream error.', file=sys.stderr)
    traceback.print_exc()
    return False  # Return True to close stream, False to keep stream open.


def on_stream_closed() -> None:
    print('Subscribe to topic stream closed.')


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

C++

Example Beispiel: Lokale Nachrichten zum Publizieren/Abonnieren abonnieren

```
#include <iostream>

#include </crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
```

```
using namespace Aws::Greengrass;

class SubscribeResponseHandler : public SubscribeToTopicStreamHandler {
    public:
        virtual ~SubscribeResponseHandler() {}

    private:
        void OnStreamEvent(SubscriptionResponseMessage *response) override {
            auto jsonMessage = response->GetJsonMessage();
            if (jsonMessage.has_value() &&
 jsonMessage.value().GetMessage().has_value()) {
                auto messageString =
 jsonMessage.value().GetMessage().value().View().WriteReadable();
                // Handle JSON message.
            } else {
                auto binaryMessage = response->GetBinaryMessage();
                if (binaryMessage.has_value() &&
 binaryMessage.value().GetMessage().has_value()) {
                    auto messageBytes = binaryMessage.value().GetMessage().value();
                    std::string messageString(messageBytes.begin(),
 messageBytes.end());
                    // Handle binary message.
                }
            }
        }

        bool OnStreamError(OperationError *error) override {
            // Handle error.
            return false; // Return true to close stream, false to keep stream open.
        }

        void OnStreamClosed() override {
            // Handle close.
        }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }
```

```
    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};

int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String topic("my/topic");
    int timeout = 10;

    SubscribeToTopicRequest request;
    request.SetTopic(topic);

    //SubscribeResponseHandler streamHandler;
    auto streamHandler = MakeShared<SubscribeResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToTopic(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
 Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
```

```
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.
        } else {
            // Handle RPC error.
        }
        exit(-1);
    }

    // Keep the main thread alive, or the process will exit.
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
    }

    operation->Close();
    return 0;
}
```

JavaScript

Example Beispiel: Lokale Nachrichten zum Publizieren/Abonnieren abonnieren

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {SubscribeToTopicRequest, SubscriptionResponseMessage} from "aws-iot-device-
sdk-v2/dist/greengrasscoreipc/model";
import {RpcError} from "aws-iot-device-sdk-v2/dist/eventstream_rpc";

class SubscribeToTopic {
    private ipcClient : greengrasscoreipc.Client
    private readonly topic : string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.subscribeToTopic().then(r => console.log("Started workflow"));
    }

    private async subscribeToTopic() {
        try {
            this.ipcClient = await getIpcClient();
```

```
            const subscribeToTopicRequest : SubscribeToTopicRequest = {
                topic: this.topic,
            }

            const streamingOperation =
 this.ipcClient.subscribeToTopic(subscribeToTopicRequest, undefined); //
 conditionally apply options

            streamingOperation.on("message", (message: SubscriptionResponseMessage)
 => {
                // parse the message depending on your use cases, e.g.
                if(message.binaryMessage && message.binaryMessage.message) {
                    const receivedMessage =
 message.binaryMessage?.message.toString();
                }
            });

            streamingOperation.on("streamError", (error : RpcError) => {
                // define your own error handling logic
            })

            streamingOperation.on("ended", () => {
                // define your own logic
            })

            await streamingOperation.activate();

            // Keep the main thread alive, or the process will exit.
            await new Promise((resolve) => setTimeout(resolve, 10000))
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}

export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
```

```
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

// starting point
const subscribeToTopic = new SubscribeToTopic();
```

Bewährte IPC-Praktiken

Die bewährten Methoden für die Verwendung von IPC in benutzerdefinierten Komponenten unterscheiden sich zwischen IPC-Client V1 und IPC-Client V2. Folgen Sie den bewährten Methoden für die IPC-Client-Version, die Sie verwenden.

IPC client V2

Der IPC-Client V2 führt Callback-Funktionen in einem separaten Thread aus. Im Vergleich zu IPC-Client V1 gibt es also weniger Richtlinien, die Sie befolgen müssen, wenn Sie IPC verwenden und Abonnement-Handler-Funktionen schreiben.

- Einen IPC-Client wiederverwenden

  Nachdem Sie einen IPC-Client erstellt haben, lassen Sie ihn geöffnet und verwenden Sie ihn für alle IPC-Operationen erneut. Das Erstellen mehrerer Clients verbraucht zusätzliche Ressourcen und kann zu Ressourcenlecks führen.

- Behandeln Sie Ausnahmen

  Der IPC-Client V2 protokolliert nicht abgefangene Ausnahmen in Abonnement-Handler-Funktionen. Sie sollten Ausnahmen in Ihren Handlerfunktionen abfangen, um Fehler zu behandeln, die in Ihrem Code auftreten.

IPC client V1

Der IPC-Client V1 verwendet einen einzigen Thread, der mit dem IPC-Server kommuniziert und Abonnement-Handler aufruft. Sie müssen dieses synchrone Verhalten berücksichtigen, wenn Sie Abonnement-Handler-Funktionen schreiben.

- Einen IPC-Client wiederverwenden

  Nachdem Sie einen IPC-Client erstellt haben, lassen Sie ihn geöffnet und verwenden Sie ihn für alle IPC-Operationen erneut. Das Erstellen mehrerer Clients verbraucht zusätzliche Ressourcen und kann zu Ressourcenlecks führen.

- Führen Sie den Blockierungscode asynchron aus

  Der IPC-Client V1 kann keine neuen Anfragen senden oder neue Ereignisnachrichten verarbeiten, solange der Thread blockiert ist. Sie sollten den Blockierungscode in einem separaten Thread ausführen, den Sie von der Handler-Funktion aus ausführen. Blockierungscode umfasst `sleep` Aufrufe, Schleifen, die kontinuierlich ausgeführt werden, und synchrone I/O-Anfragen, deren Abschluss einige Zeit in Anspruch nimmt.

- Senden Sie neue IPC-Anfragen asynchron

  Der IPC-Client V1 kann innerhalb der Abonnement-Handler-Funktionen keine neue Anfrage senden, da die Anfrage die Handler-Funktion blockiert, wenn Sie auf eine Antwort warten. Sie sollten IPC-Anfragen in einem separaten Thread senden, den Sie von der Handler-Funktion aus ausführen.

- Behandeln Sie Ausnahmen

  Der IPC-Client V1 behandelt keine nicht abgefangenen Ausnahmen in Abonnement-Handler-Funktionen. Wenn Ihre Handlerfunktion eine Ausnahme auslöst, wird das Abonnement geschlossen und die Ausnahme erscheint nicht in Ihren Komponentenprotokollen. Sie sollten Ausnahmen in Ihren Handler-Funktionen abfangen, um das Abonnement offen zu halten und Fehler zu protokollieren, die in Ihrem Code auftreten.

Lokale Nachrichten veröffentlichen/abonnieren

Mit Publish/Sub-Nachrichten (Pubsub) können Sie Nachrichten zu Themen senden und empfangen. Komponenten können Nachrichten zu Themen veröffentlichen, um Nachrichten an andere Komponenten zu senden. Komponenten, die dieses Thema abonniert haben, können dann auf die Nachrichten reagieren, die sie erhalten.

> **Note**
>
> Sie können diesen IPC-Dienst zum Veröffentlichen/Abonnieren nicht verwenden, um MQTT zu veröffentlichen oder zu abonnieren. AWS IoT Core Weitere Informationen zum

Austausch von Nachrichten mit MQTT finden Sie unter. AWS IoT Core MQTT-Nachrichten veröffentlichen/abonnieren AWS IoT Core

Themen

Minimale SDK-Versionen

In der folgenden Tabelle sind die Mindestversionen von aufgeführt AWS IoT Device SDK , die Sie verwenden müssen, um Nachrichten zu und von lokalen Themen zu veröffentlichen und zu abonnieren.

SDK	Mindestversion	
AWS IoT Device SDK für Java v2	v1.2.10	
AWS IoT Device SDK für Python v2	v1.5.3	
AWS IoT Device SDK für C++ v2	v1.17.0	
AWS IoT Device SDK for JavaScript v2	v1.12.0	

Autorisierung

Um lokale Veröffentlichungs-/Abonnementnachrichten in einer benutzerdefinierten Komponente zu verwenden, müssen Sie Autorisierungsrichtlinien definieren, die es Ihrer Komponente ermöglichen, Nachrichten an Themen zu senden und zu empfangen. Informationen zur Definition von

Autorisierungsrichtlinien finden Sie unter. Autorisieren Sie Komponenten zur Ausführung von IPC-Vorgängen

Autorisierungsrichtlinien für Publish/Subscribe-Messaging haben die folgenden Eigenschaften.

IPC-Dienst-ID: `aws.greengrass.ipc.pubsub`

Operation	Beschreibung	Ressourcen
`aws.greengrass#PublishToTopic`	Ermöglicht einer Komponente, Nachrichten zu den von Ihnen angegebenen Themen zu veröffentlichen.	Eine Themenzeichenfolge, z. `test/topic` B. Verwenden Sie an`*`, um einer beliebigen Zeichenkombination in einem Thema zu entsprechen. Diese Themenzeichenfolge unterstützt keine Platzhalter (#und+) für MQTT-Themen.
`aws.greengrass#SubscribeToTopic`	Ermöglicht einer Komponente, Nachrichten für die von Ihnen angegebenen Themen zu abonnieren.	Eine Themenzeichenfolge, z. `test/topic` B. Verwenden Sie an`*`, um einer beliebigen Zeichenkombination in einem Thema zu entsprechen. In Greengrass Nucleus v2.6.0 und höher können Sie Themen abonnieren, die Platzhalter (und) für MQTT-Themen enthalten. # + Diese Themenzeichenfolge unterstützt Platzhalter für MQTT-Themen als Literalzeichen. Wenn beispielsweise die Autorisierungsrichtlinie einer Komponente Zugriff auf gewährt`test/topic/#` , kann die Komponente zwar

Operation	Beschreibung	Ressourcen
		abonnierentest/topic/# , aber nicht abonnieren. test/topic/filter
*	Ermöglicht einer Komponente, Nachrichten zu den von Ihnen angegebenen Themen zu veröffentlichen und zu abonnieren.	Eine Themenzeichenfolge, z. test/topic B. Verwenden Sie an*, um einer beliebigen Zeichenkombination in einem Thema zu entsprechen. In Greengrass Nucleus v2.6.0 und höher können Sie Themen abonnieren, die Platzhalter (und) für MQTT-Themen enthalten. # + Diese Themenzeichenfolge unterstützt Platzhalter für MQTT-Themen als Literalzeichen. Wenn beispielsweise die Autorisierungsrichtlinie einer Komponente Zugriff auf gewährttest/topic/# , kann die Komponente zwar abonnierentest/topic/# , aber nicht abonnieren. test/topic/filter

Beispiele für Autorisierungsrichtlinien

Anhand des folgenden Beispiels für Autorisierungsrichtlinien können Sie Autorisierungsrichtlinien für Ihre Komponenten konfigurieren.

Example Beispiel für eine Autorisierungsrichtlinie

Das folgende Beispiel für eine Autorisierungsrichtlinie ermöglicht es einer Komponente, alle Themen zu veröffentlichen und zu abonnieren.

```
{
  "accessControl": {
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyLocalPubSubComponent:pubsub:1": {
        "policyDescription": "Allows access to publish/subscribe to all topics.",
        "operations": [
          "aws.greengrass#PublishToTopic",
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

PublishToTopic

Veröffentlichen einer Nachricht für ein Thema.

Anforderung

Die Anforderung dieses Vorgangs hat die folgenden Parameter:

`topic`

Das Thema, zu dem die Nachricht veröffentlicht werden soll.

`publishMessage`(Python:`publish_message`)

Die zu veröffentlichende Nachricht. Dieses Objekt,`PublishMessage`, enthält die folgenden Informationen. Sie müssen einen von `jsonMessage` und angeben`binaryMessage`.

`jsonMessage`(Python:`json_message`)

(Optional) Eine JSON-Nachricht. Dieses Objekt,`JsonMessage`, enthält die folgenden Informationen:

`message`

Die JSON-Nachricht als Objekt.

`context`

Der Kontext der Nachricht, z. B. das Thema, in dem die Nachricht veröffentlicht wurde.

Diese Funktion ist für Version 2.6.0 und höher der Greengrass Nucleus-Komponente verfügbar. In der folgenden Tabelle sind die Mindestversionen von aufgeführt AWS IoT Device SDK , die Sie für den Zugriff auf den Nachrichtenkontext verwenden müssen.

SDK	Mindestversion	
AWS IoT Device SDK für Java v2	v1.9.3	
AWS IoT Device SDK für Python v2	v1.11.3	
AWS IoT Device SDK für C ++ v2	v1.18.4	
AWS IoT Device SDK for JavaScript v2	v1.12.0	

> **Note**
>
> Die AWS IoT Greengrass Core-Software verwendet dieselben Nachrichtenobjekte in den Operationen `PublishToTopic` und`SubscribeToTopic`. Die AWS IoT Greengrass Core-Software legt dieses Kontextobjekt in Nachrichten fest, wenn Sie sie abonnieren, und ignoriert dieses Kontextobjekt in Nachrichten, die Sie veröffentlichen.

Dieses Objekt,`MessageContext`, enthält die folgenden Informationen:

`topic`

Das Thema, in dem die Nachricht veröffentlicht wurde.

`binaryMessage`(Python:`binary_message`)

(Optional) Eine binäre Nachricht. Dieses Objekt,`BinaryMessage`, enthält die folgenden Informationen:

`message`

Die binäre Nachricht als Blob.

`context`

Der Kontext der Nachricht, z. B. das Thema, in dem die Nachricht veröffentlicht wurde.

Diese Funktion ist für Version 2.6.0 und höher der Greengrass Nucleus-Komponente verfügbar. In der folgenden Tabelle sind die Mindestversionen von aufgeführt AWS IoT Device SDK , die Sie für den Zugriff auf den Nachrichtenkontext verwenden müssen.

SDK	Mindestversion
AWS IoT Device SDK für Java v2	v1.9.3
AWS IoT Device SDK für Python v2	v1.11.3
AWS IoT Device SDK für C ++ v2	v1.18.4
AWS IoT Device SDK for JavaScript v2	v1.12.0

> **Note**
>
> Die AWS IoT Greengrass Core-Software verwendet dieselben Nachrichtenobjekte in den Operationen `PublishToTopic` und`SubscribeToTopic`. Die AWS IoT Greengrass Core-Software legt dieses Kontextobjekt in Nachrichten fest, wenn Sie sie abonnieren, und ignoriert dieses Kontextobjekt in Nachrichten, die Sie veröffentlichen.

Dieses Objekt,`MessageContext`, enthält die folgenden Informationen:

`topic`

Das Thema, in dem die Nachricht veröffentlicht wurde.

Antwort

Dieser Vorgang liefert in seiner Antwort keine Informationen.

Beispiele

Die folgenden Beispiele zeigen, wie dieser Vorgang in benutzerdefiniertem Komponentencode aufgerufen wird.

Java (IPC client V2)

Example Beispiel: Veröffentlichen Sie eine binäre Nachricht

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.model.BinaryMessage;
import software.amazon.awssdk.aws.greengrass.model.PublishMessage;
import software.amazon.awssdk.aws.greengrass.model.PublishToTopicRequest;
import software.amazon.awssdk.aws.greengrass.model.PublishToTopicResponse;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;

import java.nio.charset.StandardCharsets;

public class PublishToTopicV2 {

    public static void main(String[] args) {
        String topic = args[0];
        String message = args[1];
        try (GreengrassCoreIPCClientV2 ipcClient =
 GreengrassCoreIPCClientV2.builder().build()) {
            PublishToTopicV2.publishBinaryMessageToTopic(ipcClient, topic,
 message);
            System.out.println("Successfully published to topic: " + topic);
        } catch (Exception e) {
            if (e.getCause() instanceof UnauthorizedError) {
                System.err.println("Unauthorized error while publishing to topic: "
 + topic);
```

```
            } else {
                System.err.println("Exception occurred when using IPC.");
            }
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static PublishToTopicResponse publishBinaryMessageToTopic(
            GreengrassCoreIPCClientV2 ipcClient, String topic, String message)
 throws InterruptedException {
        BinaryMessage binaryMessage =
                new
 BinaryMessage().withMessage(message.getBytes(StandardCharsets.UTF_8));
        PublishMessage publishMessage = new
 PublishMessage().withBinaryMessage(binaryMessage);
        PublishToTopicRequest publishToTopicRequest =
                new
 PublishToTopicRequest().withTopic(topic).withPublishMessage(publishMessage);
        return ipcClient.publishToTopic(publishToTopicRequest);
    }
}
```

Python (IPC client V2)

Example Beispiel: Veröffentlichen Sie eine binäre Nachricht

```
import sys
import traceback

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import (
    PublishMessage,
    BinaryMessage
)


def main():
    args = sys.argv[1:]
    topic = args[0]
    message = args[1]

    try:
        ipc_client = GreengrassCoreIPCClientV2()
```

```
        publish_binary_message_to_topic(ipc_client, topic, message)
        print('Successfully published to topic: ' + topic)
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)


def publish_binary_message_to_topic(ipc_client, topic, message):
    binary_message = BinaryMessage(message=bytes(message, 'utf-8'))
    publish_message = PublishMessage(binary_message=binary_message)
    return ipc_client.publish_to_topic(topic=topic,
 publish_message=publish_message)


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

C++

Example Beispiel: Veröffentlichen Sie eine binäre Nachricht

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};
```

```
int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String topic("my/topic");
    String message("Hello, World!");
    int timeout = 10;

    PublishToTopicRequest request;
    Vector<uint8_t> messageData({message.begin(), message.end()});
    BinaryMessage binaryMessage;
    binaryMessage.SetMessage(messageData);
    PublishMessage publishMessage;
    publishMessage.SetBinaryMessage(binaryMessage);
    request.SetTopic(topic);
    request.SetPublishMessage(publishMessage);

    auto operation = ipcClient.NewPublishToTopic();
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
 Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
```

```
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.
        } else {
            // Handle RPC error.
        }
    }
    return 0;
}
```

JavaScript

Example Beispiel: Veröffentlichen Sie eine binäre Nachricht

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {BinaryMessage, PublishMessage, PublishToTopicRequest} from "aws-iot-device-
sdk-v2/dist/greengrasscoreipc/model";

class PublishToTopic {
    private ipcClient : greengrasscoreipc.Client
    private readonly topic : string;
    private readonly messageString : string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.messageString = "<define_your_message_string>";
        this.publishToTopic().then(r => console.log("Started workflow"));
    }

    private async publishToTopic() {
        try {
            this.ipcClient = await getIpcClient();

            const binaryMessage : BinaryMessage = {
                message: this.messageString
            }

            const publishMessage : PublishMessage = {
                binaryMessage: binaryMessage
            }
```

```
            const request : PublishToTopicRequest = {
                topic: this.topic,
                publishMessage: publishMessage
            }

            this.ipcClient.publishToTopic(request).finally(() =>
 console.log(`Published message ${publishMessage.binaryMessage?.message} to topic`))

        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}


export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

// starting point
const publishToTopic = new PublishToTopic();
```

SubscribeToTopic

Abonnieren Sie Nachrichten zu einem Thema.

Bei diesem Vorgang handelt es sich um einen Abonnementvorgang, bei dem Sie einen Stream von Ereignisnachrichten abonnieren. Um diese Operation zu verwenden, definieren Sie einen Stream-Response-Handler mit Funktionen, die Ereignismeldungen, Fehler und das Schließen von Streams behandeln. Weitere Informationen finden Sie unter Abonnieren Sie IPC-Event-Streams.

Typ der Ereignisnachricht: `SubscriptionResponseMessage`

Anforderung

Die Anforderung dieser Operation hat die folgenden Parameter:

`topic`

Das Thema, das abonniert werden soll.

> **Note**
>
> In Greengrass Nucleus v2.6.0 und höher unterstützt dieses Thema Platzhalter (und) für MQTT-Themen. # +

`receiveMode`(Python:`receive_mode`)

(Optional) Das Verhalten, das angibt, ob die Komponente Nachrichten von sich selbst empfängt. Sie können dieses Verhalten ändern, damit eine Komponente auf ihre eigenen Nachrichten reagieren kann. Das Standardverhalten hängt davon ab, ob das Thema einen MQTT-Platzhalter enthält. Wählen Sie aus den folgenden Optionen aus:

- `RECEIVE_ALL_MESSAGES`— Empfangen Sie alle Nachrichten, die dem Thema entsprechen, einschließlich Nachrichten von der Komponente, die das Abonnement abonniert.

  Dieser Modus ist die Standardoption, wenn Sie ein Thema abonnieren, das keinen MQTT-Platzhalter enthält.

- `RECEIVE_MESSAGES_FROM_OTHERS`— Empfangen Sie alle Nachrichten, die dem Thema entsprechen, mit Ausnahme von Nachrichten von der Komponente, die das Abonnement abonniert.

  Dieser Modus ist die Standardoption, wenn Sie ein Thema abonnieren, das einen MQTT-Platzhalter enthält.

Diese Funktion ist für Version 2.6.0 und höher der Greengrass Nucleus-Komponente verfügbar. In der folgenden Tabelle sind die Mindestversionen von aufgeführt AWS IoT Device SDK , die Sie verwenden müssen, um den Empfangsmodus einzustellen.

SDK	Mindestversion	
AWS IoT Device SDK für Java v2	v1.9.3	
AWS IoT Device SDK für Python v2	v1.11.3	
AWS IoT Device SDK für C++ v2	v1.18.4	
AWS IoT Device SDK für v2 JavaScript	v1.12.0	

Antwort

Die Antwort dieser Operation enthält die folgenden Informationen:

`messages`

Der Nachrichtenstrom. Dieses Objekt,`SubscriptionResponseMessage`, enthält die folgenden Informationen. Jede Nachricht enthält `jsonMessage` oder`binaryMessage`.

`jsonMessage`(Python:`json_message`)

(Optional) Eine JSON-Nachricht. Dieses Objekt,`JsonMessage`, enthält die folgenden Informationen:

`message`

Die JSON-Nachricht als Objekt.

`context`

Der Kontext der Nachricht, z. B. das Thema, in dem die Nachricht veröffentlicht wurde.

Diese Funktion ist für Version 2.6.0 und höher der Greengrass Nucleus-Komponente verfügbar. In der folgenden Tabelle sind die Mindestversionen von aufgeführt AWS IoT Device SDK , die Sie für den Zugriff auf den Nachrichtenkontext verwenden müssen.

SDK	Mindestversion	
AWS IoT Device SDK für Java v2	v1.9.3	
AWS IoT Device SDK für Python v2	v1.11.3	
AWS IoT Device SDK für C ++ v2	v1.18.4	
AWS IoT Device SDK for JavaScript v2	v1.12.0	

> **Note**
>
> Die AWS IoT Greengrass Core-Software verwendet dieselben Nachrichtenobjekte in den Operationen `PublishToTopic` und`SubscribeToTopic`. Die AWS IoT Greengrass Core-Software legt dieses Kontextobjekt in Nachrichten fest, wenn Sie sie abonnieren, und ignoriert dieses Kontextobjekt in Nachrichten, die Sie veröffentlichen.

Dieses Objekt,`MessageContext`, enthält die folgenden Informationen:

`topic`

Das Thema, in dem die Nachricht veröffentlicht wurde.

`binaryMessage`(Python:`binary_message`)

(Optional) Eine binäre Nachricht. Dieses Objekt,`BinaryMessage`, enthält die folgenden Informationen:

`message`

Die binäre Nachricht als Blob.

`context`

Der Kontext der Nachricht, z. B. das Thema, in dem die Nachricht veröffentlicht wurde.

Diese Funktion ist für Version 2.6.0 und höher der Greengrass Nucleus-Komponente verfügbar. In der folgenden Tabelle sind die Mindestversionen von aufgeführt AWS IoT Device SDK , die Sie für den Zugriff auf den Nachrichtenkontext verwenden müssen.

SDK	Mindestversion
AWS IoT Device SDK für Java v2	v1.9.3
AWS IoT Device SDK für Python v2	v1.11.3
AWS IoT Device SDK für C ++ v2	v1.18.4
AWS IoT Device SDK for JavaScript v2	v1.12.0

> **Note**
>
> Die AWS IoT Greengrass Core-Software verwendet dieselben Nachrichtenobjekte in den Operationen `PublishToTopic` und`SubscribeToTopic`. Die AWS IoT Greengrass Core-Software legt dieses Kontextobjekt in Nachrichten fest, wenn Sie sie abonnieren, und ignoriert dieses Kontextobjekt in Nachrichten, die Sie veröffentlichen.

Dieses Objekt,`MessageContext`, enthält die folgenden Informationen:

`topic`

Das Thema, in dem die Nachricht veröffentlicht wurde.

`topicName`(Python:`topic_name`)

Das Thema, zu dem die Nachricht veröffentlicht wurde.

> **Note**
>
> Diese Eigenschaft wird derzeit nicht verwendet. In Greengrass Nucleus v2.6.0 und höher können Sie den `(jsonMessage|binaryMessage).context.topic` Wert von a abrufen, `SubscriptionResponseMessage` um das Thema abzurufen, in dem die Nachricht veröffentlicht wurde.

Beispiele

Die folgenden Beispiele zeigen, wie dieser Vorgang in benutzerdefiniertem Komponentencode aufgerufen wird.

Java (IPC client V2)

Example Beispiel: Lokale Nachrichten zum Publizieren/Abonnieren abonnieren

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.SubscribeToTopicResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.*;

import java.nio.charset.StandardCharsets;
import java.util.Optional;

public class SubscribeToTopicV2 {

    public static void main(String[] args) {
        String topic = args[0];
        try (GreengrassCoreIPCClientV2 ipcClient =
 GreengrassCoreIPCClientV2.builder().build()) {
            SubscribeToTopicRequest request = new
 SubscribeToTopicRequest().withTopic(topic);
            GreengrassCoreIPCClientV2.StreamingResponse<SubscribeToTopicResponse,
                    SubscribeToTopicResponseHandler> response =
                    ipcClient.subscribeToTopic(request,
 SubscribeToTopicV2::onStreamEvent,
                            Optional.of(SubscribeToTopicV2::onStreamError),
                            Optional.of(SubscribeToTopicV2::onStreamClosed));
            SubscribeToTopicResponseHandler responseHandler =
 response.getHandler();
```

```
            System.out.println("Successfully subscribed to topic: " + topic);

            // Keep the main thread alive, or the process will exit.
            try {
                while (true) {
                    Thread.sleep(10000);
                }
            } catch (InterruptedException e) {
                System.out.println("Subscribe interrupted.");
            }

            // To stop subscribing, close the stream.
            responseHandler.closeStream();
        } catch (Exception e) {
            if (e.getCause() instanceof UnauthorizedError) {
                System.err.println("Unauthorized error while publishing to topic: "
+ topic);
            } else {
                System.err.println("Exception occurred when using IPC.");
            }
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static void onStreamEvent(SubscriptionResponseMessage
subscriptionResponseMessage) {
        try {
            BinaryMessage binaryMessage =
subscriptionResponseMessage.getBinaryMessage();
            String message = new String(binaryMessage.getMessage(),
StandardCharsets.UTF_8);
            String topic = binaryMessage.getContext().getTopic();
            System.out.printf("Received new message on topic %s: %s%n", topic,
message);
        } catch (Exception e) {
            System.err.println("Exception occurred while processing subscription
response " +
                    "message.");
            e.printStackTrace();
        }
    }

    public static boolean onStreamError(Throwable error) {
```

```
        System.err.println("Received a stream error.");
        error.printStackTrace();
        return false; // Return true to close stream, false to keep stream open.
    }

    public static void onStreamClosed() {
        System.out.println("Subscribe to topic stream closed.");
    }
}
```

Python (IPC client V2)

Example Beispiel: Lokale Nachrichten zum Veröffentlichen und Abonnieren abonnieren abonnieren

```
import sys
import time
import traceback

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import (
    SubscriptionResponseMessage,
    UnauthorizedError
)


def main():
    args = sys.argv[1:]
    topic = args[0]

    try:
        ipc_client = GreengrassCoreIPCClientV2()
        # Subscription operations return a tuple with the response and the
 operation.
        _, operation = ipc_client.subscribe_to_topic(topic=topic,
 on_stream_event=on_stream_event,

 on_stream_error=on_stream_error, on_stream_closed=on_stream_closed)
        print('Successfully subscribed to topic: ' + topic)

        # Keep the main thread alive, or the process will exit.
        try:
            while True:
                time.sleep(10)
```

```
        except InterruptedError:
            print('Subscribe interrupted.')

        # To stop subscribing, close the stream.
        operation.close()
    except UnauthorizedError:
        print('Unauthorized error while subscribing to topic: ' +
              topic, file=sys.stderr)
        traceback.print_exc()
        exit(1)
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)


def on_stream_event(event: SubscriptionResponseMessage) -> None:
    try:
        message = str(event.binary_message.message, 'utf-8')
        topic = event.binary_message.context.topic
        print('Received new message on topic %s: %s' % (topic, message))
    except:
        traceback.print_exc()


def on_stream_error(error: Exception) -> bool:
    print('Received a stream error.', file=sys.stderr)
    traceback.print_exc()
    return False  # Return True to close stream, False to keep stream open.


def on_stream_closed() -> None:
    print('Subscribe to topic stream closed.')


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

C++

Example Beispiel: Lokale Nachrichten zum Veröffentlichen und Abonnieren abonnieren abonnieren

```
#include <iostream>
```

```
#include </crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class SubscribeResponseHandler : public SubscribeToTopicStreamHandler {
    public:
        virtual ~SubscribeResponseHandler() {}

    private:
        void OnStreamEvent(SubscriptionResponseMessage *response) override {
            auto jsonMessage = response->GetJsonMessage();
            if (jsonMessage.has_value() &&
 jsonMessage.value().GetMessage().has_value()) {
                auto messageString =
 jsonMessage.value().GetMessage().value().View().WriteReadable();
                // Handle JSON message.
            } else {
                auto binaryMessage = response->GetBinaryMessage();
                if (binaryMessage.has_value() &&
 binaryMessage.value().GetMessage().has_value()) {
                    auto messageBytes = binaryMessage.value().GetMessage().value();
                    std::string messageString(messageBytes.begin(),
 messageBytes.end());
                    // Handle binary message.
                }
            }
        }

        bool OnStreamError(OperationError *error) override {
            // Handle error.
            return false; // Return true to close stream, false to keep stream open.
        }

        void OnStreamClosed() override {
            // Handle close.
        }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
```

```
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};

int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String topic("my/topic");
    int timeout = 10;

    SubscribeToTopicRequest request;
    request.SetTopic(topic);

    //SubscribeResponseHandler streamHandler;
    auto streamHandler = MakeShared<SubscribeResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToTopic(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
 Core." << std::endl;
        exit(-1);
```

```
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.
        } else {
            // Handle RPC error.
        }
        exit(-1);
    }

    // Keep the main thread alive, or the process will exit.
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
    }

    operation->Close();
    return 0;
}
```

JavaScript

Example Beispiel: Lokale Nachrichten zum Veröffentlichen und Abonnieren abonnieren abonnieren

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {SubscribeToTopicRequest, SubscriptionResponseMessage} from "aws-iot-device-
sdk-v2/dist/greengrasscoreipc/model";
import {RpcError} from "aws-iot-device-sdk-v2/dist/eventstream_rpc";

class SubscribeToTopic {
    private ipcClient : greengrasscoreipc.Client
    private readonly topic : string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.subscribeToTopic().then(r => console.log("Started workflow"));
```

```
    }

    private async subscribeToTopic() {
        try {
            this.ipcClient = await getIpcClient();

            const subscribeToTopicRequest : SubscribeToTopicRequest = {
                topic: this.topic,
            }

            const streamingOperation =
 this.ipcClient.subscribeToTopic(subscribeToTopicRequest, undefined); //
 conditionally apply options

            streamingOperation.on("message", (message: SubscriptionResponseMessage)
 => {
                // parse the message depending on your use cases, e.g.
                if(message.binaryMessage && message.binaryMessage.message) {
                    const receivedMessage =
 message.binaryMessage?.message.toString();
                }
            });

            streamingOperation.on("streamError", (error : RpcError) => {
                // define your own error handling logic
            })

            streamingOperation.on("ended", () => {
                // define your own logic
            })

            await streamingOperation.activate();

            // Keep the main thread alive, or the process will exit.
            await new Promise((resolve) => setTimeout(resolve, 10000))
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}

export async function getIpcClient(){
    try {
```

```
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

// starting point
const subscribeToTopic = new SubscribeToTopic();
```

Beispiele

Anhand der folgenden Beispiele erfahren Sie, wie Sie den Dienst Publish/Subscribe IPC in Ihren Komponenten verwenden können.

Beispiel für Publish/Subscribe Publisher (Java, IPC-Client V1)

Das folgende Beispielrezept ermöglicht es der Komponente, zu allen Themen zu veröffentlichen.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubPublisherJava",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubPublisherJava:pubsub:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
```

```
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "java -jar {artifacts:path}/PubSubPublisher.jar"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubPublisherJava
ComponentVersion: '1.0.0'
ComponentDescription: A component that publishes messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        'com.example.PubSubPublisherJava:pubsub:1':
          policyDescription: Allows access to publish to all topics.
          operations:
            - 'aws.greengrass#PublishToTopic'
          resources:
            - '*'
Manifests:
  - Lifecycle:
      run: |-
        java -jar {artifacts:path}/PubSubPublisher.jar
```

Die folgende Java-Beispielanwendung zeigt, wie der Publish/Subscribe-IPC-Dienst verwendet wird, um Nachrichten in anderen Komponenten zu veröffentlichen.

```
/* Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: Apache-2.0 */

package com.example.ipc.pubsub;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.model.*;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class PubSubPublisher {

    public static void main(String[] args) {
        String message = "Hello from the pub/sub publisher (Java).";
        String topic = "test/topic/java";

        try (EventStreamRPCConnection eventStreamRPCConnection =
 IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient = new
 GreengrassCoreIPCClient(eventStreamRPCConnection);

            while (true) {
                PublishToTopicRequest publishRequest = new PublishToTopicRequest();
                PublishMessage publishMessage = new PublishMessage();
                BinaryMessage binaryMessage = new BinaryMessage();
                binaryMessage.setMessage(message.getBytes(StandardCharsets.UTF_8));
                publishMessage.setBinaryMessage(binaryMessage);
                publishRequest.setPublishMessage(publishMessage);
                publishRequest.setTopic(topic);
                CompletableFuture<PublishToTopicResponse> futureResponse = ipcClient
                        .publishToTopic(publishRequest,
 Optional.empty()).getResponse();

                try {
                    futureResponse.get(10, TimeUnit.SECONDS);
                    System.out.println("Successfully published to topic: " + topic);
                } catch (TimeoutException e) {
```

```
                    System.err.println("Timeout occurred while publishing to topic: " +
 topic);
                } catch (ExecutionException e) {
                    if (e.getCause() instanceof UnauthorizedError) {
                        System.err.println("Unauthorized error while publishing to
 topic: " + topic);
                    } else {
                        System.err.println("Execution exception while publishing to
 topic: " + topic);
                    }
                    throw e;
                }
                Thread.sleep(5000);
            }
        } catch (InterruptedException e) {
            System.out.println("Publisher interrupted.");
        } catch (Exception e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

Beispiel für einen Publish/Subscribe-Abonnenten (Java, IPC-Client V1)

Das folgende Beispielrezept ermöglicht es der Komponente, alle Themen zu abonnieren.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubSubscriberJava",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubSubscriberJava:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
```

```
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "java -jar {artifacts:path}/PubSubSubscriber.jar"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubSubscriberJava
ComponentVersion: '1.0.0'
ComponentDescription: A component that subscribes to messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        'com.example.PubSubSubscriberJava:pubsub:1':
          policyDescription: Allows access to subscribe to all topics.
          operations:
            - 'aws.greengrass#SubscribeToTopic'
          resources:
            - '*'
Manifests:
  - Lifecycle:
      run: |-
        java -jar {artifacts:path}/PubSubSubscriber.jar
```

Die folgende Java-Beispielanwendung zeigt, wie der Publish/Subscribe-IPC-Dienst verwendet wird, um Nachrichten für andere Komponenten zu abonnieren.

```
/* Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: Apache-2.0 */

package com.example.ipc.pubsub;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.SubscribeToTopicResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.SubscribeToTopicRequest;
import software.amazon.awssdk.aws.greengrass.model.SubscribeToTopicResponse;
import software.amazon.awssdk.aws.greengrass.model.SubscriptionResponseMessage;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;
import software.amazon.awssdk.eventstreamrpc.StreamResponseHandler;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class PubSubSubscriber {

    public static void main(String[] args) {
        String topic = "test/topic/java";

        try (EventStreamRPCConnection eventStreamRPCConnection =
 IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient = new
 GreengrassCoreIPCClient(eventStreamRPCConnection);

            SubscribeToTopicRequest subscribeRequest = new SubscribeToTopicRequest();
            subscribeRequest.setTopic(topic);
            SubscribeToTopicResponseHandler operationResponseHandler = ipcClient
                    .subscribeToTopic(subscribeRequest, Optional.of(new
 SubscribeResponseHandler()));
            CompletableFuture<SubscribeToTopicResponse> futureResponse =
 operationResponseHandler.getResponse();

            try {
```

```
                futureResponse.get(10, TimeUnit.SECONDS);
                System.out.println("Successfully subscribed to topic: " + topic);
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while subscribing to topic: " +
topic);
                throw e;
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.println("Unauthorized error while subscribing to topic:
" + topic);
                } else {
                    System.err.println("Execution exception while subscribing to topic:
" + topic);
                }
                throw e;
            }

            // Keep the main thread alive, or the process will exit.
            try {
                while (true) {
                    Thread.sleep(10000);
                }
            } catch (InterruptedException e) {
                System.out.println("Subscribe interrupted.");
            }
        } catch (Exception e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    private static class SubscribeResponseHandler implements
StreamResponseHandler<SubscriptionResponseMessage> {

        @Override
        public void onStreamEvent(SubscriptionResponseMessage
subscriptionResponseMessage) {
            try {
                String message = new
String(subscriptionResponseMessage.getBinaryMessage()
                        .getMessage(), StandardCharsets.UTF_8);
                System.out.println("Received new message: " + message);
            } catch (Exception e) {
```

```
                e.printStackTrace();
            }
        }

        @Override
        public boolean onStreamError(Throwable error) {
            System.err.println("Received a stream error.");
            error.printStackTrace();
            return false; // Return true to close stream, false to keep stream open.
        }

        @Override
        public void onStreamClosed() {
            System.out.println("Subscribe to topic stream closed.");
        }
    }
}
```

Beispiel für Publish/Subscribe Publisher (Python, IPC-Client V1)

Das folgende Beispielrezept ermöglicht es der Komponente, zu allen Themen zu veröffentlichen.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubPublisherPython",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubPublisherPython:pubsub:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
```

```
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk",
        "run": "python3 -u {artifacts:path}/pubsub_publisher.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk",
        "run": "py -3 -u {artifacts:path}/pubsub_publisher.py"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubPublisherPython
ComponentVersion: 1.0.0
ComponentDescription: A component that publishes messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        com.example.PubSubPublisherPython:pubsub:1:
          policyDescription: Allows access to publish to all topics.
          operations:
            - aws.greengrass#PublishToTopic
          resources:
            - "*"
```

```
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: python3 -m pip install --user awsiotsdk
      run: python3 -u {artifacts:path}/pubsub_publisher.py
  - Platform:
      os: windows
    Lifecycle:
      install: py -3 -m pip install --user awsiotsdk
      run: py -3 -u {artifacts:path}/pubsub_publisher.py
```

Die folgende Python-Beispielanwendung zeigt, wie der Publish/Subscribe-IPC-Dienst verwendet wird, um Nachrichten in anderen Komponenten zu veröffentlichen.

```
import concurrent.futures
import sys
import time
import traceback

import awsiot.greengrasscoreipc
from awsiot.greengrasscoreipc.model import (
    PublishToTopicRequest,
    PublishMessage,
    BinaryMessage,
    UnauthorizedError
)


topic = "test/topic/python"
message = "Hello from the pub/sub publisher (Python)."
TIMEOUT = 10

try:
    ipc_client = awsiot.greengrasscoreipc.connect()

    while True:
        request = PublishToTopicRequest()
        request.topic = topic
        publish_message = PublishMessage()
        publish_message.binary_message = BinaryMessage()
        publish_message.binary_message.message = bytes(message, "utf-8")
```

```
        request.publish_message = publish_message
        operation = ipc_client.new_publish_to_topic()
        operation.activate(request)
        future_response = operation.get_response()

        try:
            future_response.result(TIMEOUT)
            print('Successfully published to topic: ' + topic)
        except concurrent.futures.TimeoutError:
            print('Timeout occurred while publishing to topic: ' + topic,
 file=sys.stderr)
        except UnauthorizedError as e:
            print('Unauthorized error while publishing to topic: ' + topic,
 file=sys.stderr)
            raise e
        except Exception as e:
            print('Exception while publishing to topic: ' + topic, file=sys.stderr)
            raise e
        time.sleep(5)
except InterruptedError:
    print('Publisher interrupted.')
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

Beispiel für Abonnent zum Veröffentlichen/Abonnieren (Python, IPC-Client V1)

Das folgende Beispielrezept ermöglicht es der Komponente, alle Themen zu abonnieren.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubSubscriberPython",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubSubscriberPython:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
```

```
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk",
        "run": "python3 -u {artifacts:path}/pubsub_subscriber.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk",
        "run": "py -3 -u {artifacts:path}/pubsub_subscriber.py"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubSubscriberPython
ComponentVersion: 1.0.0
ComponentDescription: A component that subscribes to messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
```

```
    accessControl:
      aws.greengrass.ipc.pubsub:
        com.example.PubSubSubscriberPython:pubsub:1:
          policyDescription: Allows access to subscribe to all topics.
          operations:
            - aws.greengrass#SubscribeToTopic
          resources:
            - "*"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: python3 -m pip install --user awsiotsdk
      run: python3 -u {artifacts:path}/pubsub_subscriber.py
  - Platform:
      os: windows
    Lifecycle:
      install: py -3 -m pip install --user awsiotsdk
      run: py -3 -u {artifacts:path}/pubsub_subscriber.py
```

Die folgende Python-Beispielanwendung zeigt, wie der Publish/Subscribe-IPC-Dienst verwendet wird, um Nachrichten für andere Komponenten zu abonnieren.

```
import concurrent.futures
import sys
import time
import traceback

import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import (
    SubscribeToTopicRequest,
    SubscriptionResponseMessage,
    UnauthorizedError
)

topic = "test/topic/python"
TIMEOUT = 10


class StreamHandler(client.SubscribeToTopicStreamHandler):
    def __init__(self):
```

```
        super().__init__()

    def on_stream_event(self, event: SubscriptionResponseMessage) -> None:
        try:
            message = str(event.binary_message.message, "utf-8")
            print("Received new message: " + message)
        except:
            traceback.print_exc()

    def on_stream_error(self, error: Exception) -> bool:
        print("Received a stream error.", file=sys.stderr)
        traceback.print_exc()
        return False  # Return True to close stream, False to keep stream open.

    def on_stream_closed(self) -> None:
        print('Subscribe to topic stream closed.')


try:
    ipc_client = awsiot.greengrasscoreipc.connect()

    request = SubscribeToTopicRequest()
    request.topic = topic
    handler = StreamHandler()
    operation = ipc_client.new_subscribe_to_topic(handler)
    operation.activate(request)
    future_response = operation.get_response()

    try:
        future_response.result(TIMEOUT)
        print('Successfully subscribed to topic: ' + topic)
    except concurrent.futures.TimeoutError as e:
        print('Timeout occurred while subscribing to topic: ' + topic,
 file=sys.stderr)
        raise e
    except UnauthorizedError as e:
        print('Unauthorized error while subscribing to topic: ' + topic,
 file=sys.stderr)
        raise e
    except Exception as e:
        print('Exception while subscribing to topic: ' + topic, file=sys.stderr)
        raise e

    # Keep the main thread alive, or the process will exit.
```

```
    try:
        while True:
            time.sleep(10)
    except InterruptedError:
        print('Subscribe interrupted.')
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

Beispiel für Publish/Subscribe Publisher (C++)

Das folgende Beispielrezept ermöglicht es der Komponente, in allen Themen zu veröffentlichen.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubPublisherCpp",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubPublisherCpp:pubsub:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "{artifacts:path}/greengrassv2_pubsub_publisher"
      },
```

```
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.PubSubPublisherCpp/1.0.0/greengrassv2_pubsub_publisher",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubPublisherCpp
ComponentVersion: 1.0.0
ComponentDescription: A component that publishes messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        com.example.PubSubPublisherCpp:pubsub:1:
          policyDescription: Allows access to publish to all topics.
          operations:
            - aws.greengrass#PublishToTopic
          resources:
            - "*"
Manifests:
  - Lifecycle:
      run: "{artifacts:path}/greengrassv2_pubsub_publisher"
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.PubSubPublisherCpp/1.0.0/
greengrassv2_pubsub_publisher
        Permission:
          Execute: OWNER
```

Die folgende C++-Beispielanwendung zeigt, wie der IPC-Dienst Publish/Subscribe verwendet wird, um Nachrichten in anderen Komponenten zu veröffentlichen.

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() << std::endl;
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
};

int main() {
    String message("Hello from the pub/sub publisher (C++).");
    String topic("test/topic/cpp");
    int timeout = 10;

    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    while (true) {
```

```
        PublishToTopicRequest request;
        Vector<uint8_t> messageData({message.begin(), message.end()});
        BinaryMessage binaryMessage;
        binaryMessage.SetMessage(messageData);
        PublishMessage publishMessage;
        publishMessage.SetBinaryMessage(binaryMessage);
        request.SetTopic(topic);
        request.SetPublishMessage(publishMessage);

        auto operation = ipcClient.NewPublishToTopic();
        auto activate = operation->Activate(request, nullptr);
        activate.wait();

        auto responseFuture = operation->GetResult();
        if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
            std::cerr << "Operation timed out while waiting for response from
 Greengrass Core." << std::endl;
            exit(-1);
        }

        auto response = responseFuture.get();
        if (response) {
            std::cout << "Successfully published to topic: " << topic << std::endl;
        } else {
            // An error occurred.
            std::cout << "Failed to publish to topic: " << topic << std::endl;
            auto errorType = response.GetResultType();
            if (errorType == OPERATION_ERROR) {
                auto *error = response.GetOperationError();
                std::cout << "Operation error: " << error->GetMessage().value() <<
 std::endl;
            } else {
                std::cout << "RPC error: " << response.GetRpcError() << std::endl;
            }
            exit(-1);
        }

        std::this_thread::sleep_for(std::chrono::seconds(5));
    }

    return 0;
}
```

Beispiel für einen Publish/Subscribe-Abonnenten (C++)

Das folgende Beispielrezept ermöglicht es der Komponente, alle Themen zu abonnieren.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubSubscriberCpp",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubSubscriberCpp:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "{artifacts:path}/greengrassv2_pub_sub_subscriber"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pub_sub_subscriber",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
```

```
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubSubscriberCpp
ComponentVersion: 1.0.0
ComponentDescription: A component that subscribes to messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        com.example.PubSubSubscriberCpp:pubsub:1:
          policyDescription: Allows access to subscribe to all topics.
          operations:
            - aws.greengrass#SubscribeToTopic
          resources:
            - "*"
Manifests:
  - Lifecycle:
      run: "{artifacts:path}/greengrassv2_pub_sub_subscriber"
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pub_sub_subscriber
        Permission:
          Execute: OWNER
```

Die folgende C++-Beispielanwendung zeigt, wie der Publish/Subscribe-IPC-Dienst verwendet wird, um Nachrichten für andere Komponenten zu abonnieren.

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class SubscribeResponseHandler : public SubscribeToTopicStreamHandler {
```

```
    public:
        virtual ~SubscribeResponseHandler() {}

    private:
        void OnStreamEvent(SubscriptionResponseMessage *response) override {
            auto jsonMessage = response->GetJsonMessage();
            if (jsonMessage.has_value() &&
 jsonMessage.value().GetMessage().has_value()) {
                auto messageString =
 jsonMessage.value().GetMessage().value().View().WriteReadable();
                std::cout << "Received new message: " << messageString << std::endl;
            } else {
                auto binaryMessage = response->GetBinaryMessage();
                if (binaryMessage.has_value() &&
 binaryMessage.value().GetMessage().has_value()) {
                    auto messageBytes = binaryMessage.value().GetMessage().value();
                    std::string messageString(messageBytes.begin(),
 messageBytes.end());
                    std::cout << "Received new message: " << messageString <<
 std::endl;
                }
            }
        }

        bool OnStreamError(OperationError *error) override {
            std::cout << "Received an operation error: ";
            if (error->GetMessage().has_value()) {
                std::cout << error->GetMessage().value();
            }
            std::cout << std::endl;
            return false; // Return true to close stream, false to keep stream open.
        }

        void OnStreamClosed() override {
            std::cout << "Subscribe to topic stream closed." << std::endl;
        }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
```

```
        std::cout << "OnDisconnectCallback: " << error.StatusToString() << std::endl;
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
};

int main() {
    String topic("test/topic/cpp");
    int timeout = 10;

    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    SubscribeToTopicRequest request;
    request.SetTopic(topic);
    auto streamHandler = MakeShared<SubscribeResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToTopic(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
 Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (response) {
```

```
        std::cout << "Successfully subscribed to topic: " << topic << std::endl;
    } else {
        // An error occurred.
        std::cout << "Failed to subscribe to topic: " << topic << std::endl;
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            std::cout << "Operation error: " << error->GetMessage().value() <<
 std::endl;
        } else {
            std::cout << "RPC error: " << response.GetRpcError() << std::endl;
        }
        exit(-1);
    }

    // Keep the main thread alive, or the process will exit.
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
    }

    operation->Close();
    return 0;
}
```

MQTT-Nachrichten veröffentlichen/abonnieren AWS IoT Core

Mit dem AWS IoT Core MQTT-Messaging-IPC-Dienst können Sie MQTT-Nachrichten an und von ihnen senden und empfangen. AWS IoT Core Komponenten können Nachrichten veröffentlichen AWS IoT Core und Themen abonnieren, um auf MQTT-Nachrichten aus anderen Quellen zu reagieren. Weitere Informationen zur AWS IoT Core Implementierung von MQTT finden Sie unter MQTT im AWS IoT Core Developer Guide.

> **Note**
>
> Mit diesem MQTT-Messaging-IPC-Dienst können Sie Nachrichten austauschen mit. AWS IoT Core Weitere Hinweise zum Austausch von Nachrichten zwischen Komponenten finden Sie unter. Lokale Nachrichten veröffentlichen/abonnieren

Themen

Minimale SDK-Versionen

In der folgenden Tabelle sind die Mindestversionen von aufgeführt AWS IoT Device SDK , die Sie zum Veröffentlichen und Abonnieren von AWS IoT Core MQTT-Nachrichten verwenden müssen.

SDK	Mindestversion	
AWS IoT Device SDK für Java v2	v1.2.10	
AWS IoT Device SDK für Python v2	v1.5.3	
AWS IoT Device SDK für C++ v2	v1.17.0	
AWS IoT Device SDK für v2 JavaScript	v1.12.0	

Autorisierung

Um AWS IoT Core MQTT-Messaging in einer benutzerdefinierten Komponente zu verwenden, müssen Sie Autorisierungsrichtlinien definieren, die es Ihrer Komponente ermöglichen, Nachrichten zu Themen zu senden und zu empfangen. Informationen zur Definition von Autorisierungsrichtlinien finden Sie unterAutorisieren Sie Komponenten zur Ausführung von IPC-Vorgängen.

Autorisierungsrichtlinien für AWS IoT Core MQTT-Messaging haben die folgenden Eigenschaften.

IPC-Dienst-ID: `aws.greengrass.ipc.mqttproxy`

Operation	Beschreibung	Ressourcen
`aws.greengrass#PublishToIoTCore`	Ermöglicht einer Komponente, Nachrichten zu den von Ihnen AWS IoT Core angegebenen MQTT-Themen zu veröffentlichen.	Eine Themenzeichenfolge, z. B. oder `test/topic` , `*` um den Zugriff auf alle Themen zu ermöglichen. Sie können Platzhalter (#und+) für MQTT-Themen verwenden, um mehrere Ressourcen zuzuordnen.
`aws.greengrass#SubscribeToIoTCore`	Ermöglicht einer Komponente, Nachrichten zu den von Ihnen AWS IoT Core angegebenen Themen zu abonnieren.	Eine Themenzeichenfolge, z. B. oder `test/topic` , `*` um den Zugriff auf alle Themen zu ermöglichen. Sie können Platzhalter (#und+) für MQTT-Themen verwenden, um mehrere Ressourcen zuzuordnen.
*	Ermöglicht einer Komponente, AWS IoT Core MQTT-Nachrichten für die von Ihnen angegebenen Themen zu veröffentlichen und zu abonnieren.	Eine Themenzeichenfolge, z. B. oder `test/topic` , `*` um den Zugriff auf alle Themen zu ermöglichen. Sie können Platzhalter (#und+) für MQTT-Themen verwenden, um mehrere Ressourcen zuzuordnen.

MQTT-Platzhalter in MQTT-Autorisierungsrichtlinien AWS IoT Core

Sie können MQTT-Platzhalter in AWS IoT Core MQTT-IPC-Autorisierungsrichtlinien verwenden. Komponenten können Themen veröffentlichen und abonnieren, die dem Themenfilter entsprechen, den Sie in einer Autorisierungsrichtlinie zulassen. Wenn beispielsweise die Autorisierungsrichtlinie einer Komponente Zugriff auf gewährt`test/topic/#`, kann die Komponente diese abonnieren`test/topic/#`, veröffentlichen und abonnieren`test/topic/filter`.

Rezeptvariablen in AWS IoT Core MQTT-Autorisierungsrichtlinien

Wenn Sie Version 2.6.0 oder höher von Greengrass Nucleus verwenden, können Sie die `{iot:thingName}` Rezeptvariable in Autorisierungsrichtlinien verwenden. Mit dieser Funktion können Sie eine einzige Autorisierungsrichtlinie für eine Gruppe von Kerngeräten konfigurieren, sodass jedes Kerngerät nur auf Themen zugreifen kann, die seinen eigenen Namen enthalten. Sie können einer Komponente beispielsweise Zugriff auf die folgende Themenressource gewähren.

```
devices/{iot:thingName}/messages
```

Weitere Informationen finden Sie unter Rezeptvariablen und Verwenden Sie Rezeptvariablen bei Merge-Updates.

Beispiele für Autorisierungsrichtlinien

Anhand der folgenden Beispiele für Autorisierungsrichtlinien können Sie Autorisierungsrichtlinien für Ihre Komponenten konfigurieren.

Example Beispiel für eine Autorisierungsrichtlinie mit uneingeschränktem Zugriff

Das folgende Beispiel für eine Autorisierungsrichtlinie ermöglicht es einer Komponente, alle Themen zu veröffentlichen und zu abonnieren.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "com.example.MyIoTCorePubSubComponent:mqttproxy:1": {
        "policyDescription": "Allows access to publish/subscribe to all topics.",
        "operations": [
          "aws.greengrass#PublishToIoTCore",
          "aws.greengrass#SubscribeToIoTCore"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

YAML

```
---
accessControl:
  aws.greengrass.ipc.mqttproxy:
    com.example.MyIoTCorePubSubComponent:mqttproxy:1:
      policyDescription: Allows access to publish/subscribe to all topics.
      operations:
        - aws.greengrass#PublishToIoTCore
        - aws.greengrass#SubscribeToIoTCore
      resources:
        - "*"
```

Example Beispiel für eine Autorisierungsrichtlinie mit eingeschränktem Zugriff

Das folgende Beispiel für eine Autorisierungsrichtlinie ermöglicht es einer Komponente, zwei Themen mit dem Namen und zu veröffentlichen `factory/1/events` und zu abonnieren`factory/1/actions`.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "com.example.MyIoTCorePubSubComponent:mqttproxy:1": {
        "policyDescription": "Allows access to publish/subscribe to factory 1
 topics.",
        "operations": [
          "aws.greengrass#PublishToIoTCore",
          "aws.greengrass#SubscribeToIoTCore"
        ],
        "resources": [
          "factory/1/actions",
          "factory/1/events"
        ]
      }
    }
  }
}
```

YAML

```
---
accessControl:
  aws.greengrass.ipc.mqttproxy:
    "com.example.MyIoTCorePubSubComponent:mqttproxy:1":
      policyDescription: Allows access to publish/subscribe to factory 1 topics.
      operations:
        - aws.greengrass#PublishToIoTCore
        - aws.greengrass#SubscribeToIoTCore
      resources:
        - factory/1/actions
        - factory/1/events
```

Example Beispiel für eine Autorisierungsrichtlinie für eine Gruppe von Kerngeräten

> **Important**
>
> In diesem Beispiel wird eine Funktion verwendet, die für Version 2.6.0 und höher der Greengrass Nucleus-Komponente verfügbar ist. Greengrass Nucleus v2.6.0 bietet Unterstützung für die meisten Rezeptvariablen, z. B. in `{iot:thingName}` Komponentenkonfigurationen.

Das folgende Beispiel für eine Autorisierungsrichtlinie ermöglicht es einer Komponente, ein Thema zu veröffentlichen und zu abonnieren, das den Namen des Kerngeräts enthält, auf dem die Komponente ausgeführt wird.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "com.example.MyIoTCorePubSubComponent:mqttproxy:1": {
        "policyDescription": "Allows access to publish/subscribe to all topics.",
        "operations": [
          "aws.greengrass#PublishToIoTCore",
          "aws.greengrass#SubscribeToIoTCore"
        ],
        "resources": [
```

```
          "factory/1/devices/{iot:thingName}/controls"
        ]
      }
    }
  }
}
```

YAML

```
---
accessControl:
  aws.greengrass.ipc.mqttproxy:
    "com.example.MyIoTCorePubSubComponent:mqttproxy:1":
      policyDescription: Allows access to publish/subscribe to all topics.
      operations:
        - aws.greengrass#PublishToIoTCore
        - aws.greengrass#SubscribeToIoTCore
      resources:
        - factory/1/devices/{iot:thingName}/controls
```

PublishToIoTCore

Veröffentlicht eine MQTT-Nachricht zu einem AWS IoT Core Thema.

Wenn Sie MQTT-Nachrichten veröffentlichen AWS IoT Core, gilt ein Kontingent von 100 Transaktionen pro Sekunde. Wenn Sie dieses Kontingent überschreiten, werden Nachrichten auf dem Greengrass-Gerät zur Verarbeitung in die Warteschlange gestellt. Außerdem gibt es ein Kontingent von 512 KB an Daten pro Sekunde und ein kontoweites Kontingent von 20.000 Veröffentlichungen pro Sekunde (2.000 in einigen Fällen). AWS-Regionen Weitere Informationen zu den Grenzwerten für den MQTT-Nachrichtenbroker finden Sie unter Grenzwerte und Kontingente für AWS IoT Core Nachrichtenbroker und Protokolle. AWS IoT Core

Wenn Sie diese Kontingente überschreiten, beschränkt das Greengrass-Gerät die Veröffentlichung von Nachrichten auf AWS IoT Core. Nachrichten werden in einem Spooler im Arbeitsspeicher gespeichert. Standardmäßig beträgt der dem Spooler zugewiesene Speicher 2,5 MB. Wenn der Spooler voll ist, werden neue Nachrichten zurückgewiesen. Sie können den Spooler vergrößern. Weitere Informationen finden Sie unter Konfiguration in der Grüngraskern-Dokumentation. Um zu vermeiden, dass der Spooler voll wird und der zugewiesene Speicher vergrößert werden muss, sollten Sie Veröffentlichungsanforderungen auf nicht mehr als 100 Anfragen pro Sekunde beschränken.

Wenn Ihre Anwendung Nachrichten mit einer höheren Geschwindigkeit oder größere Nachrichten senden muss, sollten Sie die Stream-Manager zum Senden von Nachrichten an Kinesis Data Streams verwenden. Die Stream Manager-Komponente ist für die Übertragung großer Datenmengen an den konzipiert. AWS Cloud Weitere Informationen finden Sie unter Verwalten von Datenströmen auf Greengrass-Core-Geräten.

Anforderung

Die Anforderung dieses Vorgangs hat die folgenden Parameter:

`topicName`(Python:`topic_name`)

Das Thema, zu dem die Nachricht veröffentlicht werden soll.

`qos`

Die zu verwendende MQTT-QoS. Diese Aufzählung,`QOS`, hat die folgenden Werte:

- `AT_MOST_ONCE`— QoS 0. Die MQTT-Nachricht wird höchstens einmal zugestellt.
- `AT_LEAST_ONCE`— QoS 1. Die MQTT-Nachricht wird mindestens einmal zugestellt.

`payload`

(Optional) Die Nutzlast der Nachricht als Blob.

Die folgenden Funktionen sind für Version 2.10.0 und höher verfügbar, Grüngraskern wenn Sie MQTT 5 verwenden. Diese Funktionen werden ignoriert, wenn Sie MQTT 3.1.1 verwenden. In der folgenden Tabelle ist die Mindestversion des AWS IoT Geräte-SDK aufgeführt, die Sie für den Zugriff auf diese Funktionen verwenden müssen.

SDK	Mindestversion
AWS IoT-Geräte-SDK for Python v2	v1.15.0
AWS IoT Device SDK for Java v2	v1.13.0
AWS IoT Device SDK for C++ v2	v1.24,0
AWS IoT Device SDK for JavaScript v2	v1.13.0

`payloadFormat`

(Optional) Das Format der Nachrichtennutzlast. Wenn Sie den nicht festlegen`payloadFormat`, wird davon ausgegangen, dass `BYTES` der Typ Die Aufzählung hat die folgenden Werte:

- `BYTES`— Der Inhalt der Payload ist ein binärer Blob.
- `UTF8`— Der Inhalt der Payload ist eine UTF8-Zeichenfolge.

`retain`

(Optional) Gibt an, ob die MQTT-Aufbewahrungsoption beim Veröffentlichen auf gesetzt werden soll. `true`

`userProperties`

(Optional) Eine Liste von anwendungsspezifischen `UserProperty` Objekten, die gesendet werden sollen. Das `UserProperty` Objekt ist wie folgt definiert:

```
UserProperty:
  key: string
  value: string
```

`messageExpiryIntervalSeconds`

(Optional) Die Anzahl der Sekunden, bevor die Nachricht abläuft und vom Server gelöscht wird. Wenn dieser Wert nicht festgelegt ist, läuft die Nachricht nicht ab.

`correlationData`

(Optional) Der Anfrage hinzugefügte Informationen, die verwendet werden können, um eine Anfrage mit einer Antwort zu verknüpfen.

`responseTopic`

(Optional) Das Thema, das für die Antwortnachricht verwendet werden soll.

`contentType`

(Optional) Eine anwendungsspezifische Kennung für den Inhaltstyp der Nachricht.

Antwort

Dieser Vorgang liefert in seiner Antwort keine Informationen.

Beispiele

Die folgenden Beispiele zeigen, wie dieser Vorgang in benutzerdefiniertem Komponentencode aufgerufen wird.

Java (IPC client V2)

Example Beispiel: Veröffentlichen Sie eine Nachricht

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.model.PublishToIoTCoreRequest;
import software.amazon.awssdk.aws.greengrass.model.QOS;
import java.nio.charset.StandardCharsets;

public class PublishToIoTCore {

    public static void main(String[] args) {
        String topic = args[0];
        String message = args[1];
        QOS qos = QOS.get(args[2]);

        try (GreengrassCoreIPCClientV2 ipcClientV2 =
 GreengrassCoreIPCClientV2.builder().build()) {
            ipcClientV2.publishToIoTCore(new PublishToIoTCoreRequest()
                    .withTopicName(topic)
                    .withPayload(message.getBytes(StandardCharsets.UTF_8))
                    .withQos(qos));
            System.out.println("Successfully published to topic: " + topic);
        } catch (Exception e) {
            System.err.println("Exception occurred.");
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

Python (IPC client V2)

Example Beispiel: Veröffentlichen Sie eine Nachricht

> **Note**
>
> In diesem Beispiel wird davon ausgegangen, dass Sie Version 1.5.4 oder höher von AWS IoT Device SDK für Python v2 verwenden.

```
import awsiot.greengrasscoreipc.clientv2 as clientV2

topic = 'my/topic'
qos = '1'
payload = 'Hello, World'

ipc_client = clientV2.GreengrassCoreIPCClientV2()
resp = ipc_client.publish_to_iot_core(topic_name=topic, qos=qos, payload=payload)
ipc_client.close()
```

Java (IPC client V1)

Example Beispiel: Veröffentlichen Sie eine Nachricht

> **Note**
>
> In diesem Beispiel wird eine `IPCUtils` Klasse verwendet, um eine Verbindung zum AWS IoT Greengrass Core-IPC-Dienst herzustellen. Weitere Informationen finden Sie unter Connect zum AWS IoT Greengrass Core IPC-Dienst her.

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.PublishToIoTCoreResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.PublishToIoTCoreRequest;
import software.amazon.awssdk.aws.greengrass.model.PublishToIoTCoreResponse;
import software.amazon.awssdk.aws.greengrass.model.QOS;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;
```

```
import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class PublishToIoTCore {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        String topic = args[0];
        String message = args[1];
        QOS qos = QOS.get(args[2]);
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            PublishToIoTCoreResponseHandler responseHandler =
                    PublishToIoTCore.publishBinaryMessageToTopic(ipcClient, topic,
 message, qos);
            CompletableFuture<PublishToIoTCoreResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                System.out.println("Successfully published to topic: " + topic);
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while publishing to topic: " +
 topic);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.println("Unauthorized error while publishing to
 topic: " + topic);
                } else {
                    throw e;
                }
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
```

```
            System.exit(1);
        }
    }

    public static PublishToIoTCoreResponseHandler
 publishBinaryMessageToTopic(GreengrassCoreIPCClient greengrassCoreIPCClient, String
 topic, String message, QOS qos) {
        PublishToIoTCoreRequest publishToIoTCoreRequest = new
 PublishToIoTCoreRequest();
        publishToIoTCoreRequest.setTopicName(topic);

 publishToIoTCoreRequest.setPayload(message.getBytes(StandardCharsets.UTF_8));
        publishToIoTCoreRequest.setQos(qos);
        return greengrassCoreIPCClient.publishToIoTCore(publishToIoTCoreRequest,
 Optional.empty());
    }
}
```

Python (IPC client V1)

Example Beispiel: Veröffentlichen Sie eine Nachricht

> **Note**
>
> In diesem Beispiel wird davon ausgegangen, dass Sie Version 1.5.4 oder höher von AWS IoT Device SDK für Python v2 verwenden.

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import (
    QOS,
    PublishToIoTCoreRequest
)

TIMEOUT = 10

ipc_client = awsiot.greengrasscoreipc.connect()

topic = "my/topic"
message = "Hello, World"
qos = QOS.AT_LEAST_ONCE
```

```
request = PublishToIoTCoreRequest()
request.topic_name = topic
request.payload = bytes(message, "utf-8")
request.qos = qos
operation = ipc_client.new_publish_to_iot_core()
operation.activate(request)
future_response = operation.get_response()
future_response.result(TIMEOUT)
```

C++

Example Beispiel: Veröffentlichen Sie eine Nachricht

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};

int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
```

```
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String message("Hello, World!");
    String topic("my/topic");
    QOS qos = QOS_AT_MOST_ONCE;
    int timeout = 10;

    PublishToIoTCoreRequest request;
    Vector<uint8_t> messageData({message.begin(), message.end()});
    request.SetTopicName(topic);
    request.SetPayload(messageData);
    request.SetQos(qos);

    auto operation = ipcClient.NewPublishToIoTCore();
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.
        } else {
            // Handle RPC error.
        }
    }

    return 0;
```

```
}
```

JavaScript

Example Beispiel: Veröffentlichen Sie eine Nachricht

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {QOS, PublishToIoTCoreRequest} from "aws-iot-device-sdk-v2/dist/
greengrasscoreipc/model";

class PublishToIoTCore {
    private ipcClient: greengrasscoreipc.Client
    private readonly topic: string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.publishToIoTCore().then(r => console.log("Started workflow"));
    }

    private async publishToIoTCore() {
        try {
            const request: PublishToIoTCoreRequest = {
                topicName: this.topic,
                qos: QOS.AT_LEAST_ONCE, // you can change this depending on your use
 case
            }

            this.ipcClient = await getIpcClient();

            await this.ipcClient.publishToIoTCore(request);
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}


export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
```

```
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

// starting point
const publishToIoTCore = new PublishToIoTCore();
```

SubscribeToIoTCore

Abonnieren Sie MQTT-Nachrichten zu einem AWS IoT Core Thema oder einem Themenfilter. Die AWS IoT Greengrass Core-Software entfernt Abonnements, wenn die Komponente das Ende ihres Lebenszyklus erreicht hat.

Bei diesem Vorgang handelt es sich um einen Abonnementvorgang, bei dem Sie einen Stream von Ereignisnachrichten abonnieren. Um diese Operation zu verwenden, definieren Sie einen Stream-Response-Handler mit Funktionen, die Ereignismeldungen, Fehler und das Schließen von Streams behandeln. Weitere Informationen finden Sie unter Abonnieren Sie IPC-Event-Streams.

Typ der Ereignisnachricht: `IoTCoreMessage`

Anforderung

Die Anforderung dieses Vorgangs hat die folgenden Parameter:

`topicName`(Python:`topic_name`)

Das Thema, das abonniert werden soll. Sie können Platzhalter (#und+) für MQTT-Themen verwenden, um mehrere Themen zu abonnieren.

`qos`

Die zu verwendende MQTT-QoS. Diese Aufzählung,`QOS`, hat die folgenden Werte:

- `AT_MOST_ONCE`— QoS 0. Die MQTT-Nachricht wird höchstens einmal zugestellt.
- `AT_LEAST_ONCE`— QoS 1. Die MQTT-Nachricht wird mindestens einmal zugestellt.

Antwort

Die Antwort dieser Operation enthält die folgenden Informationen:

`messages`

Der Stream von MQTT-Nachrichten. Dieses Objekt,`IoTCoreMessage`, enthält die folgenden Informationen:

`message`

Die MQTT-Nachricht. Dieses Objekt,`MQTTMessage`, enthält die folgenden Informationen:

`topicName`(Python:`topic_name`)

Das Thema, zu dem die Nachricht veröffentlicht wurde.

`payload`

(Optional) Die Nachrichten-Payload als Blob.

Die folgenden Funktionen sind für Version 2.10.0 und höher verfügbar, Grüngraskern wenn Sie MQTT 5 verwenden. Diese Funktionen werden ignoriert, wenn Sie MQTT 3.1.1 verwenden. In der folgenden Tabelle ist die Mindestversion des AWS IoT Geräte-SDK aufgeführt, die Sie für den Zugriff auf diese Funktionen verwenden müssen.

SDK	Mindestversion
AWS IoT-Geräte-SDK for Python v2	v1.15.0
AWS IoT Device SDK for Java v2	v1.13.0
AWS IoT Device SDK for C++ v2	v1.24,0
AWS IoT Device SDK for JavaScript v2	v1.13.0

`payloadFormat`

(Optional) Das Format der Nachrichtennutzlast. Wenn Sie den nicht festlegen`payloadFormat`, wird davon ausgegangen, dass `BYTES` der Typ Die Aufzählung hat die folgenden Werte:

- `BYTES`— Der Inhalt der Payload ist ein binärer Blob.

- UTF8— Der Inhalt der Payload ist eine UTF8-Zeichenfolge.

`retain`

(Optional) Gibt an, ob die MQTT-Aufbewahrungsoption beim Veröffentlichen auf gesetzt werden soll. `true`

`userProperties`

(Optional) Eine Liste von anwendungsspezifischen `UserProperty` Objekten, die gesendet werden sollen. Das `UserProperty` Objekt ist wie folgt definiert:

```
UserProperty:
  key: string
  value: string
```

`messageExpiryIntervalSeconds`

(Optional) Die Anzahl der Sekunden, bevor die Nachricht abläuft und vom Server gelöscht wird. Wenn dieser Wert nicht festgelegt ist, läuft die Nachricht nicht ab.

`correlationData`

(Optional) Der Anfrage hinzugefügte Informationen, die verwendet werden können, um eine Anfrage mit einer Antwort zu verknüpfen.

`responseTopic`

(Optional) Das Thema, das für die Antwortnachricht verwendet werden soll.

`contentType`

(Optional) Eine anwendungsspezifische Kennung für den Inhaltstyp der Nachricht.

Beispiele

Die folgenden Beispiele zeigen, wie dieser Vorgang in benutzerdefiniertem Komponentencode aufgerufen wird.

Java (IPC client V2)

Example Beispiel: Nachrichten abonnieren

```
package com.aws.greengrass.docs.samples.ipc;
```

```
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.SubscribeToIoTCoreResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.QOS;
import software.amazon.awssdk.aws.greengrass.model.IoTCoreMessage;
import software.amazon.awssdk.aws.greengrass.model.SubscribeToIoTCoreRequest;
import software.amazon.awssdk.aws.greengrass.model.SubscribeToIoTCoreResponse;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.function.Consumer;
import java.util.function.Function;


public class SubscribeToIoTCore {

    public static void main(String[] args) {
        String topic = args[0];
        QOS qos = QOS.get(args[1]);

        Consumer<IoTCoreMessage> onStreamEvent = ioTCoreMessage ->
                System.out.printf("Received new message on topic %s: %s%n",
                        ioTCoreMessage.getMessage().getTopicName(),
                        new String(ioTCoreMessage.getMessage().getPayload(),
 StandardCharsets.UTF_8));

        Optional<Function<Throwable, Boolean>> onStreamError =
                Optional.of(e -> {
                    System.err.println("Received a stream error.");
                    e.printStackTrace();
                    return false;
                });

        Optional<Runnable> onStreamClosed = Optional.of(() ->
                System.out.println("Subscribe to IoT Core stream closed."));

        try (GreengrassCoreIPCClientV2 ipcClientV2 =
 GreengrassCoreIPCClientV2.builder().build()) {
            SubscribeToIoTCoreRequest request = new SubscribeToIoTCoreRequest()
                    .withTopicName(topic)
                    .withQos(qos);

            GreengrassCoreIPCClientV2.StreamingResponse<SubscribeToIoTCoreResponse,
 SubscribeToIoTCoreResponseHandler>
```

```
                    streamingResponse = ipcClientV2.subscribeToIoTCore(request,
 onStreamEvent, onStreamError, onStreamClosed);

            streamingResponse.getResponse();
            System.out.println("Successfully subscribed to topic: " + topic);

            // Keep the main thread alive, or the process will exit.
            while (true) {
                Thread.sleep(10000);
            }

            // To stop subscribing, close the stream.
            streamingResponse.getHandler().closeStream();
        } catch (InterruptedException e) {
            System.out.println("Subscribe interrupted.");
        } catch (Exception e) {
            System.err.println("Exception occurred.");
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

Python (IPC client V2)

Example Beispiel: Nachrichten abonnieren

> **Note**
>
> In diesem Beispiel wird davon ausgegangen, dass Sie Version 1.5.4 oder höher von AWS IoT Device SDK für Python v2 verwenden.

```
import threading
import traceback

import awsiot.greengrasscoreipc.clientv2 as clientV2

topic = 'my/topic'
qos = '1'
```

```
def on_stream_event(event):
    try:
        topic_name = event.message.topic_name
        message = str(event.message.payload, 'utf-8')
        print(f'Received new message on topic {topic_name}:  {message}')
    except:
        traceback.print_exc()

def on_stream_error(error):
    # Return True to close stream, False to keep stream open.
    return True

def on_stream_closed():
    pass

ipc_client = clientV2.GreengrassCoreIPCClientV2()
resp, operation = ipc_client.subscribe_to_iot_core(
    topic_name=topic,
    qos=qos,
    on_stream_event=on_stream_event,
    on_stream_error=on_stream_error,
    on_stream_closed=on_stream_closed
)

# Keep the main thread alive, or the process will exit.
event = threading.Event()
event.wait()

# To stop subscribing, close the operation stream.
operation.close()
ipc_client.close()
```

Java (IPC client V1)

Example Beispiel: Nachrichten abonnieren

> **Note**
>
> In diesem Beispiel wird eine `IPCUtils` Klasse verwendet, um eine Verbindung zum AWS IoT Greengrass Core-IPC-Dienst herzustellen. Weitere Informationen finden Sie unter Connect zum AWS IoT Greengrass Core IPC-Dienst her.

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.SubscribeToIoTCoreResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.*;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;
import software.amazon.awssdk.eventstreamrpc.StreamResponseHandler;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class SubscribeToIoTCore {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        String topic = args[0];
        QOS qos = QOS.get(args[1]);
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            StreamResponseHandler<IoTCoreMessage> streamResponseHandler =
                    new SubscriptionResponseHandler();
            SubscribeToIoTCoreResponseHandler responseHandler =
                    SubscribeToIoTCore.subscribeToIoTCore(ipcClient, topic, qos,
                            streamResponseHandler);
            CompletableFuture<SubscribeToIoTCoreResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                System.out.println("Successfully subscribed to topic: " + topic);
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while subscribing to topic: " +
 topic);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
```

```
                    System.err.println("Unauthorized error while subscribing to
topic: " + topic);
                } else {
                    throw e;
                }
            }

            // Keep the main thread alive, or the process will exit.
            try {
                while (true) {
                    Thread.sleep(10000);
                }
            } catch (InterruptedException e) {
                System.out.println("Subscribe interrupted.");
            }

            // To stop subscribing, close the stream.
            responseHandler.closeStream();
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static SubscribeToIoTCoreResponseHandler
subscribeToIoTCore(GreengrassCoreIPCClient greengrassCoreIPCClient, String topic,
QOS qos, StreamResponseHandler<IoTCoreMessage> streamResponseHandler) {
        SubscribeToIoTCoreRequest subscribeToIoTCoreRequest = new
SubscribeToIoTCoreRequest();
        subscribeToIoTCoreRequest.setTopicName(topic);
        subscribeToIoTCoreRequest.setQos(qos);
        return
greengrassCoreIPCClient.subscribeToIoTCore(subscribeToIoTCoreRequest,
                Optional.of(streamResponseHandler));
    }

    public static class SubscriptionResponseHandler implements
StreamResponseHandler<IoTCoreMessage> {

        @Override
        public void onStreamEvent(IoTCoreMessage ioTCoreMessage) {
```

```
            try {
                String topic = ioTCoreMessage.getMessage().getTopicName();
                String message = new
 String(ioTCoreMessage.getMessage().getPayload(),
                        StandardCharsets.UTF_8);
                System.out.printf("Received new message on topic %s: %s%n", topic,
 message);
            } catch (Exception e) {
                System.err.println("Exception occurred while processing subscription
 response " +
                        "message.");
                e.printStackTrace();
            }
        }

        @Override
        public boolean onStreamError(Throwable error) {
            System.err.println("Received a stream error.");
            error.printStackTrace();
            return false;
        }

        @Override
        public void onStreamClosed() {
            System.out.println("Subscribe to IoT Core stream closed.");
        }
    }
}
```

Python (IPC client V1)

Example Beispiel: Nachrichten abonnieren

> **Note**
>
> In diesem Beispiel wird davon ausgegangen, dass Sie Version 1.5.4 oder höher von AWS IoT Device SDK für Python v2 verwenden.

```
import time
import traceback

import awsiot.greengrasscoreipc
```

```
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import (
    IoTCoreMessage,
    QOS,
    SubscribeToIoTCoreRequest
)

TIMEOUT = 10

ipc_client = awsiot.greengrasscoreipc.connect()

class StreamHandler(client.SubscribeToIoTCoreStreamHandler):
    def __init__(self):
        super().__init__()

    def on_stream_event(self, event: IoTCoreMessage) -> None:
        try:
            message = str(event.message.payload, "utf-8")
            topic_name = event.message.topic_name
            # Handle message.
        except:
            traceback.print_exc()

    def on_stream_error(self, error: Exception) -> bool:
        # Handle error.
        return True  # Return True to close stream, False to keep stream open.

    def on_stream_closed(self) -> None:
        # Handle close.
        pass


topic = "my/topic"
qos = QOS.AT_MOST_ONCE

request = SubscribeToIoTCoreRequest()
request.topic_name = topic
request.qos = qos
handler = StreamHandler()
operation = ipc_client.new_subscribe_to_iot_core(handler)
operation.activate(request)
future_response = operation.get_response()
future_response.result(TIMEOUT)
```

```
# Keep the main thread alive, or the process will exit.
while True:
    time.sleep(10)

# To stop subscribing, close the operation stream.
operation.close()
```

C++

Example Beispiel: Nachrichten abonnieren

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IoTCoreResponseHandler : public SubscribeToIoTCoreStreamHandler {

    public:
        virtual ~IoTCoreResponseHandler() {}

    private:
        void OnStreamEvent(IoTCoreMessage *response) override {
            auto message = response->GetMessage();
            if (message.has_value() && message.value().GetPayload().has_value()) {
                auto messageBytes = message.value().GetPayload().value();
                std::string messageString(messageBytes.begin(), messageBytes.end());
                std::string topicName =
 message.value().GetTopicName().value().c_str();
                // Handle message.
            }
        }

        bool OnStreamError(OperationError *error) override {
            // Handle error.
            return false; // Return true to close stream, false to keep stream open.
        }

        void OnStreamClosed() override {
            // Handle close.
        }
```

```
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};

int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String topic("my/topic");
    QOS qos = QOS_AT_MOST_ONCE;
    int timeout = 10;

    SubscribeToIoTCoreRequest request;
    request.SetTopicName(topic);
    request.SetQos(qos);
    auto streamHandler = MakeShared<IoTCoreResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToIoTCore(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
```

```
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
 Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.
        } else {
            // Handle RPC error.
        }
        exit(-1);
    }

    // Keep the main thread alive, or the process will exit.
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
    }

    operation->Close();
    return 0;
}
```

JavaScript

Example Beispiel: Nachrichten abonnieren

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {IoTCoreMessage, QOS, SubscribeToIoTCoreRequest} from "aws-iot-device-sdk-v2/
dist/greengrasscoreipc/model";
import {RpcError} from "aws-iot-device-sdk-v2/dist/eventstream_rpc";

class SubscribeToIoTCore {
    private ipcClient: greengrasscoreipc.Client
    private readonly topic: string;
```

```
    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.subscribeToIoTCore().then(r => console.log("Started workflow"));
    }

    private async subscribeToIoTCore() {
        try {
            const request: SubscribeToIoTCoreRequest = {
                topicName: this.topic,
                qos: QOS.AT_LEAST_ONCE, // you can change this depending on your use
 case
            }

            this.ipcClient = await getIpcClient();

            const streamingOperation = this.ipcClient.subscribeToIoTCore(request);

            streamingOperation.on('message', (message: IoTCoreMessage) => {
                // parse the message depending on your use cases, e.g.
                if (message.message && message.message.payload) {
                    const receivedMessage = message.message.payload.toString();
                }
            });

            streamingOperation.on('streamError', (error : RpcError) => {
                // define your own error handling logic
            });

            streamingOperation.on('ended', () => {
                // define your own logic
            });

            await streamingOperation.activate();

            // Keep the main thread alive, or the process will exit.
            await new Promise((resolve) => setTimeout(resolve, 10000))
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}
```

```
export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

// starting point
const subscribeToIoTCore = new SubscribeToIoTCore();
```

Beispiele

Verwenden Sie die folgenden Beispiele, um zu erfahren, wie Sie den AWS IoT Core MQTT IPC-Service in Ihren Komponenten verwenden können.

Beispiel AWS IoT Core MQTT-Publisher (C++)

Das folgende Beispielrezept ermöglicht es der Komponente, in allen Themen zu veröffentlichen.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IoTCorePublisherCpp",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes MQTT messages to IoT Core.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.IoTCorePublisherCpp:mqttproxy:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": [
```

```
              "aws.greengrass#PublishToIoTCore"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "{artifacts:path}/greengrassv2_iotcore_publisher"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.IoTCorePublisherCpp/1.0.0/greengrassv2_iotcore_publisher",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.IoTCorePublisherCpp
ComponentVersion: 1.0.0
ComponentDescription: A component that publishes MQTT messages to IoT Core.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.mqttproxy:
        com.example.IoTCorePublisherCpp:mqttproxy:1:
          policyDescription: Allows access to publish to all topics.
          operations:
```

```
            - aws.greengrass#PublishToIoTCore
          resources:
            - "*"
Manifests:
  - Lifecycle:
      run: "{artifacts:path}/greengrassv2_iotcore_publisher"
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.IoTCorePublisherCpp/1.0.0/greengrassv2_iotcore_publisher
        Permission:
          Execute: OWNER
```

Die folgende C++-Beispielanwendung zeigt, wie der AWS IoT Core MQTT-IPC-Dienst verwendet wird, um Nachrichten zu veröffentlichen. AWS IoT Core

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() << std::endl;
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
};

int main() {
    String message("Hello from the Greengrass IPC MQTT publisher (C++).");
    String topic("test/topic/cpp");
```

```
    QOS qos = QOS_AT_LEAST_ONCE;
    int timeout = 10;

    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    while (true) {
        PublishToIoTCoreRequest request;
        Vector<uint8_t> messageData({message.begin(), message.end()});
        request.SetTopicName(topic);
        request.SetPayload(messageData);
        request.SetQos(qos);

        auto operation = ipcClient.NewPublishToIoTCore();
        auto activate = operation->Activate(request, nullptr);
        activate.wait();

        auto responseFuture = operation->GetResult();
        if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
std::future_status::timeout) {
            std::cerr << "Operation timed out while waiting for response from
Greengrass Core." << std::endl;
            exit(-1);
        }

        auto response = responseFuture.get();
        if (response) {
            std::cout << "Successfully published to topic: " << topic << std::endl;
        } else {
            // An error occurred.
            std::cout << "Failed to publish to topic: " << topic << std::endl;
            auto errorType = response.GetResultType();
            if (errorType == OPERATION_ERROR) {
                auto *error = response.GetOperationError();
```

```
                std::cout << "Operation error: " << error->GetMessage().value() <<
 std::endl;
            } else {
                std::cout << "RPC error: " << response.GetRpcError() << std::endl;
            }
            exit(-1);
        }

        std::this_thread::sleep_for(std::chrono::seconds(5));
    }

    return 0;
}
```

Beispiel für einen AWS IoT Core MQTT-Abonnenten (C++)

Das folgende Beispielrezept ermöglicht es der Komponente, alle Themen zu abonnieren.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IoTCoreSubscriberCpp",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to MQTT messages from IoT
 Core.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.IoTCoreSubscriberCpp:mqttproxy:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToIoTCore"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
```

```
  "Manifests": [
    {
      "Lifecycle": {
        "run": "{artifacts:path}/greengrassv2_iotcore_subscriber"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.IoTCoreSubscriberCpp/1.0.0/greengrassv2_iotcore_subscriber",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.IoTCoreSubscriberCpp
ComponentVersion: 1.0.0
ComponentDescription: A component that subscribes to MQTT messages from IoT Core.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.mqttproxy:
        com.example.IoTCoreSubscriberCpp:mqttproxy:1:
          policyDescription: Allows access to subscribe to all topics.
          operations:
            - aws.greengrass#SubscribeToIoTCore
          resources:
            - "*"
Manifests:
  - Lifecycle:
      run: "{artifacts:path}/greengrassv2_iotcore_subscriber"
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.IoTCoreSubscriberCpp/1.0.0/greengrassv2_iotcore_subscriber
        Permission:
```

```
            Execute: OWNER
```

Die folgende C++-Beispielanwendung zeigt, wie der AWS IoT Core MQTT-IPC-Dienst verwendet wird, um Nachrichten von zu abonnieren. AWS IoT Core

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IoTCoreResponseHandler : public SubscribeToIoTCoreStreamHandler {

    public:
        virtual ~IoTCoreResponseHandler() {}

    private:

        void OnStreamEvent(IoTCoreMessage *response) override {
            auto message = response->GetMessage();
            if (message.has_value() && message.value().GetPayload().has_value()) {
                auto messageBytes = message.value().GetPayload().value();
                std::string messageString(messageBytes.begin(), messageBytes.end());
                std::string messageTopic =
 message.value().GetTopicName().value().c_str();
                std::cout << "Received new message on topic: " << messageTopic <<
 std::endl;
                std::cout << "Message: " << messageString << std::endl;
            }
        }

        bool OnStreamError(OperationError *error) override {
            std::cout << "Received an operation error: ";
            if (error->GetMessage().has_value()) {
                std::cout << error->GetMessage().value();
            }
            std::cout << std::endl;
            return false; // Return true to close stream, false to keep stream open.
        }
```

```
        void OnStreamClosed() override {
            std::cout << "Subscribe to IoT Core stream closed." << std::endl;
        }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() << std::endl;
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
};

int main() {
    String topic("test/topic/cpp");
    QOS qos = QOS_AT_LEAST_ONCE;
    int timeout = 10;

    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    SubscribeToIoTCoreRequest request;
    request.SetTopicName(topic);
    request.SetQos(qos);
    auto streamHandler = MakeShared<IoTCoreResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToIoTCore(streamHandler);
```

```
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
 Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (response) {
        std::cout << "Successfully subscribed to topic: " << topic << std::endl;
    } else {
        // An error occurred.
        std::cout << "Failed to subscribe to topic: " << topic << std::endl;
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            std::cout << "Operation error: " << error->GetMessage().value() <<
 std::endl;
        } else {
            std::cout << "RPC error: " << response.GetRpcError() << std::endl;
        }
        exit(-1);
    }

    // Keep the main thread alive, or the process will exit.
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
    }

    operation->Close();
    return 0;
}
```

Interagieren mit dem Komponentenlebenszyklus

Verwenden Sie den Komponentenlebenszyklus-IPK-Service, um:

- Aktualisieren Sie den Komponentenstatus auf dem Core-Gerät.

- Abonnieren Sie Aktualisierungen des Komponentenstatus.
- Verhindern Sie, dass der Kern die Komponente stoppt, um während einer Bereitstellung ein Update anzuwenden.
- Pausieren und Fortsetzen von Komponentenprozessen.

Themen

SDK-Mindestversionen

In der folgenden Tabelle sind die Mindestversionen von aufgeführtAWS IoT Device SDK, die Sie für die Interaktion mit dem Komponentenlebenszyklus verwenden müssen.

SDK	Mindestversion	
AWS IoT Device SDK für Java v2	v1.2.10	
AWS IoT Device SDK für Python v2	v1.5.3	
AWS IoT Device SDK für C++ v2	v1.17.0	
AWS IoT Device SDK für JavaScript v2	v1.12.0	

Autorisierung

Um andere Komponenten von einer benutzerdefinierten Komponente aus anzuhalten oder fortzusetzen, müssen Sie Autorisierungsrichtlinien definieren, die es Ihrer Komponente ermöglichen, andere Komponenten zu verwalten. Informationen zum Definieren von Autorisierungsrichtlinien finden Sie unter Autorisieren Sie Komponenten zur Ausführung von IPC-Vorgängen.

Autorisierungsrichtlinien für die Verwaltung des Komponentenlebenszyklus haben die folgenden Eigenschaften.

IPC-Service-ID: `aws.greengrass.ipc.lifecycle`

Operation	Beschreibung	Ressourcen
`aws.greengrass#PauseComponent`	Ermöglicht einer Komponente, die von Ihnen angegebenen Komponenten anzuhalten.	Ein Komponentenname oder `*`, um den Zugriff auf alle Komponenten zu ermöglichen.
`aws.greengrass#ResumeComponent`	Ermöglicht einer Komponente, die von Ihnen angegebenen Komponenten fortzusetzen.	Ein Komponentenname oder `*`, um den Zugriff auf alle Komponenten zu ermöglichen.
`*`	Ermöglicht einer Komponente, die von Ihnen angegebenen Komponenten anzuhalten und fortzusetzen.	Ein Komponentenname oder `*`, um den Zugriff auf alle Komponenten zu ermöglichen.

Beispiele für Autorisierungsrichtlinien

Sie können auf das folgende Beispiel für eine Autorisierungsrichtlinie verweisen, um Ihnen bei der Konfiguration von Autorisierungsrichtlinien für Ihre Komponenten zu helfen.

Example Beispiel für eine Autorisierungsrichtlinie

Die folgende Beispiel-Autorisierungsrichtlinie ermöglicht es einer Komponente, alle Komponenten anzuhalten und fortzusetzen.

```
{
```

```
  "accessControl": {
    "aws.greengrass.ipc.lifecycle": {
      "com.example.MyLocalLifecycleComponent:lifecycle:1": {
        "policyDescription": "Allows access to pause/resume all components.",
        "operations": [
          "aws.greengrass#PauseComponent",
          "aws.greengrass#ResumeComponent"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

UpdateState

Aktualisieren Sie den Status der Komponente auf dem Core-Gerät.

Anforderung

Die Anforderung dieser Operation hat die folgenden Parameter:

`state`

Der Status, der festgelegt werden soll. Diese Aufzählung, `LifecycleState`, hat die folgenden Werte:

- `RUNNING`
- `ERRORED`

Antwort

Dieser Vorgang liefert keine Informationen in seiner Antwort.

SubscribeToComponentUpdates

Abonnieren Sie , um Benachrichtigungen zu erhalten, bevor die AWS IoT Greengrass -Core-Software eine Komponente aktualisiert. Die Benachrichtigung gibt an, ob der Kern im Rahmen des Updates neu gestartet wird oder nicht.

Der -Kern sendet Aktualisierungsbenachrichtigungen nur, wenn die Richtlinie zur Komponentenaktualisierung der Bereitstellung angibt, Komponenten zu benachrichtigen. Das Standardverhalten besteht darin, Komponenten zu benachrichtigen. Weitere Informationen finden Sie unter Erstellen von Bereitstellungen und das -DeploymentComponentUpdatePolicyObjekt, das Sie beim Aufrufen der -CreateDeploymentOperation angeben können.

> **Important**
>
> Lokale Bereitstellungen benachrichtigen Komponenten nicht vor Aktualisierungen.

Dieser Vorgang ist ein Abonnementvorgang, bei dem Sie einen Stream von Ereignisnachrichten abonnieren. Um diesen Vorgang zu verwenden, definieren Sie einen Stream-Antwort-Handler mit Funktionen, die Ereignismeldungen, Fehler und Stream-Schließung verarbeiten. Weitere Informationen finden Sie unter Abonnieren Sie IPC-Event-Streams.

Typ der Ereignisnachricht: `ComponentUpdatePolicyEvents`

> **Tip**
>
> Sie können einem Tutorial folgen, um zu erfahren, wie Sie eine Komponente entwickeln, die Komponentenaktualisierungen bedingt verzögert. Weitere Informationen finden Sie unter Tutorial: Entwickeln einer Greengrass-Komponente, die Komponentenaktualisierungen verzögert.

Anforderung

Die Anforderung dieser Operation hat keine Parameter.

Antwort

Die Antwort dieser Operation enthält die folgenden Informationen:

`messages`

Der Stream von Benachrichtigungen. Dieses Objekt, `ComponentUpdatePolicyEvents`, enthält die folgenden Informationen:

`preUpdateEvent` (Python: `pre_update_event`)

(Optional) Ein Ereignis, das angibt, dass der Kern eine Komponente aktualisieren möchte. Sie können mit der -DeferComponentUpdateOperation antworten, um das Update zu bestätigen oder zu verschieben, bis Ihre Komponente zum Neustart bereit ist. Dieses Objekt, `PreComponentUpdateEvent`, enthält die folgenden Informationen:

`deploymentId` (Python: `deployment_id`)

Die ID der AWS IoT Greengrass Bereitstellung, die die Komponente aktualisiert.

`isGgcRestarting` (Python: `is_ggc_restarting`)

Gibt an, ob der Kern neu gestartet werden muss, um das Update anzuwenden.

`postUpdateEvent` (Python: `post_update_event`)

(Optional) Ein Ereignis, das angibt, dass der Kern eine Komponente aktualisiert hat. Dieses Objekt, `PostComponentUpdateEvent`, enthält die folgenden Informationen:

`deploymentId` (Python: `deployment_id`)

Die ID der AWS IoT Greengrass Bereitstellung, die die Komponente aktualisiert hat.

> **Note**
>
> Diese Funktion erfordert v2.7.0 oder höher der Greengrass-Kernkomponente.

DeferComponentUpdate

Bestätigen oder verschieben Sie eine Komponentenaktualisierung, die Sie mit entdeckenSubscribeToComponentUpdates. Sie geben die Wartezeit an, bevor der Kern erneut prüft, ob Ihre Komponente bereit ist, die Komponentenaktualisierung fortzusetzen. Sie können diese Operation auch verwenden, um dem Kern mitzuteilen, dass Ihre Komponente für die Aktualisierung bereit ist.

Wenn eine Komponente nicht auf die Benachrichtigung zur Komponentenaktualisierung reagiert, wartet der Kern die Zeit, die Sie in der Richtlinie zur Komponentenaktualisierung der Bereitstellung angeben. Nach diesem Timeout fährt der Kern mit der Bereitstellung fort. Das Standard-Timeout für Komponentenaktualisierungen beträgt 60 Sekunden. Weitere Informationen finden Sie unter Erstellen von Bereitstellungen und das -DeploymentComponentUpdatePolicyObjekt, das Sie beim Aufrufen der -CreateDeploymentOperation angeben können.

> **Tip**
>
> Sie können einem Tutorial folgen, um zu erfahren, wie Sie eine Komponente entwickeln, die Komponentenaktualisierungen bedingt verzögert. Weitere Informationen finden Sie unter Tutorial: Entwickeln einer Greengrass-Komponente, die Komponentenaktualisierungen verzögert.

Anforderung

Die Anforderung dieser Operation hat die folgenden Parameter:

`deploymentId` (Python: `deployment_id`)

Die ID der zu verschiebenden AWS IoT Greengrass Bereitstellung.

`message`

(Optional) Der Name der Komponente, für die Updates verschoben werden sollen.

Standardmäßig ist der Name der Komponente, die die Anforderung stellt.

`recheckAfterMs` (Python: `recheck_after_ms`)

Die Zeit in Millisekunden, für die die Aktualisierung verschoben werden soll. Der Kern wartet diese Zeit und sendet dann ein weiteres `PreComponentUpdateEvent`, das Sie mit erkennen könnenSubscribeToComponentUpdates.

Geben Sie an`0`, um die Aktualisierung zu bestätigen. Dadurch wird dem Kern mitgeteilt, dass Ihre Komponente für das Update bereit ist.

Der Standardwert ist null Millisekunden, was bedeutet, dass die Aktualisierung bestätigt wird.

Antwort

Dieser Vorgang liefert keine Informationen in seiner Antwort.

PauseComponent

Diese Funktion ist für v2.4.0 und höher der Greengrass-Kernkomponente verfügbar. unterstützt diese Funktion derzeit AWS IoT Greengrass nicht auf Windows-Kerngeräten.

Pausiert die Prozesse einer Komponente auf dem Core-Gerät. Um eine Komponente fortzusetzen, verwenden Sie die -ResumeComponentOperation.

Sie können nur generische Komponenten pausieren. Wenn Sie versuchen, einen anderen Komponententyp anzuhalten, löst dieser Vorgang eine aus`InvalidRequestError`.

> **Note**
>
> Dieser Vorgang kann keine containerisierten Prozesse wie Docker-Container anhalten. Um einen Docker-Container anzuhalten und fortzusetzen, können Sie die Befehle Docker anhalten und Docker anhalten verwenden.

Dieser Vorgang unterbricht keine Komponentenabhängigkeiten oder Komponenten, die von der Komponente abhängen, die Sie pausieren. Beachten Sie dieses Verhalten, wenn Sie eine Komponente anhalten, die eine Abhängigkeit einer anderen Komponente ist, da bei der abhängigen Komponente Probleme auftreten können, wenn ihre Abhängigkeit angehalten wird.

Wenn Sie eine angehaltene Komponente neu starten oder herunterfahren, z. B. durch eine Bereitstellung, nimmt der Greengrass-Kernus die Komponente wieder auf und führt ihren Lebenszyklus zum Herunterfahren aus. Weitere Informationen zum Neustarten einer Komponente finden Sie unter RestartComponent.

> **Important**
>
> Um diese Operation verwenden zu können, müssen Sie eine Autorisierungsrichtlinie definieren, die die Berechtigung zur Verwendung dieser Operation erteilt. Weitere Informationen finden Sie unter Autorisierung.

SDK-Mindestversionen

In der folgenden Tabelle sind die Mindestversionen von aufgeführtAWS IoT Device SDK, die Sie zum Anhalten und Fortsetzen von Komponenten verwenden müssen.

SDK	Mindestversion	
AWS IoT Device SDK für Java v2	v1.4.3	

SDK	Mindestversion	
AWS IoT Device SDK für Python v2	v1.6.2	
AWS IoT Device SDK für C++ v2	v1.13.1	
AWS IoT Device SDK für JavaScript v2	v1.12.0	

Anforderung

Die Anforderung dieser Operation hat die folgenden Parameter:

`componentName` (Python: `component_name`)

Der Name der Komponente, die angehalten werden soll. Dabei muss es sich um eine generische Komponente handeln. Weitere Informationen finden Sie unter Komponententypen.

Antwort

Dieser Vorgang liefert keine Informationen in seiner Antwort.

ResumeComponent

Diese Funktion ist für v2.4.0 und höher der Greengrass-Kernkomponente verfügbar. unterstützt diese Funktion derzeit AWS IoT Greengrass nicht auf Windows-Kerngeräten.

Setzt die Prozesse einer Komponente auf dem Core-Gerät fort. Um eine Komponente anzuhalten, verwenden Sie die -PauseComponentOperation.

Sie können nur angehaltene Komponenten fortsetzen. Wenn Sie versuchen, eine Komponente fortzusetzen, die nicht angehalten wurde, löst dieser Vorgang eine aus`InvalidRequestError`.

> **Important**
>
> Um diesen Vorgang verwenden zu können, müssen Sie eine Autorisierungsrichtlinie definieren, die die entsprechende Berechtigung erteilt. Weitere Informationen finden Sie unter Autorisierung.

SDK-Mindestversionen

In der folgenden Tabelle sind die Mindestversionen von aufgeführtAWS IoT Device SDK, die Sie zum Anhalten und Fortsetzen von Komponenten verwenden müssen.

SDK	Mindestversion
AWS IoT Device SDK für Java v2	v1.4.3
AWS IoT Device SDK für Python v2	v1.6.2
AWS IoT Device SDK für C++ v2	v1.13.1
AWS IoT Device SDK für JavaScript v2	v1.12.0

Anforderung

Die Anforderung dieser Operation hat die folgenden Parameter:

`componentName` (Python: `component_name`)

Der Name der Komponente, die fortgesetzt werden soll.

Antwort

Dieser Vorgang liefert keine Informationen in seiner Antwort.

Interagieren mit der Komponentenkonfiguration

Mit dem Komponentenkonfigurations-IPK-Service können Sie Folgendes tun:

- Abrufen und Festlegen von Komponentenkonfigurationsparametern.
- Abonnieren Sie Aktualisierungen der Komponentenkonfiguration.
- Validieren Sie Aktualisierungen der Komponentenkonfiguration, bevor der Kern sie anwendet.

Themen

SDK-Mindestversionen

In der folgenden Tabelle sind die Mindestversionen von aufgeführtAWS IoT Device SDK, die Sie für die Interaktion mit der Komponentenkonfiguration verwenden müssen.

SDK	Mindestversion	
AWS IoT Device SDK für Java v2	v1.2.10	
AWS IoT Device SDK für Python v2	v1.5.3	
AWS IoT Device SDK für C++ v2	v1.17.0	
AWS IoT Device SDK für JavaScript v2	v1.12.0	

GetConfiguration

Ruft einen Konfigurationswert für eine Komponente auf dem Core-Gerät ab. Sie geben den Schlüsselpfad an, für den ein Konfigurationswert abgerufen werden soll.

Anforderung

Die Anforderung dieser Operation hat die folgenden Parameter:

`componentName` (Python: `component_name`)

(Optional) Der Name der Komponente.

Standardmäßig ist der Name der Komponente, die die Anforderung stellt.

`keyPath` (Python: `key_path`)

Der Schlüsselpfad zum Konfigurationswert. Geben Sie eine Liste an, in der jeder Eintrag der Schlüssel für eine einzelne Ebene im Konfigurationsobjekt ist. Geben Sie beispielsweise an, `["mqtt", "port"]` um den Wert von `port` in der folgenden Konfiguration abzurufen.

```
{
  "mqtt": {
    "port": 443
  }
}
```

Um die vollständige Konfiguration der Komponente abzurufen, geben Sie eine leere Liste an.

Antwort

Die Antwort dieser Operation enthält die folgenden Informationen:

`componentName` (Python: `component_name`)

Der Name der Komponente.

`value`

Die angeforderte Konfiguration als Objekt.

UpdateConfiguration

Aktualisiert einen Konfigurationswert für diese Komponente auf dem Core-Gerät.

Anforderung

Die Anforderung dieser Operation hat die folgenden Parameter:

`keyPath` (Python: `key_path`)

(Optional) Der Schlüsselpfad zum zu aktualisierenden Container-Knoten (das Objekt). Geben Sie eine Liste an, in der jeder Eintrag der Schlüssel für eine einzelne Ebene im Konfigurationsobjekt ist. Geben Sie beispielsweise den Schlüsselpfad `["mqtt"]` und den Zusammenführungswert an, `{ "port": 443 }` um den Wert von `port` in der folgenden Konfiguration festzulegen.

```
{
  "mqtt": {
    "port": 443
  }
}
```

Der Schlüsselpfad muss einen Containerknoten (ein Objekt) in der Konfiguration angeben. Wenn der Knoten in der Konfiguration der Komponente nicht vorhanden ist, wird er von dieser Operation erstellt und auf den Wert für das Objekt in festgelegt`valueToMerge`.

Standardmäßig ist das Stammverzeichnis des Konfigurationsobjekts.

`timestamp`

Die aktuelle Unix-Epochenzeit in Millisekunden. Dieser Vorgang verwendet diesen Zeitstempel, um gleichzeitige Aktualisierungen des Schlüssels aufzulösen. Wenn der Schlüssel in der Komponentenkonfiguration einen größeren Zeitstempel als der Zeitstempel in der Anforderung hat, schlägt die Anforderung fehl.

`valueToMerge` (Python: `value_to_merge`)

Das Konfigurationsobjekt, das an dem Speicherort zusammengeführt werden soll, den Sie in angeben`keyPath`. Weitere Informationen finden Sie unter Komponentenkonfigurationen aktualisieren.

Antwort

Dieser Vorgang liefert keine Informationen in seiner Antwort.

SubscribeToConfigurationUpdate

Abonnieren Sie , um Benachrichtigungen zu erhalten, wenn die Konfiguration einer Komponente aktualisiert wird. Wenn Sie einen Schlüssel abonnieren, erhalten Sie eine Benachrichtigung, wenn ein untergeordnetes Element dieses Schlüssels aktualisiert wird.

Bei diesem Vorgang handelt es sich um einen Abonnementvorgang, bei dem Sie einen Stream von Ereignisnachrichten abonnieren. Um diesen Vorgang zu verwenden, definieren Sie einen Stream-Antwort-Handler mit Funktionen, die Ereignismeldungen, Fehler und Stream-Schließung verarbeiten. Weitere Informationen finden Sie unter Abonnieren Sie IPC-Event-Streams.

Typ der Ereignisnachricht: `ConfigurationUpdateEvents`

Anforderung

Die Anforderung dieser Operation hat die folgenden Parameter:

`componentName` (Python: `component_name`)

(Optional) Der Name der Komponente.

Standardmäßig ist der Name der Komponente, die die Anforderung stellt.

`keyPath` (Python: `key_path`)

Der Schlüsselpfad zum Konfigurationswert, den Sie abonnieren möchten. Geben Sie eine Liste an, in der jeder Eintrag der Schlüssel für eine einzelne Ebene im Konfigurationsobjekt ist. Geben Sie beispielsweise an, `["mqtt", "port"]` um den Wert von `port` in der folgenden Konfiguration abzurufen.

```
{
  "mqtt": {
    "port": 443
  }
}
```

Um Updates für alle Werte in der Konfiguration der Komponente zu abonnieren, geben Sie eine leere Liste an.

Antwort

Die Antwort dieser Operation enthält die folgenden Informationen:

`messages`

Der Stream von Benachrichtigungen. Dieses Objekt, `ConfigurationUpdateEvents`, enthält die folgenden Informationen:

`configurationUpdateEvent` (Python: `configuration_update_event`)

Das Konfigurationsaktualisierungsereignis. Dieses Objekt, `ConfigurationUpdateEvent`, enthält die folgenden Informationen:

`componentName` (Python: `component_name`)

Der Name der Komponente.

`keyPath` (Python: `key_path`)

Der Schlüsselpfad zum Konfigurationswert, den aktualisiert hat.

SubscribeToValidateConfigurationUpdates

Abonnieren Sie , um Benachrichtigungen zu erhalten, bevor die Konfigurationsaktualisierungen dieser Komponente aktualisiert werden. Auf diese Weise können Komponenten Aktualisierungen an ihrer eigenen Konfiguration validieren. Verwenden Sie die -SendConfigurationValidityReportOperation, um dem Kern mitzuteilen, ob die Konfiguration gültig ist oder nicht.

> **Important**
>
> Lokale Bereitstellungen benachrichtigen Komponenten nicht über Updates.

Bei diesem Vorgang handelt es sich um einen Abonnementvorgang, bei dem Sie einen Stream von Ereignisnachrichten abonnieren. Um diesen Vorgang zu verwenden, definieren Sie einen Stream-Antwort-Handler mit Funktionen, die Ereignismeldungen, Fehler und Stream-Schließung verarbeiten. Weitere Informationen finden Sie unter Abonnieren Sie IPC-Event-Streams.

Typ der Ereignisnachricht: `ValidateConfigurationUpdateEvents`

Anforderung

Die Anforderung dieser Operation hat keine Parameter.

Antwort

Die Antwort dieser Operation enthält die folgenden Informationen:

`messages`

Der Stream von Benachrichtigungen. Dieses Objekt, `ValidateConfigurationUpdateEvents`, enthält die folgenden Informationen:

`validateConfigurationUpdateEvent` (Python: `validate_configuration_update_event`)

Das Konfigurationsaktualisierungsereignis. Dieses Objekt, `ValidateConfigurationUpdateEvent`, enthält die folgenden Informationen:

`deploymentId` (Python: `deployment_id`)

Die ID der AWS IoT Greengrass Bereitstellung, die die Komponente aktualisiert.

`configuration`

Das Objekt, das die neue Konfiguration enthält.

SendConfigurationValidityReport

Teilen Sie dem Kern mit, ob eine Konfigurationsaktualisierung für diese Komponente gültig ist oder nicht. Die Bereitstellung schlägt fehl, wenn Sie dem Kern mitteilen, dass die neue Konfiguration ungültig ist. Verwenden Sie die -SubscribeToValidateConfigurationUpdatesOperation, um Konfigurationsaktualisierungen zu validieren.

Wenn eine Komponente nicht auf eine Benachrichtigung zur Konfigurationsaktualisierung antwortet, wartet der Kern die Zeit, die Sie in der Konfigurationsvalidierungsrichtlinie der Bereitstellung angeben. Nach diesem Timeout fährt der Kern mit der Bereitstellung fort. Das Standardzeitlimit für die Komponentenvalidierung beträgt 20 Sekunden. Weitere Informationen finden Sie unter Erstellen von Bereitstellungen und das -DeploymentConfigurationValidationPolicyObjekt, das Sie beim Aufrufen der -CreateDeploymentOperation angeben können.

Anforderung

Die Anforderung dieser Operation hat die folgenden Parameter:

`configurationValidityReport` (Python: `configuration_validity_report`)

Der Bericht, der dem Kern mitteilt, ob die Konfigurationsaktualisierung gültig ist oder nicht. Dieses Objekt, `ConfigurationValidityReport`, enthält die folgenden Informationen:

`status`

Der Gültigkeitsstatus. Diese Aufzählung, `ConfigurationValidityStatus`, hat die folgenden Werte:

- `ACCEPTED` – Die Konfiguration ist gültig und der Kern kann sie auf diese Komponente anwenden.
- `REJECTED` – Die Konfiguration ist ungültig und die Bereitstellung schlägt fehl.

`deploymentId` (Python: `deployment_id`)

Die ID der AWS IoT Greengrass Bereitstellung, die das Konfigurationsupdate angefordert hat.

`message`

(Optional) Eine Meldung, die meldet, warum die Konfiguration ungültig ist.

Antwort

Dieser Vorgang liefert keine Informationen in seiner Antwort.

Abrufen von Secret-Werten

Verwenden Sie den Secret Manager IPC-Service, um Secret-Werte aus Secrets auf dem Core-Gerät abzurufen. Sie verwenden die Secret-Manager-Komponente, um verschlüsselte Secrets auf -Core-Geräten bereitzustellen. Anschließend können Sie eine IPC-Operation verwenden, um das Secret zu entschlüsseln und seinen Wert in Ihren benutzerdefinierten Komponenten zu verwenden.

Themen

SDK-Mindestversionen

In der folgenden Tabelle sind die Mindestversionen von aufgeführtAWS IoT Device SDK, die Sie verwenden müssen, um Secret-Werte von Secrets auf dem Core-Gerät abzurufen.

SDK	Mindestversion	
AWS IoT Device SDK für Java v2	v1.2.10	
AWS IoT Device SDK für Python v2	v1.5.3	
AWS IoT Device SDK für C++ v2	v1.17.0	
AWS IoT Device SDK für JavaScript v2	v1.12.0	

Autorisierung

Um Secret Manager in einer benutzerdefinierten Komponente zu verwenden, müssen Sie Autorisierungsrichtlinien definieren, die es Ihrer Komponente ermöglichen, den Wert von Secrets abzurufen, die Sie auf dem Core-Gerät speichern. Informationen zum Definieren von Autorisierungsrichtlinien finden Sie unter Autorisieren Sie Komponenten zur Ausführung von IPC-Vorgängen.

Autorisierungsrichtlinien für Secret Manager haben die folgenden Eigenschaften.

IPC-Service-ID: `aws.greengrass.SecretManager`

Operation	Beschreibung	Ressourcen
`aws.greengrass#GetSecretValue`	Ermöglicht einer Komponente, den Wert von Secrets abzurufen, die auf dem Core-Gerät verschlüsselt sind.	Ein Secrets-Manager-Geheimnis-ARN oder , * um den Zugriff auf alle Geheimnisse zu ermöglichen.

Beispiele für Autorisierungsrichtlinien

Sie können auf das folgende Beispiel für eine Autorisierungsrichtlinie verweisen, um Ihnen bei der Konfiguration von Autorisierungsrichtlinien für Ihre Komponenten zu helfen.

Example Beispiel für eine Autorisierungsrichtlinie

Die folgende Beispielautorisierungsrichtlinie erlaubt es einer Komponente, den Wert eines beliebigen Secrets auf dem Core-Gerät abzurufen.

> **Note**
>
> Wir empfehlen, dass Sie in einer Produktionsumgebung den Umfang der Autorisierungsrichtlinie reduzieren, sodass die Komponente nur die Secrets abruft, die sie verwendet. Sie können den * Platzhalter in eine Liste geheimer ARNs ändern, wenn Sie die Komponente bereitstellen.

```
{
  "accessControl": {
    "aws.greengrass.SecretManager": {
      "com.example.MySecretComponent:secrets:1": {
        "policyDescription": "Allows access to a secret.",
        "operations": [
          "aws.greengrass#GetSecretValue"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

GetSecretValue

Ruft den Wert eines Secrets ab, das Sie auf dem Core-Gerät speichern.

Dieser Vorgang ähnelt dem Secrets-Manager-Vorgang, mit dem Sie den Wert eines Secrets im abrufen könnenAWS Cloud. Weitere Informationen finden Sie unter GetSecretValue in der AWS Secrets Manager-API-Referenz.

Anforderung

Die Anforderung dieser Operation hat die folgenden Parameter:

`secretId` (Python: `secret_id`)

Der Name des abzurufenden Secrets. Sie können entweder den Amazon-Ressourcennamen (ARN) oder den Anzeigenamen des Secrets angeben.

`versionId` (Python: `version_id`)

(Optional) Die ID der abzurufenden Version.

Sie können entweder `versionId` oder `versionStage` angeben.

Wenn Sie oder nicht angeben`versionId``versionStage`, wird dieser Vorgang standardmäßig auf die Version mit der `AWSCURRENT` Bezeichnung gesetzt.

`versionStage` (Python: `version_stage`)

(Optional) Die Staging-Bezeichnung der abzurufenden Version.

Sie können entweder `versionId` oder `versionStage` angeben.

Wenn Sie oder nicht angeben`versionId``versionStage`, wird dieser Vorgang standardmäßig auf die Version mit der `AWSCURRENT` Bezeichnung gesetzt.

Antwort

Die Antwort dieser Operation enthält die folgenden Informationen:

`secretId` (Python: `secret_id`)

Die ID des Secrets.

`versionId` (Python: `version_id`)

Die ID dieser Version des Secrets.

`versionStage` (Python: `version_stage`)

Die Liste der Staging-Bezeichnungen, die an diese Version des Secrets angehängt sind.

`secretValue` (Python: `secret_value`)

Der Wert dieser Version des Secrets. Dieses Objekt, `SecretValue`, enthält die folgenden Informationen.

`secretString` (Python: `secret_string`)

Der entschlüsselte Teil der geschützten geheimen Informationen, den Sie Secrets Manager als Zeichenfolge zur Verfügung gestellt haben.

`secretBinary` (Python: `secret_binary`)

(Optional) Der entschlüsselte Teil der geschützten Secret-Informationen, den Sie Secrets Manager als Binärdaten in Form eines Byte-Arrays bereitgestellt haben. Diese Eigenschaft enthält die Binärdaten als base64-kodierte Zeichenfolge.

Diese Eigenschaft wird nicht verwendet, wenn Sie das Secret in der Secrets-Manager-Konsole erstellt haben.

Beispiele

Die folgenden Beispiele veranschaulichen, wie Sie diese Operation im benutzerdefinierten Komponentencode aufrufen.

Java (IPC client V1)

Example Beispiel: Abrufen eines Secret-Werts

> **Note**
>
> In diesem Beispiel wird eine -`IPCUtils`Klasse verwendet, um eine Verbindung zum AWS IoT Greengrass Core IPC-Service herzustellen. Weitere Informationen finden Sie unter Connect zum AWS IoT Greengrass Core IPC-Dienst her.

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GetSecretValueResponseHandler;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.model.GetSecretValueRequest;
import software.amazon.awssdk.aws.greengrass.model.GetSecretValueResponse;
```

```
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class GetSecretValue {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        String secretArn = args[0];
        String versionStage = args[1];
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            GetSecretValueResponseHandler responseHandler =
                    GetSecretValue.getSecretValue(ipcClient, secretArn,
 versionStage);
            CompletableFuture<GetSecretValueResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                GetSecretValueResponse response =
 futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                response.getSecretValue().postFromJson();
                String secretString = response.getSecretValue().getSecretString();
                System.out.println("Successfully retrieved secret value: " +
 secretString);
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while retrieving secret: " +
 secretArn);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.println("Unauthorized error while retrieving secret:
 " + secretArn);
                } else {
                    throw e;
                }
            }
        } catch (InterruptedException e) {
```

```
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static GetSecretValueResponseHandler
 getSecretValue(GreengrassCoreIPCClient greengrassCoreIPCClient, String secretArn,
 String versionStage) {
        GetSecretValueRequest getSecretValueRequest = new GetSecretValueRequest();
        getSecretValueRequest.setSecretId(secretArn);
        getSecretValueRequest.setVersionStage(versionStage);
        return greengrassCoreIPCClient.getSecretValue(getSecretValueRequest,
 Optional.empty());
    }
}
```

Python (IPC client V1)

Example Beispiel: Abrufen eines Secret-Werts

> **Note**
>
> In diesem Beispiel wird davon ausgegangen, dass Sie Version 1.5.4 oder höher von AWS IoT Device SDK für Python v2 verwenden.

```
import json

import awsiot.greengrasscoreipc
from awsiot.greengrasscoreipc.model import (
    GetSecretValueRequest,
    GetSecretValueResponse,
    UnauthorizedError
)

secret_id = 'arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyGreengrassSecret-abcdef'
TIMEOUT = 10

ipc_client = awsiot.greengrasscoreipc.connect()
```

```
request = GetSecretValueRequest()
request.secret_id = secret_id
request.version_stage = 'AWSCURRENT'
operation = ipc_client.new_get_secret_value()
operation.activate(request)
future_response = operation.get_response()
response = future_response.result(TIMEOUT)
secret_json = json.loads(response.secret_value.secret_string)
# Handle secret value.
```

JavaScript

Example Beispiel: Abrufen eines Secret-Werts

```
import {
    GetSecretValueRequest,
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";

class GetSecretValue {
    private readonly secretId : string;
    private readonly versionStage : string;
    private ipcClient : greengrasscoreipc.Client

    constructor() {
        this.secretId = "<define_your_own_secretId>"
        this.versionStage = "<define_your_own_versionStage>"

        this.getSecretValue().then(r => console.log("Started workflow"));
    }

    private async getSecretValue() {
        try {
            this.ipcClient = await getIpcClient();

            const getSecretValueRequest : GetSecretValueRequest = {
                secretId: this.secretId,
                versionStage: this.versionStage,
            };

            const result = await
 this.ipcClient.getSecretValue(getSecretValueRequest);
```

```
            const secretString = result.secretValue.secretString;
            console.log("Successfully retrieved secret value: " + secretString)
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}

export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

const getSecretValue = new GetSecretValue();
```

Beispiele

Verwenden Sie die folgenden Beispiele, um zu erfahren, wie Sie den Secret Manager IPC-Service in Ihren Komponenten verwenden.

Beispiel: Geheimnis drucken (Python, IPC-Client V1)

Diese Beispielkomponente gibt den Wert eines Secrets aus, das Sie auf dem Core-Gerät bereitstellen.

> **Important**
>
> Diese Beispielkomponente gibt den Wert eines Secrets aus. Verwenden Sie es daher nur mit Secrets, die Testdaten speichern. Verwenden Sie diese Komponente nicht, um den Wert eines Secrets zu drucken, das wichtige Informationen speichert.

Themen

Rezept

Das folgende Beispielrezept definiert einen geheimen ARN-Konfigurationsparameter und ermöglicht es der Komponente, den Wert eines Secrets auf dem Core-Gerät abzurufen.

> **Note**
>
> Wir empfehlen, dass Sie in einer Produktionsumgebung den Umfang der Autorisierungsrichtlinie reduzieren, sodass die Komponente nur die Secrets abruft, die sie verwendet. Sie können den * Platzhalter in eine Liste geheimer ARNs ändern, wenn Sie die Komponente bereitstellen.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PrintSecret",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Prints the value of an AWS Secrets Manager secret.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.SecretManager": {
      "VersionRequirement": "^2.0.0",
      "DependencyType": "HARD"
    }
  },
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "SecretArn": "",
      "accessControl": {
        "aws.greengrass.SecretManager": {
          "com.example.PrintSecret:secrets:1": {
            "policyDescription": "Allows access to a secret.",
            "operations": [
```

```
              "aws.greengrass#GetSecretValue"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk",
        "run": "python3 -u {artifacts:path}/print_secret.py \"{configuration:/
SecretArn}\""
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk",
        "run": "py -3 -u {artifacts:path}/print_secret.py \"{configuration:/
SecretArn}\""
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PrintSecret
ComponentVersion: 1.0.0
ComponentDescription: Prints the value of a Secrets Manager secret.
ComponentPublisher: Amazon
ComponentDependencies:
```

```
  aws.greengrass.SecretManager:
    VersionRequirement: "^2.0.0"
    DependencyType: HARD
ComponentConfiguration:
  DefaultConfiguration:
    SecretArn: ''
    accessControl:
      aws.greengrass.SecretManager:
        com.example.PrintSecret:secrets:1:
          policyDescription: Allows access to a secret.
          operations:
            - aws.greengrass#GetSecretValue
          resources:
            - "*"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: python3 -m pip install --user awsiotsdk
      run: python3 -u {artifacts:path}/print_secret.py "{configuration:/SecretArn}"
  - Platform:
      os: windows
    Lifecycle:
      install: py -3 -m pip install --user awsiotsdk
      run: py -3 -u {artifacts:path}/print_secret.py "{configuration:/SecretArn}"
```

-Artefakte

Die folgende Python-Beispielanwendung zeigt, wie der Secret Manager IPC-Service verwendet wird, um den Wert eines Secrets auf dem Core-Gerät abzurufen.

```
import concurrent.futures
import json
import sys
import traceback

import awsiot.greengrasscoreipc
from awsiot.greengrasscoreipc.model import (
    GetSecretValueRequest,
    GetSecretValueResponse,
    UnauthorizedError
)
```

```
TIMEOUT = 10

if len(sys.argv) == 1:
    print('Provide SecretArn in the component configuration.', file=sys.stdout)
    exit(1)

secret_id = sys.argv[1]

try:
    ipc_client = awsiot.greengrasscoreipc.connect()

    request = GetSecretValueRequest()
    request.secret_id = secret_id
    operation = ipc_client.new_get_secret_value()
    operation.activate(request)
    future_response = operation.get_response()

    try:
        response = future_response.result(TIMEOUT)
        secret_json = json.loads(response.secret_value.secret_string)
        print('Successfully got secret: ' + secret_id)
        print('Secret value: ' + str(secret_json))
    except concurrent.futures.TimeoutError:
        print('Timeout occurred while getting secret: ' + secret_id, file=sys.stderr)
    except UnauthorizedError as e:
        print('Unauthorized error while getting secret: ' + secret_id,
 file=sys.stderr)
        raise e
    except Exception as e:
        print('Exception while getting secret: ' + secret_id, file=sys.stderr)
        raise e
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

Verwendung

Sie können diese Beispielkomponente mit der Secret-Manager-Komponente verwenden, um den Wert eines Secrets auf Ihrem Core-Gerät bereitzustellen und zu drucken.

So erstellen, stellen Sie ein Test-Secret bereit und drucken es

1. Erstellen Sie ein Secrets-Manager-Secret mit Testdaten.

   Linux or Unix

   ```
   aws secretsmanager create-secret \
     --name MyTestGreengrassSecret \
     --secret-string '{"my-secret-key": "my-secret-value"}'
   ```

   Windows Command Prompt (CMD)

   ```
   aws secretsmanager create-secret ^
     --name MyTestGreengrassSecret ^
     --secret-string '{"my-secret-key": "my-secret-value"}'
   ```

   PowerShell

   ```
   aws secretsmanager create-secret `
     --name MyTestGreengrassSecret `
     --secret-string '{"my-secret-key": "my-secret-value"}'
   ```

   Speichern Sie den ARN des Secrets, das Sie in den folgenden Schritten verwenden möchten.

   Weitere Informationen finden Sie unter Erstellen eines Secrets im AWS Secrets Manager - Benutzerhandbuch.

2. Stellen Sie die Secret-Manager-Komponente (`aws.greengrass.SecretManager`) mit der folgenden Konfigurationszusammenführungsaktualisierung bereit. Geben Sie den ARN des Secrets an, das Sie zuvor erstellt haben.

   ```
   {
     "cloudSecrets": [
       {
         "arn": "arn:aws:secretsmanager:us-
   west-2:123456789012:secret:MyTestGreengrassSecret-abcdef"
       }
     ]
   }
   ```

Weitere Informationen finden Sie unter Bereitstellen von AWS IoT Greengrass Komponenten auf Geräten oder im Greengrass-CLI-Bereitstellungsbefehl .

3. Erstellen Sie die Beispielkomponente in diesem Abschnitt und stellen Sie sie mit der folgenden Aktualisierung der Konfigurationszusammenführung bereit. Geben Sie den ARN des Secrets an, das Sie zuvor erstellt haben.

```
{
  "SecretArn": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyTestGreengrassSecret",
  "accessControl": {
    "aws.greengrass.SecretManager": {
      "com.example.PrintSecret:secrets:1": {
        "policyDescription": "Allows access to a secret.",
        "operations": [
          "aws.greengrass#GetSecretValue"
        ],
        "resources": [
          "arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyTestGreengrassSecret-abcdef"
        ]
      }
    }
  }
}
```

Weitere Informationen finden Sie unter Erstellen von AWS IoT Greengrass Komponenten.

4. Zeigen Sie die AWS IoT Greengrass -Core-Softwareprotokolle an, um zu überprüfen, ob die Bereitstellungen erfolgreich sind, und zeigen Sie das `com.example.PrintSecret` Komponentenprotokoll an, um den Secret-Wert zu sehen. Weitere Informationen finden Sie unter Überwachen von AWS IoT Greengrass Protokollen.

Interagieren mit lokalen Schatten

Verwenden Sie den Shadow IPC-Service, um mit lokalen Schatten auf einem Gerät zu interagieren. Das Gerät, mit dem Sie interagieren möchten, kann Ihr Core-Gerät oder ein verbundenes Client-Gerät sein.

Um diese IPC-Operationen zu verwenden, fügen Sie die Shadow Manager-Komponente als Abhängigkeit in Ihre benutzerdefinierte Komponente ein. Anschließend können Sie IPC-Operationen in Ihren benutzerdefinierten Komponenten verwenden, um über den Schattenmanager mit lokalen Schatten auf Ihrem Gerät zu interagieren. Damit benutzerdefinierte Komponenten auf Änderungen des lokalen Schattenstatus reagieren können, können Sie auch den IPC-Service zum Veröffentlichen/Abonnementieren verwenden, um Schattenereignisse zu abonnieren. Weitere Informationen zur Verwendung des Services zum Veröffentlichen/Abonnieren finden Sie unter Lokale Nachrichten veröffentlichen/abonnieren.

> **Note**
>
> Damit ein Core-Gerät mit Client-Geräteschatten interagieren kann, müssen Sie auch die MQTT-Bridge-Komponente konfigurieren und bereitstellen. Weitere Informationen finden Sie unter Schattenmanager für die Kommunikation mit Client-Geräten aktivieren.

Themen

SDK-Mindestversionen

In der folgenden Tabelle sind die Mindestversionen von aufgeführtAWS IoT Device SDK, die Sie für die Interaktion mit lokalen Schatten verwenden müssen.

SDK	Mindestversion	
AWS IoT Device SDK für Java v2	v1.4.0	
AWS IoT Device SDK für Python v2	v1.6.0	

SDK	Mindestversion	
AWS IoT Device SDK für C++ v2	v1.17.0	
AWS IoT Device SDK für JavaScript v2	v1.12.0	

Autorisierung

Um den Shadow IPC-Service in einer benutzerdefinierten Komponente zu verwenden, müssen Sie Autorisierungsrichtlinien definieren, die es Ihrer Komponente ermöglichen, mit Schatten zu interagieren. Informationen zum Definieren von Autorisierungsrichtlinien finden Sie unter Autorisieren Sie Komponenten zur Ausführung von IPC-Vorgängen.

Autorisierungsrichtlinien für die Schatteninteraktion haben die folgenden Eigenschaften.

IPC-Service-ID: `aws.greengrass.ShadowManager`

Operation	Beschreibung	Ressourcen
`aws.greengrass#GetThingShadow`	Ermöglicht einer Komponente, den Schatten eines Objekts abzurufen.	Eine der folgenden Zeichenfolgen: • `$aws/things/`*`thingName`*`/shadow/`, um den Zugriff auf den klassischen Geräteschatten zu ermöglichen. • `$aws/things/`*`thingName`*`/shadow/name/`*`shadowName`*, um den Zugriff auf einen benannten Schatten zu erlauben.

Operation	Beschreibung	Ressourcen
		• `*` , um den Zugriff auf alle Schatten zu ermöglichen.
`aws.greengrass#UpdateThingShadow`	Ermöglicht einer Komponente, den Schatten eines Objekts zu aktualisieren.	Eine der folgenden Zeichenfolgen: • `$aws/things/` *`thingName`* `/shadow/`, um den Zugriff auf den klassischen Geräteschatten zu ermöglichen. • `$aws/things/` *`thingName`* `/shadow/name/` *`shadowName`* , um den Zugriff auf einen benannten Schatten zu erlauben. • `*` , um den Zugriff auf alle Schatten zu ermöglichen.

Operation	Beschreibung	Ressourcen
`aws.greengrass#DeleteThingShadow`	Ermöglicht einer Komponente das Löschen des Schattens eines Objekts.	Eine der folgenden Zeichenfolgen: • `$aws/things/` *`thingName`* `/shadow/`, um den Zugriff auf den klassischen Geräteschatten zu ermöglichen • `$aws/things/` *`thingName`* `/shadow/name/` *`shadowName`*, um den Zugriff auf einen benannten Schatten zu erlauben • `*`, um den Zugriff auf alle Schatten zu ermöglichen.
`aws.greengrass#ListNamedShadowsForThing`	Ermöglicht einer Komponente, die Liste der benannten Schatten für ein Objekt abzurufen.	Eine Objektnamenzeichenfolge, die den Zugriff auf das Objekt ermöglicht, um seine Schatten aufzulisten. Verwenden Sie `*`, um den Zugriff auf alle Objekte zu erlauben.

IPC-Service-ID: `aws.greengrass.ipc.pubsub`

Operation	Beschreibung	Ressourcen
`aws.greengrass#SubscribeToTopic`	Ermöglicht einer Komponente das Abonnieren von	Eine der folgenden Themenzeichenfolgen:

Operation	Beschreibung	Ressourcen
	Nachrichten für die von Ihnen angegebenen Themen.	• *shadowTopicPrefix* /get/accepted • *shadowTopicPrefix* /get/rejected • *shadowTopicPrefix* /delete/accepted • *shadowTopicPrefix* /delete/rejected • *shadowTopicPrefix* /update/accepted • *shadowTopicPrefix* /update/delta • *shadowTopicPrefix* /update/rejected Der Wert des Themenpräfixes hängt vom Typ des Schattens *shadowTopicPrefix* ab: • Klassischer Schatten: $aws/things/ *thingName* /shadow • Benannter Schatten: $aws/things/ *thingName* /shadow/name/ *shadowName* Verwenden Sie *, um den Zugriff auf alle Themen zu erlauben. In Greengrass-Kern v2.6.0 und höher können Sie

Operation	Beschreibung	Ressourcen
		Themen abonnieren, die MQTT-Themen-Platzh alter (# und +) enthalten. Diese Themenzeichenfolge unterstützt MQTT-Them en-Platzhalter als Literalze ichen. Wenn beispielsweise die Autorisierungsrichtlinie einer Komponente Zugriff auf gewährt`test/topic/#`, kann die Komponente abonnieren`test/topic/#` , aber sie kann nicht abonniere `ntest/topic/filter` .

Rezeptvariablen in lokalen Schattenautorisierungsrichtlinien

Wenn Sie v2.6.0 oder höher des Greengrass-Kerns verwenden und die interpolateComponentConfiguration Konfigurationsoption des Greengrass-Kerns auf festlegen`true`, können Sie die -`{iot:thingName}`Rezeptvariable in Autorisierungsrichtlinien verwenden. Mit dieser Funktion können Sie eine einzelne Autorisierungsrichtlinie für eine Gruppe von Core-Geräten konfigurieren, bei der jedes Core-Gerät nur auf seinen eigenen Schatten zugreifen kann. Sie können beispielsweise einer Komponente Zugriff auf die folgende Ressource für Schatten- IPC-Operationen gewähren.

```
$aws/things/{iot:thingName}/shadow/
```

Beispiele für Autorisierungsrichtlinien

Sie können auf die folgenden Beispiele für Autorisierungsrichtlinien verweisen, um Ihnen bei der Konfiguration von Autorisierungsrichtlinien für Ihre Komponenten zu helfen.

Example Beispiel: Erlauben Sie einer Gruppe von -Core-Geräten, mit lokalen Schatten zu interagieren

> **Important**
>
> In diesem Beispiel wird eine Funktion verwendet, die für v2.6.0 und höher der Greengrass-Kernkomponente verfügbar ist. Greengrass-Kern v2.6.0 bietet Unterstützung für die meisten Rezeptvariablen, z. B. `{iot:thingName}`, in Komponentenkonfigurationen. Um dieses Feature zu aktivieren, setzen Sie die interpolateComponentConfiguration Konfigurationsoption des Greengrass-Kerns auf `true`. Ein Beispiel, das für alle Versionen des Greengrass-Kerns funktioniert, finden Sie in der Beispielautorisierungsrichtlinie für ein einzelnes Core-Gerät .

Die folgende Beispielautorisierungsrichtlinie ermöglicht es der Komponente`com.example.MyShadowInteractionComponent`, mit dem klassischen Geräteschatten und dem benannten Schatten `myNamedShadow` für das Core-Gerät zu interagieren, das die Komponente ausführt. Diese Richtlinie ermöglicht es dieser Komponente auch, Nachrichten zu lokalen Themen für diese Schatten zu empfangen.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ShadowManager": {
      "com.example.MyShadowInteractionComponent:shadow:1": {
        "policyDescription": "Allows access to shadows",
        "operations": [
          "aws.greengrass#GetThingShadow",
          "aws.greengrass#UpdateThingShadow",
          "aws.greengrass#DeleteThingShadow"
        ],
        "resources": [
          "$aws/things/{iot:thingName}/shadow",
          "$aws/things/{iot:thingName}/shadow/name/myNamedShadow"
        ]
      },
      "com.example.MyShadowInteractionComponent:shadow:2": {
        "policyDescription": "Allows access to things with shadows",
        "operations": [
          "aws.greengrass#ListNamedShadowsForThing"
```

```
        ],
        "resources": [
          "{iot:thingName}"
        ]
      }
    },
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyShadowInteractionComponent:pubsub:1": {
        "policyDescription": "Allows access to shadow pubsub topics",
        "operations": [
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "$aws/things/{iot:thingName}/shadow/get/accepted",
          "$aws/things/{iot:thingName}/shadow/name/myNamedShadow/get/accepted"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ShadowManager:
    'com.example.MyShadowInteractionComponent:shadow:1':
      policyDescription: 'Allows access to shadows'
      operations:
        - 'aws.greengrass#GetThingShadow'
        - 'aws.greengrass#UpdateThingShadow'
        - 'aws.greengrass#DeleteThingShadow'
      resources:
        - $aws/things/{iot:thingName}/shadow
        - $aws/things/{iot:thingName}/shadow/name/myNamedShadow
    'com.example.MyShadowInteractionComponent:shadow:2':
      policyDescription: 'Allows access to things with shadows'
      operations:
        - 'aws.greengrass#ListNamedShadowsForThing'
      resources:
        - '{iot:thingName}'
  aws.greengrass.ipc.pubsub:
    'com.example.MyShadowInteractionComponent:pubsub:1':
      policyDescription: 'Allows access to shadow pubsub topics'
```

```
      operations:
        - 'aws.greengrass#SubscribeToTopic'
      resources:
        - $aws/things/{iot:thingName}/shadow/get/accepted
        - $aws/things/{iot:thingName}/shadow/name/myNamedShadow/get/accepted
```

Example Beispiel: Erlauben Sie einer Gruppe von Core-Geräten die Interaktion mit Client-Geräteschatten

> **Important**
>
> Für diese Funktion sind Greengrass-Kern v2.6.0 oder höher, Shadow Manager v2.2.0 oder höher und MQTT Bridge v2.2.0 oder höher erforderlich. Sie müssen die MQTT-Brücke konfigurieren, damit der Shadow Manager mit Client-Geräten kommunizieren kann.

Die folgende Beispielautorisierungsrichtlinie ermöglicht es der Komponente`com.example.MyShadowInteractionComponent`, mit allen Geräteschatten für Client-Geräte zu interagieren, deren Namen mit beginnen`MyClientDevice`.

> **Note**
>
> Damit ein Core-Gerät mit Client-Geräteschatten interagieren kann, müssen Sie auch die MQTT-Bridge-Komponente konfigurieren und bereitstellen. Weitere Informationen finden Sie unter Shadow Manager die Kommunikation mit Client-Geräten ermöglichen.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ShadowManager": {
      "com.example.MyShadowInteractionComponent:shadow:1": {
        "policyDescription": "Allows access to shadows",
        "operations": [
          "aws.greengrass#GetThingShadow",
          "aws.greengrass#UpdateThingShadow",
          "aws.greengrass#DeleteThingShadow"
        ],
        "resources": [
```

```
          "$aws/things/MyClientDevice*/shadow",
          "$aws/things/MyClientDevice*/shadow/name/*"
        ]
      },
      "com.example.MyShadowInteractionComponent:shadow:2": {
        "policyDescription": "Allows access to things with shadows",
        "operations": [
          "aws.greengrass#ListNamedShadowsForThing"
        ],
        "resources": [
          "MyClientDevice*"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ShadowManager:
    'com.example.MyShadowInteractionComponent:shadow:1':
      policyDescription: 'Allows access to shadows'
      operations:
        - 'aws.greengrass#GetThingShadow'
        - 'aws.greengrass#UpdateThingShadow'
        - 'aws.greengrass#DeleteThingShadow'
      resources:
        - $aws/things/MyClientDevice*/shadow
        - $aws/things/MyClientDevice*/shadow/name/*
    'com.example.MyShadowInteractionComponent:shadow:2':
      policyDescription: 'Allows access to things with shadows'
      operations:
        - 'aws.greengrass#ListNamedShadowsForThing'
      resources:
        - MyClientDevice*
```

Example Beispiel: Einem einzelnen Core-Gerät erlauben, mit lokalen Schatten zu interagieren

Die folgende Beispielautorisierungsrichtlinie ermöglicht es der Komponente`com.example.MyShadowInteractionComponent`, mit dem klassischen

Geräteschatten und dem benannten Schatten `myNamedShadow` für das Gerät zu interagieren`MyThingName`. Diese Richtlinie ermöglicht es dieser Komponente auch, Nachrichten zu lokalen Themen für diese Schatten zu empfangen.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ShadowManager": {
      "com.example.MyShadowInteractionComponent:shadow:1": {
        "policyDescription": "Allows access to shadows",
        "operations": [
          "aws.greengrass#GetThingShadow",
          "aws.greengrass#UpdateThingShadow",
          "aws.greengrass#DeleteThingShadow"
        ],
        "resources": [
          "$aws/things/MyThingName/shadow",
          "$aws/things/MyThingName/shadow/name/myNamedShadow"
        ]
      },
      "com.example.MyShadowInteractionComponent:shadow:2": {
        "policyDescription": "Allows access to things with shadows",
        "operations": [
          "aws.greengrass#ListNamedShadowsForThing"
        ],
        "resources": [
          "MyThingName"
        ]
      }
    },
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyShadowInteractionComponent:pubsub:1": {
        "policyDescription": "Allows access to shadow pubsub topics",
        "operations": [
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "$aws/things/MyThingName/shadow/get/accepted",
          "$aws/things/MyThingName/shadow/name/myNamedShadow/get/accepted"
        ]
      }
    }
```

```
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ShadowManager:
    'com.example.MyShadowInteractionComponent:shadow:1':
      policyDescription: 'Allows access to shadows'
      operations:
        - 'aws.greengrass#GetThingShadow'
        - 'aws.greengrass#UpdateThingShadow'
        - 'aws.greengrass#DeleteThingShadow'
      resources:
        - $aws/things/MyThingName/shadow
        - $aws/things/MyThingName/shadow/name/myNamedShadow
    'com.example.MyShadowInteractionComponent:shadow:2':
      policyDescription: 'Allows access to things with shadows'
      operations:
        - 'aws.greengrass#ListNamedShadowsForThing'
      resources:
        - MyThingName
  aws.greengrass.ipc.pubsub:
    'com.example.MyShadowInteractionComponent:pubsub:1':
      policyDescription: 'Allows access to shadow pubsub topics'
      operations:
        - 'aws.greengrass#SubscribeToTopic'
      resources:
        - $aws/things/MyThingName/shadow/get/accepted
        - $aws/things/MyThingName/shadow/name/myNamedShadow/get/accepted
```

Example Beispiel: Erlauben Sie einer Gruppe von -Core-Geräten, auf lokale Schattenzustandsänderungen zu reagieren

> **Important**
>
> In diesem Beispiel wird eine Funktion verwendet, die für v2.6.0 und höher der Greengrass-Kernkomponente verfügbar ist. Greengrass-Kern v2.6.0 bietet Unterstützung für die meisten Rezeptvariablen, z. B. `{iot:thingName}`, in Komponentenkonfigurationen. Um dieses Feature zu aktivieren, setzen Sie die interpolateComponentConfiguration

Konfigurationsoption des Greengrass-Kerns auf `true`. Ein Beispiel, das für alle Versionen des Greengrass-Kerns funktioniert, finden Sie in der Beispielautorisierungsrichtlinie für ein einzelnes Core-Gerät .

Die folgende Beispiel-Zugriffskontrollrichtlinie ermöglicht es dem Benutzer`com.example.MyShadowReactiveComponent`, Nachrichten zum `/update/delta` Thema für den klassischen Geräteschatten und den benannten Schatten `myNamedShadow` auf jedem Core-Gerät zu empfangen, das die Komponente ausführt.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyShadowReactiveComponent:pubsub:1": {
        "policyDescription": "Allows access to shadow pubsub topics",
        "operations": [
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "$aws/things/{iot:thingName}/shadow/update/delta",
          "$aws/things/{iot:thingName}/shadow/name/myNamedShadow/update/delta"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ipc.pubsub:
    "com.example.MyShadowReactiveComponent:pubsub:1":
      policyDescription: Allows access to shadow pubsub topics
      operations:
        - 'aws.greengrass#SubscribeToTopic'
      resources:
        - $aws/things/{iot:thingName}/shadow/update/delta
        - $aws/things/{iot:thingName}/shadow/name/myNamedShadow/update/delta
```

Example Beispiel: Erlauben Sie einem einzelnen Core-Gerät, auf Änderungen des lokalen Schattenzustands zu reagieren

Die folgende Beispiel-Zugriffskontrollrichtlinie ermöglicht es dem Benutzer`com.example.MyShadowReactiveComponent`, Nachrichten zum `/update/delta` Thema für den klassischen Geräteschatten und den benannten Schatten `myNamedShadow` für das Gerät zu empfangen`MyThingName`.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyShadowReactiveComponent:pubsub:1": {
        "policyDescription": "Allows access to shadow pubsub topics",
        "operations": [
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "$aws/things/MyThingName/shadow/update/delta",
          "$aws/things/MyThingName/shadow/name/myNamedShadow/update/delta"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ipc.pubsub:
    "com.example.MyShadowReactiveComponent:pubsub:1":
      policyDescription: Allows access to shadow pubsub topics
      operations:
        - 'aws.greengrass#SubscribeToTopic'
      resources:
        - $aws/things/MyThingName/shadow/update/delta
        - $aws/things/MyThingName/shadow/name/myNamedShadow/update/delta
```

GetThingShadow

Rufen Sie den Schatten für ein bestimmtes Objekt ab.

Anforderung

Die Anforderung dieser Operation hat die folgenden Parameter:

`thingName` (Python: `thing_name`)

Der Name des Objekts.

Typ: `string`

`shadowName` (Python: `shadow_name`)

Der Name des Schattens. Um den klassischen Schatten des Objekts anzugeben, setzen Sie diesen Parameter auf eine leere Zeichenfolge (`""`).

> **Warning**
>
> Der AWS IoT Greengrass Service verwendet den `AWSManagedGreengrassV2Deployment` benannten Schatten, um Bereitstellungen zu verwalten, die auf einzelne Core-Geräte abzielen. Dieser benannte Schatten ist für die Verwendung durch den AWS IoT Greengrass Service reserviert. Aktualisieren oder löschen Sie diesen benannten Schatten nicht.

Typ: `string`

Antwort

Die Antwort dieser Operation enthält die folgenden Informationen:

`payload`

Das Antwortstatusdokument als Blob.

Typ: `object`, der die folgenden Informationen enthält:

`state`

Die Statusinformationen.

Dieses Objekt enthält die folgenden Informationen.

`desired`

Die Zustandseigenschaften und -werte, die zur Aktualisierung im Gerät angefordert wurden.

Typ: `map` von Schlüssel-Wert-Paaren

`reported`

Die vom Gerät gemeldeten Zustandseigenschaften und -werte.

Typ: `map` von Schlüssel-Wert-Paaren

`delta`

Der Unterschied zwischen den gewünschten und gemeldeten Zustandseigenschaften und -werten. Diese Eigenschaft ist nur vorhanden, wenn die `reported` Status `desired` und unterschiedlich sind.

Typ: `map` von Schlüssel-Wert-Paaren

`metadata`

Die Zeitstempel für jedes Attribut in den `reported` Abschnitten `desired` und , sodass Sie feststellen können, wann der Status aktualisiert wurde.

Typ: `string`

`timestamp`

Das Epochendatum und die Epochenzeit, zu der die Antwort generiert wurde.

Typ: `integer`

`clientToken` (Python: `clientToken`)

Das Token, das verwendet wird, um die Anforderung und die entsprechende Antwort abzugleichen

Typ: `string`

`version`

Die Version des lokalen Schattendokuments.

Typ: `integer`

Fehler

Dieser Vorgang kann die folgenden Fehler zurückgeben.

`InvalidArgumentsError`

Der lokale Schattenservice kann die Anforderungsparameter nicht validieren. Dies kann der Fall sein, wenn die Anforderung fehlerhafte JSON- oder nicht unterstützte Zeichen enthält.

`ResourceNotFoundError`

Das angeforderte lokale Schattendokument kann nicht gefunden werden.

`ServiceError`

Es ist ein interner Servicefehler aufgetreten oder die Anzahl der Anfragen an den IPC-Service hat die in den `maxTotalLocalRequestsRate` Konfigurationsparametern `maxLocalRequestsPerSecondPerThing` und in der Shadow-Manager-Komponente angegebenen Grenzwerte überschritten.

`UnauthorizedError`

Die Autorisierungsrichtlinie der Komponente enthält keine erforderlichen Berechtigungen für diesen Vorgang.

Beispiele

Die folgenden Beispiele zeigen, wie Sie diese Operation im benutzerdefinierten Komponentencode aufrufen.

Java (IPC client V1)

Example Beispiel: Abrufen eines Objektschattens

> **Note**
>
> In diesem Beispiel wird eine -`IPCUtils`Klasse verwendet, um eine Verbindung zum AWS IoT Greengrass Core IPC-Service herzustellen. Weitere Informationen finden Sie unter Connect zum AWS IoT Greengrass Core IPC-Dienst her.

```
package com.aws.greengrass.docs.samples.ipc;
```

```
import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GetThingShadowResponseHandler;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.model.GetThingShadowRequest;
import software.amazon.awssdk.aws.greengrass.model.GetThingShadowResponse;
import software.amazon.awssdk.aws.greengrass.model.ResourceNotFoundError;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class GetThingShadow {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        // Use the current core device's name if thing name isn't set.
        String thingName = args[0].isEmpty() ? System.getenv("AWS_IOT_THING_NAME") :
 args[0];
        String shadowName = args[1];
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            GetThingShadowResponseHandler responseHandler =
                    GetThingShadow.getThingShadow(ipcClient, thingName,
 shadowName);
            CompletableFuture<GetThingShadowResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                GetThingShadowResponse response =
 futureResponse.get(TIMEOUT_SECONDS,
                        TimeUnit.SECONDS);
                String shadowPayload = new String(response.getPayload(),
 StandardCharsets.UTF_8);
                System.out.printf("Successfully got shadow %s/%s: %s%n", thingName,
 shadowName,
                        shadowPayload);
            } catch (TimeoutException e) {
```

```
                System.err.printf("Timeout occurred while getting shadow: %s/%s%n",
 thingName,
                        shadowName);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.printf("Unauthorized error while getting shadow: %s/
%s%n",
                            thingName, shadowName);
                } else if (e.getCause() instanceof ResourceNotFoundError) {
                    System.err.printf("Unable to find shadow to get: %s/%s%n",
 thingName,
                            shadowName);
                } else {
                    throw e;
                }
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static GetThingShadowResponseHandler
 getThingShadow(GreengrassCoreIPCClient greengrassCoreIPCClient, String thingName,
 String shadowName) {
        GetThingShadowRequest getThingShadowRequest = new GetThingShadowRequest();
        getThingShadowRequest.setThingName(thingName);
        getThingShadowRequest.setShadowName(shadowName);
        return greengrassCoreIPCClient.getThingShadow(getThingShadowRequest,
 Optional.empty());
    }
}
```

Python (IPC client V1)

Example Beispiel: Abrufen eines Objektschattens

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import GetThingShadowRequest
```

```
TIMEOUT = 10

def sample_get_thing_shadow_request(thingName, shadowName):
    try:
        # set up IPC client to connect to the IPC server
        ipc_client = awsiot.greengrasscoreipc.connect()

        # create the GetThingShadow request
        get_thing_shadow_request = GetThingShadowRequest()
        get_thing_shadow_request.thing_name = thingName
        get_thing_shadow_request.shadow_name = shadowName

        # retrieve the GetThingShadow response after sending the request to the IPC
 server
        op = ipc_client.new_get_thing_shadow()
        op.activate(get_thing_shadow_request)
        fut = op.get_response()

        result = fut.result(TIMEOUT)
        return result.payload

    except InvalidArgumentsError as e:
        # add error handling
        ...
    # except ResourceNotFoundError | UnauthorizedError | ServiceError
```

JavaScript

Example Beispiel: Abrufen eines Objektschattens

```
import {
    GetThingShadowRequest
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc';

class GetThingShadow {
    private ipcClient: greengrasscoreipc.Client;
    private thingName: string;
    private shadowName: string;

    constructor() {
        // Define args parameters here
        this.thingName = "<define_your_own_thingName>";
        this.shadowName = "<define_your_own_shadowName>";
```

```
        this.bootstrap();
    }

    async bootstrap() {
        try {
            this.ipcClient = await getIpcClient();
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }

        try {
            await this.handleGetThingShadowOperation(this.thingName,
                this.shadowName);
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }
    }

    async handleGetThingShadowOperation(
        thingName: string,
        shadowName: string
    ) {
        const request: GetThingShadowRequest = {
            thingName: thingName,
            shadowName: shadowName
        };
        const response = await this.ipcClient.getThingShadow(request);
    }
}

export async function getIpcClient() {
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use caseså
        throw err
```

```
    }
}

const startScript = new GetThingShadow();
```

UpdateThingShadow

Aktualisieren Sie den Schatten für das angegebene Objekt. Wenn kein Schatten vorhanden ist, wird ein Schatten erstellt.

Anforderung

Die Anforderung dieser Operation hat die folgenden Parameter:

`thingName` (Python: `thing_name`)

Der Name des Objekts.

Typ: `string`

`shadowName` (Python: `shadow_name`)

Der Name des Schattens. Um den klassischen Schatten des Objekts anzugeben, setzen Sie diesen Parameter auf eine leere Zeichenfolge (`""`).

> **Warning**
>
> Der AWS IoT Greengrass Service verwendet den `AWSManagedGreengrassV2Deployment` benannten Schatten, um Bereitstellungen zu verwalten, die auf einzelne Core-Geräte abzielen. Dieser benannte Schatten ist für die Verwendung durch den AWS IoT Greengrass Service reserviert. Aktualisieren oder löschen Sie diesen benannten Schatten nicht.

Typ: `string`

`payload`

Das Anforderungsstatusdokument als Blob.

Typ: `object`, der die folgenden Informationen enthält:

`state`

Die zu aktualisierenden Statusinformationen. Diese IPC-Operation wirkt sich nur auf die angegebenen Felder aus.

Dieses Objekt enthält die folgenden Informationen. In der Regel verwenden Sie entweder die -`desired`Eigenschaft oder die -`reported`Eigenschaft, aber nicht beides in derselben Anforderung.

`desired`

Die Zustandseigenschaften und -werte, die für die Aktualisierung im Gerät angefordert wurden.

Typ: `map` von Schlüssel-Wert-Paaren

`reported`

Die vom Gerät gemeldeten Zustandseigenschaften und -werte.

Typ: `map` von Schlüssel-Wert-Paaren

`clientToken` (Python: `client_token`)

(Optional) Das Token, das verwendet wird, um die Anforderung und die entsprechende Antwort durch das Client-Token abzugleichen.

Typ: `string`

`version`

(Optional) Die Version des lokalen Schattendokuments, das aktualisiert werden soll. Der Schattenservice verarbeitet die Aktualisierung nur, wenn die angegebene Version mit der neuesten Version übereinstimmt, die er hat.

Typ: `integer`

Antwort

Die Antwort dieser Operation enthält die folgenden Informationen:

`payload`

Das Antwortstatusdokument als Blob.

Typ: `object`, der die folgenden Informationen enthält:

`state`

Die Statusinformationen.

Dieses Objekt enthält die folgenden Informationen.

`desired`

Die Zustandseigenschaften und -werte, die für die Aktualisierung im Gerät angefordert wurden.

Typ: `map` von Schlüssel-Wert-Paaren

`reported`

Die vom Gerät gemeldeten Zustandseigenschaften und -werte.

Typ: `map` von Schlüssel-Wert-Paaren

`delta`

Die vom Gerät gemeldeten Zustandseigenschaften und -werte.

Typ: `map` von Schlüssel-Wert-Paaren

`metadata`

Die Zeitstempel für jedes Attribut in den `reported` Abschnitten `desired` und , sodass Sie feststellen können, wann der Status aktualisiert wurde.

Typ: `string`

`timestamp`

Das Epochendatum und die Epochenzeit, zu der die Antwort generiert wurde.

Typ: `integer`

`clientToken` (Python: `client_token`)

Das Token, das verwendet wird, um die Anforderung und die entsprechende Antwort abzugleichen.

Typ: `string`

`version`

Die Version des lokalen Schattendokuments nach Abschluss des Updates.

Typ: `integer`

Fehler

Dieser Vorgang kann die folgenden Fehler zurückgeben.

`ConflictError`

Beim lokalen Schattenservice ist während des Aktualisierungsvorgangs ein Versionskonflikt aufgetreten. Dies tritt auf, wenn die Version in der Anforderungsnutzlast nicht mit der Version im neuesten verfügbaren lokalen Schattendokument übereinstimmt.

`InvalidArgumentsError`

Der lokale Schattenservice kann die Anforderungsparameter nicht validieren. Dies kann der Fall sein, wenn die Anforderung fehlerhafte JSON- oder nicht unterstützte Zeichen enthält.

Ein gültiger `payload` hat die folgenden Eigenschaften:

- Der `state` Knoten ist vorhanden und ist ein Objekt, das die - `desired` oder -`reported`Statusinformationen enthält.
- Die `reported` Knoten `desired` und sind entweder Objekte oder Null. Mindestens eines dieser Objekte muss gültige Statusinformationen enthalten.
- Die Tiefe der `reported` Objekte `desired` und darf acht Knoten nicht überschreiten.
- Die Länge des `clientToken` Wertes darf 64 Zeichen nicht überschreiten.
- Der `version` Wert muss 1 oder höher sein.

`ServiceError`

Es ist ein interner Servicefehler aufgetreten oder die Anzahl der Anfragen an den IPC-Service hat die in den `maxTotalLocalRequestsRate` Konfigurationsparametern `maxLocalRequestsPerSecondPerThing` und in der Shadow-Manager-Komponente angegebenen Grenzwerte überschritten.

`UnauthorizedError`

Die Autorisierungsrichtlinie der Komponente enthält keine erforderlichen Berechtigungen für diesen Vorgang.

Beispiele

Die folgenden Beispiele zeigen, wie Sie diese Operation im benutzerdefinierten Komponentencode aufrufen.

Java (IPC client V1)

Example Beispiel: Aktualisieren eines Objektschattens

> **Note**
>
> In diesem Beispiel wird eine -`IPCUtils`Klasse verwendet, um eine Verbindung zum AWS IoT Greengrass Core IPC-Service herzustellen. Weitere Informationen finden Sie unter Connect zum AWS IoT Greengrass Core IPC-Dienst her.

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.UpdateThingShadowResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.aws.greengrass.model.UpdateThingShadowRequest;
import software.amazon.awssdk.aws.greengrass.model.UpdateThingShadowResponse;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class UpdateThingShadow {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        // Use the current core device's name if thing name isn't set.
        String thingName = args[0].isEmpty() ? System.getenv("AWS_IOT_THING_NAME") :
 args[0];
        String shadowName = args[1];
```

```
        byte[] shadowPayload = args[2].getBytes(StandardCharsets.UTF_8);
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            UpdateThingShadowResponseHandler responseHandler =
                    UpdateThingShadow.updateThingShadow(ipcClient, thingName,
 shadowName,
                            shadowPayload);
            CompletableFuture<UpdateThingShadowResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                System.out.printf("Successfully updated shadow: %s/%s%n", thingName,
 shadowName);
            } catch (TimeoutException e) {
                System.err.printf("Timeout occurred while updating shadow: %s/%s%n",
 thingName,
                        shadowName);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.printf("Unauthorized error while updating shadow: %s/
%s%n",
                            thingName, shadowName);
                } else {
                    throw e;
                }
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static UpdateThingShadowResponseHandler
 updateThingShadow(GreengrassCoreIPCClient greengrassCoreIPCClient, String
 thingName, String shadowName, byte[] shadowPayload) {
        UpdateThingShadowRequest updateThingShadowRequest = new
 UpdateThingShadowRequest();
        updateThingShadowRequest.setThingName(thingName);
        updateThingShadowRequest.setShadowName(shadowName);
```

```
        updateThingShadowRequest.setPayload(shadowPayload);
        return greengrassCoreIPCClient.updateThingShadow(updateThingShadowRequest,
                Optional.empty());
    }
}
```

Python (IPC client V1)

Example Beispiel: Aktualisieren eines Objektschattens

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import UpdateThingShadowRequest

TIMEOUT = 10

def sample_update_thing_shadow_request(thingName, shadowName, payload):
    try:
        # set up IPC client to connect to the IPC server
        ipc_client = awsiot.greengrasscoreipc.connect()

        # create the UpdateThingShadow request
        update_thing_shadow_request = UpdateThingShadowRequest()
        update_thing_shadow_request.thing_name = thingName
        update_thing_shadow_request.shadow_name = shadowName
        update_thing_shadow_request.payload = payload

        # retrieve the UpdateThingShadow response after sending the request to the
 IPC server
        op = ipc_client.new_update_thing_shadow()
        op.activate(update_thing_shadow_request)
        fut = op.get_response()

        result = fut.result(TIMEOUT)
        return result.payload

    except InvalidArgumentsError as e:
        # add error handling
    ...
    # except ConflictError | UnauthorizedError | ServiceError
```

JavaScript

Example Beispiel: Aktualisieren eines Objektschattens

```
import {
    UpdateThingShadowRequest
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc';

class UpdateThingShadow {
    private ipcClient: greengrasscoreipc.Client;
    private thingName: string;
    private shadowName: string;
    private shadowDocumentStr: string;

    constructor() {
        // Define args parameters here

        this.thingName = "<define_your_own_thingName>";
        this.shadowName = "<define_your_own_shadowName>";
        this.shadowDocumentStr = "<define_your_own_payload>";

        this.bootstrap();
    }

    async bootstrap() {
        try {
            this.ipcClient = await getIpcClient();
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }

        try {
            await this.handleUpdateThingShadowOperation(
                this.thingName,
                this.shadowName,
                this.shadowDocumentStr);
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }
    }
```

```
    async handleUpdateThingShadowOperation(
        thingName: string,
        shadowName: string,
        payloadStr: string
    ) {
        const request: UpdateThingShadowRequest = {
            thingName: thingName,
            shadowName: shadowName,
            payload: payloadStr
        }
        // make the UpdateThingShadow request
        const response = await this.ipcClient.updateThingShadow(request);
    }
}

export async function getIpcClient() {
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

const startScript = new UpdateThingShadow();
```

DeleteThingShadow

Löscht das Schattengerät für das angegebene Objekt.

Ab Shadow Manager v2.0.4 erhöht das Löschen eines Schattens die Versionsnummer. Wenn Sie beispielsweise den Schatten `MyThingShadow` in Version 1 löschen, ist die Version des gelöschten Schattens 2. Wenn Sie dann einen Schatten mit dem Namen neu erstellen`MyThingShadow`, ist die Version für diesen Schatten 3.

Anforderung

Die Anforderung dieser Operation hat die folgenden Parameter:

`thingName` (Python: `thing_name`)

Der Name des Objekts.

Typ: `string`

`shadowName` (Python: `shadow_name`)

Der Name des Schattens. Um den klassischen Schatten des Objekts anzugeben, setzen Sie diesen Parameter auf eine leere Zeichenfolge ("").

> **Warning**
>
> Der AWS IoT Greengrass Service verwendet den `AWSManagedGreengrassV2Deployment` benannten Schatten, um Bereitstellungen zu verwalten, die auf einzelne Core-Geräte abzielen. Dieser benannte Schatten ist für die Verwendung durch den AWS IoT Greengrass Service reserviert. Aktualisieren oder löschen Sie diesen benannten Schatten nicht.

Typ: `string`

Antwort

Die Antwort dieser Operation enthält die folgenden Informationen:

`payload`

Ein leeres Antwortstatusdokument.

Fehler

Dieser Vorgang kann die folgenden Fehler zurückgeben.

`InvalidArgumentsError`

Der lokale Schattenservice kann die Anforderungsparameter nicht validieren. Dies kann der Fall sein, wenn die Anforderung fehlerhafte JSON- oder nicht unterstützte Zeichen enthält.

`ResourceNotFoundError`

Das angeforderte lokale Schattendokument kann nicht gefunden werden.

`ServiceError`

Es ist ein interner Servicefehler aufgetreten oder die Anzahl der Anfragen an den IPC-Service hat die in den `maxTotalLocalRequestsRate` Konfigurationsparametern `maxLocalRequestsPerSecondPerThing` und in der Shadow-Manager-Komponente angegebenen Grenzwerte überschritten.

`UnauthorizedError`

Die Autorisierungsrichtlinie der Komponente enthält keine erforderlichen Berechtigungen für diesen Vorgang.

Beispiele

Die folgenden Beispiele zeigen, wie Sie diese Operation im benutzerdefinierten Komponentencode aufrufen.

Java (IPC client V1)

Example Beispiel: Löschen eines Objektschattens

> **Note**
>
> In diesem Beispiel wird eine -`IPCUtils`Klasse verwendet, um eine Verbindung zum AWS IoT Greengrass Core IPC-Service herzustellen. Weitere Informationen finden Sie unter Connect zum AWS IoT Greengrass Core IPC-Dienst her.

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.DeleteThingShadowResponseHandler;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.model.DeleteThingShadowRequest;
```

```
import software.amazon.awssdk.aws.greengrass.model.DeleteThingShadowResponse;
import software.amazon.awssdk.aws.greengrass.model.ResourceNotFoundError;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class DeleteThingShadow {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        // Use the current core device's name if thing name isn't set.
        String thingName = args[0].isEmpty() ? System.getenv("AWS_IOT_THING_NAME") :
 args[0];
        String shadowName = args[1];
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            DeleteThingShadowResponseHandler responseHandler =
                    DeleteThingShadow.deleteThingShadow(ipcClient, thingName,
 shadowName);
            CompletableFuture<DeleteThingShadowResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                System.out.printf("Successfully deleted shadow: %s/%s%n", thingName,
 shadowName);
            } catch (TimeoutException e) {
                System.err.printf("Timeout occurred while deleting shadow: %s/%s%n",
 thingName,
                        shadowName);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.printf("Unauthorized error while deleting shadow: %s/
%s%n",
                            thingName, shadowName);
                } else if (e.getCause() instanceof ResourceNotFoundError) {
```

```
                    System.err.printf("Unable to find shadow to delete: %s/%s%n",
 thingName,
                            shadowName);
                } else {
                    throw e;
                }
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static DeleteThingShadowResponseHandler
 deleteThingShadow(GreengrassCoreIPCClient greengrassCoreIPCClient, String
 thingName, String shadowName) {
        DeleteThingShadowRequest deleteThingShadowRequest = new
 DeleteThingShadowRequest();
        deleteThingShadowRequest.setThingName(thingName);
        deleteThingShadowRequest.setShadowName(shadowName);
        return greengrassCoreIPCClient.deleteThingShadow(deleteThingShadowRequest,
                Optional.empty());
    }
}
```

Python (IPC client V1)

Example Beispiel: Löschen eines Objektschattens

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import DeleteThingShadowRequest

TIMEOUT = 10

def sample_delete_thing_shadow_request(thingName, shadowName):
    try:
        # set up IPC client to connect to the IPC server
        ipc_client = awsiot.greengrasscoreipc.connect()

        # create the DeleteThingShadow request
```

```
        delete_thing_shadow_request = DeleteThingShadowRequest()
        delete_thing_shadow_request.thing_name = thingName
        delete_thing_shadow_request.shadow_name = shadowName

        # retrieve the DeleteThingShadow response after sending the request to the
 IPC server
        op = ipc_client.new_delete_thing_shadow()
        op.activate(delete_thing_shadow_request)
        fut = op.get_response()

        result = fut.result(TIMEOUT)
        return result.payload

    except InvalidArgumentsError as e:
        # add error handling
    ...
    # except ResourceNotFoundError | UnauthorizedError | ServiceError
```

JavaScript

Example Beispiel: Löschen eines Objektschattens

```
import {
    DeleteThingShadowRequest
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc';

class DeleteThingShadow {
    private ipcClient: greengrasscoreipc.Client;
    private thingName: string;
    private shadowName: string;

    constructor() {
        // Define args parameters here
        this.thingName = "<define_your_own_thingName>";
        this.shadowName = "<define_your_own_shadowName>";
        this.bootstrap();
    }

    async bootstrap() {
        try {
            this.ipcClient = await getIpcClient();
        } catch (err) {
            // parse the error depending on your use cases
```

```
            throw err
        }

        try {
            await this.handleDeleteThingShadowOperation(this.thingName,
 this.shadowName)
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }
    }

    async handleDeleteThingShadowOperation(thingName: string, shadowName: string) {
        const request: DeleteThingShadowRequest = {
            thingName: thingName,
            shadowName: shadowName
        }
        // make the DeleteThingShadow request
        const response = await this.ipcClient.deleteThingShadow(request);
    }
}

export async function getIpcClient() {
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

const startScript = new DeleteThingShadow();
```

ListNamedShadowsForThing

Listen Sie die benannten Schatten für das angegebene Objekt auf.

Anforderung

Die Anforderung dieser Operation hat die folgenden Parameter:

`thingName` (Python: `thing_name`)

Der Name des Objekts.

Typ: `string`

`pageSize` (Python: `page_size`)

(Optional) Die Anzahl der Schattennamen, die bei jedem Aufruf zurückgegeben werden sollen.

Typ: `integer`

Standard: 25

Maximum: 100

`nextToken` (Python: `next_token`)

(Optional) Das Token, das den nächsten Ergebnissatz abrufen soll. Dieser Wert wird für nach Seiten organisierte Ergebnisse zurückgegeben und in dem Aufruf verwendet, der die nächste Seite zurückgibt.

Typ: `string`

Antwort

Die Antwort dieser Operation enthält die folgenden Informationen:

`results`

Die Liste der Schattennamen.

Typ: `array`

`timestamp`

(Optional) Das Datum und die Uhrzeit, zu der die Antwort generiert wurde.

Typ: `integer`

`nextToken` (Python: `next_token`)

(Optional) Der Token-Wert, der in ausgelagerten Anforderungen verwendet werden soll, um die nächste Seite in der Sequenz abzurufen. Dieses Token ist nicht vorhanden, wenn keine Schattennamen mehr zurückgegeben werden müssen.

Typ: `string`

> **Note**
>
> Wenn die angeforderte Seitengröße genau mit der Anzahl der Schattennamen in der Antwort übereinstimmt, ist dieses Token vorhanden. Bei Verwendung gibt es jedoch eine leere Liste zurück.

Fehler

Dieser Vorgang kann die folgenden Fehler zurückgeben.

`InvalidArgumentsError`

Der lokale Schattenservice kann die Anforderungsparameter nicht validieren. Dies kann der Fall sein, wenn die Anforderung fehlerhafte JSON- oder nicht unterstützte Zeichen enthält.

`ResourceNotFoundError`

Das angeforderte lokale Schattendokument kann nicht gefunden werden.

`ServiceError`

Es ist ein interner Servicefehler aufgetreten oder die Anzahl der Anfragen an den IPC-Service hat die in den `maxTotalLocalRequestsRate` Konfigurationsparametern `maxLocalRequestsPerSecondPerThing` und in der Shadow-Manager-Komponente angegebenen Grenzwerte überschritten.

`UnauthorizedError`

Die Autorisierungsrichtlinie der Komponente enthält keine erforderlichen Berechtigungen für diesen Vorgang.

Beispiele

Die folgenden Beispiele veranschaulichen, wie Sie diese Operation im benutzerdefinierten Komponentencode aufrufen.

Java (IPC client V1)

Example Beispiel: Auflisten der benannten Schatten eines Objekts

> **Note**
>
> In diesem Beispiel wird eine -`IPCUtils`Klasse verwendet, um eine Verbindung zum AWS IoT Greengrass Core IPC-Service herzustellen. Weitere Informationen finden Sie unter Connect zum AWS IoT Greengrass Core IPC-Dienst her.

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import
 software.amazon.awssdk.aws.greengrass.ListNamedShadowsForThingResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.ListNamedShadowsForThingRequest;
import
 software.amazon.awssdk.aws.greengrass.model.ListNamedShadowsForThingResponse;
import software.amazon.awssdk.aws.greengrass.model.ResourceNotFoundError;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class ListNamedShadowsForThing {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
```

```
        // Use the current core device's name if thing name isn't set.
        String thingName = args[0].isEmpty() ? System.getenv("AWS_IOT_THING_NAME") :
 args[0];
        try (EventStreamRPCConnection eventStreamRPCConnection =
                    IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            List<String> namedShadows = new ArrayList<>();
            String nextToken = null;
            try {
                // Send additional requests until there's no pagination token in the
 response.
                do {
                    ListNamedShadowsForThingResponseHandler responseHandler =

 ListNamedShadowsForThing.listNamedShadowsForThing(ipcClient, thingName,
                                    nextToken, 25);
                    CompletableFuture<ListNamedShadowsForThingResponse>
 futureResponse =
                            responseHandler.getResponse();
                    ListNamedShadowsForThingResponse response =
                            futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                    List<String> responseNamedShadows = response.getResults();
                    namedShadows.addAll(responseNamedShadows);
                    nextToken = response.getNextToken();
                } while (nextToken != null);
                System.out.printf("Successfully got named shadows for thing %s: %s
%n", thingName,
                        String.join(",", namedShadows));
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while listing named shadows for
 thing: " + thingName);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.println("Unauthorized error while listing named
 shadows for " +
                            "thing: " + thingName);
                } else if (e.getCause() instanceof ResourceNotFoundError) {
                    System.err.println("Unable to find thing to list named shadows:
 " + thingName);
                } else {
                    throw e;
                }
            }
```

```
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static ListNamedShadowsForThingResponseHandler
 listNamedShadowsForThing(GreengrassCoreIPCClient greengrassCoreIPCClient, String
 thingName, String nextToken, int pageSize) {
        ListNamedShadowsForThingRequest listNamedShadowsForThingRequest =
                new ListNamedShadowsForThingRequest();
        listNamedShadowsForThingRequest.setThingName(thingName);
        listNamedShadowsForThingRequest.setNextToken(nextToken);
        listNamedShadowsForThingRequest.setPageSize(pageSize);
        return
 greengrassCoreIPCClient.listNamedShadowsForThing(listNamedShadowsForThingRequest,
                Optional.empty());
    }
}
```

Python (IPC client V1)

Example Beispiel: Auflisten der benannten Schatten eines Objekts

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import ListNamedShadowsForThingRequest

TIMEOUT = 10

def sample_list_named_shadows_for_thing_request(thingName, nextToken, pageSize):
    try:
        # set up IPC client to connect to the IPC server
        ipc_client = awsiot.greengrasscoreipc.connect()

        # create the ListNamedShadowsForThingRequest request
        list_named_shadows_for_thing_request = ListNamedShadowsForThingRequest()
        list_named_shadows_for_thing_request.thing_name = thingName
        list_named_shadows_for_thing_request.next_token = nextToken
        list_named_shadows_for_thing_request.page_size = pageSize
```

```
        # retrieve the ListNamedShadowsForThingRequest response after sending the
 request to the IPC server
        op = ipc_client.new_list_named_shadows_for_thing()
        op.activate(list_named_shadows_for_thing_request)
        fut = op.get_response()

        list_result = fut.result(TIMEOUT)

        # additional returned fields
        timestamp = list_result.timestamp
        next_token = result.next_token
        named_shadow_list = list_result.results

        return named_shadow_list, next_token, timestamp

    except InvalidArgumentsError as e:
        # add error handling
    ...
    # except ResourceNotFoundError | UnauthorizedError | ServiceError
```

JavaScript

Example Beispiel: Auflisten der benannten Schatten eines Objekts

```
import {
    ListNamedShadowsForThingRequest
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc';

class listNamedShadowsForThing {
    private ipcClient: greengrasscoreipc.Client;
    private thingName: string;
    private pageSizeStr: string;
    private nextToken: string;

    constructor() {
        // Define args parameters here
        this.thingName = "<define_your_own_thingName>";
        this.pageSizeStr = "<define_your_own_pageSize>";
        this.nextToken = "<define_your_own_token>";
        this.bootstrap();
    }

    async bootstrap() {
```

```
        try {
            this.ipcClient = await getIpcClient();
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }

        try {
            await this.handleListNamedShadowsForThingOperation(this.thingName,
                this.nextToken, this.pageSizeStr);
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }
    }

    async handleListNamedShadowsForThingOperation(
        thingName: string,
        nextToken: string,
        pageSizeStr: string
    ) {
        let request: ListNamedShadowsForThingRequest = {
            thingName: thingName,
            nextToken: nextToken,
        };
        if (pageSizeStr) {
            request.pageSize = parseInt(pageSizeStr);
        }
        // make the ListNamedShadowsForThing request
        const response = await this.ipcClient.listNamedShadowsForThing(request);
        const shadowNames = response.results;
    }
}

export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
```

```
        // parse the error depending on your use cases
        throw err
    }
}

const startScript = new listNamedShadowsForThing();
```

Verwalten von lokalen Bereitstellungen und Komponenten

> **Note**
>
> Diese Funktion ist für v2.6.0 und höher der Greengrass-Kernkomponente verfügbar.

Verwenden Sie den Greengrass CLI IPC-Service, um lokale Bereitstellungen und Greengrass-Komponenten auf dem Core-Gerät zu verwalten.

Um diese IPC-Operationen zu verwenden, fügen Sie Version 2.6.0 oder höher der Greengrass-CLI-Komponente als Abhängigkeit in Ihre benutzerdefinierte Komponente ein. Sie können dann IPC-Operationen in Ihren benutzerdefinierten Komponenten verwenden, um Folgendes zu tun:

- Erstellen Sie lokale Bereitstellungen, um Greengrass-Komponenten auf dem Core-Gerät zu ändern und zu konfigurieren.
- Starten und stoppen Sie Greengrass-Komponenten auf dem Core-Gerät neu.
- Generieren Sie ein Passwort, mit dem Sie sich bei der lokalen Debug-Konsole anmelden können.

Themen

- StopComponent
- CreateDebugPassword

SDK-Mindestversionen

In der folgenden Tabelle sind die Mindestversionen von aufgeführtAWS IoT Device SDK, die Sie für die Interaktion mit dem Greengrass CLI IPC-Service verwenden müssen.

SDK	Mindestversion	
AWS IoT Device SDK für Java v2	v1.2.10	
AWS IoT Device SDK für Python v2	v1.5.3	
AWS IoT Device SDK für C++ v2	v1.17.0	
AWS IoT Device SDK für JavaScript v2	v1.12.0	

Autorisierung

Um den Greengrass CLI IPC-Service in einer benutzerdefinierten Komponente zu verwenden, müssen Sie Autorisierungsrichtlinien definieren, die es Ihrer Komponente ermöglichen, lokale Bereitstellungen und Komponenten zu verwalten. Informationen zum Definieren von Autorisierungsrichtlinien finden Sie unter Autorisieren Sie Komponenten zur Ausführung von IPC-Vorgängen.

Autorisierungsrichtlinien für die Greengrass-CLI haben die folgenden Eigenschaften.

IPC-Service-ID: `aws.greengrass.Cli`

Operation	Beschreibung	Ressourcen
`aws.greengrass#CreateLocalDeployment`	Ermöglicht einer Komponente das Erstellen einer lokalen	*

Operation	Beschreibung	Ressourcen
	Bereitstellung auf dem Core-Gerät.	
`aws.greengrass#ListLocalDeployments`	Ermöglicht einer Komponente, lokale Bereitstellungen auf dem Core-Gerät aufzulisten.	*
`aws.greengrass#GetLocalDeploymentStatus`	Ermöglicht einer Komponente, den Status einer lokalen Bereitstellung auf dem Core-Gerät abzurufen.	Eine lokale Bereitstellungs-ID oder , * um den Zugriff auf alle lokalen Bereitstellungen zu ermöglichen.
`aws.greengrass#ListComponents`	Ermöglicht einer Komponente das Auflisten von Komponenten auf dem Core-Gerät.	*
`aws.greengrass#GetComponentDetails`	Ermöglicht es einer Komponente, Details zu einer Komponente auf dem Core-Gerät abzurufen.	Ein Komponentenname, z. B. oder `com.example.HelloWorld` , * um den Zugriff auf alle Komponenten zu ermöglichen.
`aws.greengrass#RestartComponent`	Ermöglicht einer Komponente den Neustart einer Komponente auf dem Core-Gerät.	Ein Komponentenname, z. B. oder `com.example.HelloWorld` , * um den Zugriff auf alle Komponenten zu ermöglichen.
`aws.greengrass#StopComponent`	Ermöglicht einer Komponente, eine Komponente auf dem Core-Gerät zu stoppen.	Ein Komponentenname, z. B. oder `com.example.HelloWorld` , * um den Zugriff auf alle Komponenten zu ermöglichen.

Operation	Beschreibung	Ressourcen
`aws.greengrass#CreateDebugPassword`	Ermöglicht es einer Komponente, ein Passwort zu generieren, mit dem sie sich bei der lokalen Debug-Konsolenkomponente anmeldet.	*

Example Beispiel für eine Autorisierungsrichtlinie

Die folgenden Beispielautorisierungsrichtlinien ermöglichen es einer Komponente, lokale Bereitstellungen zu erstellen, alle lokalen Bereitstellungen und Komponenten anzuzeigen und eine Komponente mit dem Namen neu zu starten und zu stoppen`com.example.HelloWorld`.

```
{
  "accessControl": {
    "aws.greengrass.Cli": {
      "com.example.MyLocalManagerComponent:cli:1": {
        "policyDescription": "Allows access to create local deployments and view
 deployments and components.",
        "operations": [
          "aws.greengrass#CreateLocalDeployment",
          "aws.greengrass#ListLocalDeployments",
          "aws.greengrass#GetLocalDeploymentStatus",
          "aws.greengrass#ListComponents",
          "aws.greengrass#GetComponentDetails"
        ],
        "resources": [
          "*"
        ]
      }
    },
    "aws.greengrass.Cli": {
      "com.example.MyLocalManagerComponent:cli:2": {
        "policyDescription": "Allows access to restart and stop the Hello World
 component.",
        "operations": [
          "aws.greengrass#RestartComponent",
          "aws.greengrass#StopComponent"
        ],
        "resources": [
```

```
          "com.example.HelloWorld"
        ]
      }
    }
  }
}
```

CreateLocalDeployment

Erstellen oder aktualisieren Sie eine lokale Bereitstellung mit bestimmten Komponentenrezepten, Artefakten und Laufzeitargumenten.

Dieser Vorgang bietet die gleiche Funktionalität wie der Befehl „create“ für die Bereitstellung in der Greengrass-CLI.

Anforderung

Die Anforderung dieser Operation hat die folgenden Parameter:

`recipeDirectoryPath` (Python: `recipe_directory_path`)

(Optional) Der absolute Pfad zu dem Ordner, der Komponentenrezeptdateien enthält.

`artifactDirectoryPath` (Python: `artifact_directory_path`)

(Optional) Der absolute Pfad zu dem Ordner, der die Artefaktdateien enthält, die in die Bereitstellung aufgenommen werden sollen. Der Ordner Artefakte muss die folgende Ordnerstruktur enthalten:

```
/path/to/artifact/folder/component-name/component-version/artifacts
```

`rootComponentVersionsToAdd` (Python: `root_component_versions_to_add`)

(Optional) Die Komponentenversionen, die auf dem Core-Gerät installiert werden sollen. Dieses Objekt, `ComponentToVersionMap`, ist eine Zuordnung, die die folgenden Schlüssel-Wert-Paare enthält:

`key`

Der Name der Komponente.

`value`

Die Version der Komponente.

`rootComponentsToRemove` (Python: `root_components_to_remove`)

(Optional) Die Komponenten, die vom Core-Gerät deinstalliert werden sollen. Geben Sie eine Liste an, in der jeder Eintrag der Name einer Komponente ist.

`componentToConfiguration` (Python: `component_to_configuration`)

(Optional) Die Konfiguration wird für jede Komponente in der Bereitstellung aktualisiert. Dieses Objekt, `ComponentToConfiguration`, ist eine Zuordnung, die die folgenden Schlüssel-Wert-Paare enthält:

`key`

Der Name der Komponente.

`value`

Das Konfigurations-Update-JSON-Objekt für die Komponente. Das JSON-Objekt muss das folgende Format haben.

```
{
  "MERGE": {
    "config-key": "config-value"
  },
  "RESET": [
    "path/to/reset/"
  ]
}
```

Weitere Informationen zu Konfigurationsaktualisierungen finden Sie unter Komponentenkonfigurationen aktualisieren.

`componentToRunWithInfo` (Python: `component_to_run_with_info`)

(Optional) Die Laufzeitkonfiguration für jede Komponente in der Bereitstellung. Diese Konfiguration umfasst den Systembenutzer, der die Prozesse jeder Komponente besitzt, und die Systemlimits, die für jede Komponente gelten sollen. Dieses Objekt, `ComponentToRunWithInfo`, ist eine Zuordnung, die die folgenden Schlüssel-Wert-Paare enthält:

`key`

Der Name der Komponente.

`value`

Die Laufzeitkonfiguration für die Komponente. Wenn Sie einen Laufzeitkonfigurationsparameter weglassen, verwendet die -AWS IoT GreengrassCore-Software die Standardwerte, die Sie für den Greengrass-Kern konfigurieren. Dieses Objekt, `RunWithInfo`, enthält die folgenden Informationen:

`posixUser` (Python: `posix_user`)

(Optional) Der POSIX-Systembenutzer und optional die Gruppe, die zum Ausführen dieser Komponente auf Linux-Core-Geräten verwendet werden sollen. Der Benutzer und die Gruppe, falls angegeben, müssen auf jedem Linux-Core-Gerät vorhanden sein. Geben Sie den Benutzer und die Gruppe durch einen Doppelpunkt (`:`) getrennt im folgenden Format an: `user:group`. Die Gruppe ist optional. Wenn Sie keine Gruppe angeben, verwendet die AWS IoT Greengrass Core-Software die primäre Gruppe für den Benutzer. Weitere Informationen finden Sie unter Konfigurieren Sie den Benutzer, der die Komponenten ausführt.

`windowsUser` (Python: `windows_user`)

(Optional) Der Windows-Benutzer, der zum Ausführen dieser Komponente auf Windows-Core-Geräten verwendet werden soll. Der Benutzer muss auf jedem Windows-Core-Gerät vorhanden sein und sein Name und sein Passwort müssen in der Credentials Manager-Instance des LocalSystem Kontos gespeichert werden. Weitere Informationen finden Sie unter Konfigurieren Sie den Benutzer, der die Komponenten ausführt.

`systemResourceLimits` (Python: `system_resource_limits`)

(Optional) Die Systemressourcenlimits, die auf die Prozesse dieser Komponente angewendet werden sollen. Sie können Systemressourcenlimits auf generische und nicht containerisierte Lambda-Komponenten anwenden. Weitere Informationen finden Sie unter Konfigurieren Sie die Systemressourcenlimits für Komponenten.

AWS IoT Greengrass unterstützt diese Funktion derzeit nicht auf Windows-Core-Geräten.

Dieses Objekt, `SystemResourceLimits`, enthält die folgenden Informationen:

`cpus`

(Optional) Die maximale CPU-Zeit, die die Prozesse dieser Komponente auf dem Core-Gerät verwenden können. Die gesamte CPU-Zeit eines Core-Geräts entspricht der

Anzahl der CPU-Kerne des Geräts. Auf einem Core-Gerät mit 4 CPU-Kernen können Sie diesen Wert beispielsweise auf setzen, 2 um die Prozesse dieser Komponente auf eine Auslastung von 50 Prozent jedes CPU-Kerns zu beschränken. Auf einem Gerät mit 1 CPU-Kern können Sie diesen Wert auf setzen, `0.25` um die Prozesse dieser Komponente auf eine CPU-Auslastung von 25 Prozent zu beschränken. Wenn Sie diesen Wert auf eine Zahl festlegen, die größer als die Anzahl der CPU-Kerne ist, schränkt die AWS IoT Greengrass Core-Software die CPU-Auslastung der Komponente nicht ein.

`memory`

(Optional) Die maximale Menge an RAM (in Kilobyte), die die Prozesse dieser Komponente auf dem Core-Gerät verwenden können.

`groupName` (Python: `group_name`)

(Optional) Der Name der Objektgruppe, auf die mit dieser Bereitstellung abgezielt werden soll.

Antwort

Die Antwort dieser Operation enthält die folgenden Informationen:

`deploymentId` (Python: `deployment_id`)

Die ID der lokalen Bereitstellung, die die Anforderung erstellt hat.

ListLocalDeployments

Ruft den Status der letzten 10 lokalen Bereitstellungen ab.

Dieser Vorgang bietet die gleiche Funktionalität wie der Bereitstellungslistenbefehl in der Greengrass-CLI.

Anforderung

Die Anforderung dieser Operation hat keine Parameter.

Antwort

Die Antwort dieser Operation enthält die folgenden Informationen:

`localDeployments` (Python: `local_deployments`)

Die Liste der lokalen Bereitstellungen. Jedes Objekt in dieser Liste ist ein `LocalDeployment` Objekt, das die folgenden Informationen enthält:

`deploymentId` (Python: `deployment_id`)

Die ID der lokalen Bereitstellung.

`status`

Der Status der lokalen Bereitstellung. Diese Aufzählung, `DeploymentStatus`, hat die folgenden Werte:

- `QUEUED`
- `IN_PROGRESS`
- `SUCCEEDED`
- `FAILED`

GetLocalDeploymentStatus

Ruft den Status einer lokalen Bereitstellung ab.

Dieser Vorgang bietet die gleiche Funktionalität wie der Bereitstellungsstatusbefehl in der Greengrass-CLI.

Anforderung

Die Anforderung dieser Operation hat die folgenden Parameter:

`deploymentId` (Python: `deployment_id`)

Die ID der lokalen Bereitstellung, die abgerufen werden soll.

Antwort

Die Antwort dieser Operation enthält die folgenden Informationen:

`deployment`

Die lokale Bereitstellung. Dieses Objekt, `LocalDeployment`, enthält die folgenden Informationen:

`deploymentId` (Python: `deployment_id`)

Die ID der lokalen Bereitstellung.

`status`

Der Status der lokalen Bereitstellung. Diese Aufzählung, `DeploymentStatus`, hat die folgenden Werte:

- `QUEUED`
- `IN_PROGRESS`
- `SUCCEEDED`
- `FAILED`

ListComponents

Ruft den Namen, die Version, den Status und die Konfiguration jeder Root-Komponente auf dem Core-Gerät ab. Eine Stammkomponente ist eine Komponente, die Sie in einer Bereitstellung angeben. Diese Antwort enthält keine Komponenten, die als Abhängigkeiten anderer Komponenten installiert sind.

Diese Operation bietet die gleiche Funktionalität wie der Komponentenlistenbefehl in der Greengrass-CLI.

Anforderung

Die Anforderung dieser Operation hat keine Parameter.

Antwort

Die Antwort dieser Operation enthält die folgenden Informationen:

`components`

Die Liste der Root-Komponenten auf dem Core-Gerät. Jedes Objekt in dieser Liste ist ein `ComponentDetails` Objekt, das die folgenden Informationen enthält:

`componentName` (Python: `component_name`)

Der Name der Komponente.

`version`

Die Version der Komponente.

`state`

Der Status der Komponente. Dieser Status kann einer der folgenden sein:

- `BROKEN`
- `ERRORED`
- `FINISHED`
- `INSTALLED`
- `NEW`
- `RUNNING`
- `STARTING`
- `STOPPING`

`configuration`

Die Konfiguration der Komponente als JSON-Objekt.

GetComponentDetails

Ruft die Version, den Status und die Konfiguration einer Komponente auf dem Core-Gerät ab.

Diese Operation bietet die gleiche Funktionalität wie der Komponentendetailbefehl in der Greengrass-CLI.

Anforderung

Die Anforderung dieser Operation hat die folgenden Parameter:

`componentName` (Python: `component_name`)

Der Name der Komponente, die abgerufen werden soll.

Antwort

Die Antwort dieser Operation enthält die folgenden Informationen:

`componentDetails` (Python: `component_details`)

Die Details der Komponente. Dieses Objekt, `ComponentDetails`, enthält die folgenden Informationen:

`componentName` (Python: `component_name`)

Der Name der Komponente.

`version`

Die Version der Komponente.

`state`

Der Status der Komponente. Dieser Status kann einer der folgenden sein:

- `BROKEN`
- `ERRORED`
- `FINISHED`
- `INSTALLED`
- `NEW`
- `RUNNING`
- `STARTING`
- `STOPPING`

`configuration`

Die Konfiguration der Komponente als JSON-Objekt.

RestartComponent

Startet eine Komponente auf dem Core-Gerät neu.

> **Note**
>
> Sie können zwar jede Komponente neu starten, wir empfehlen jedoch, nur generische Komponenten neu zu starten.

Dieser Vorgang bietet die gleiche Funktionalität wie der Befehl zum Neustart der Komponente in der Greengrass-CLI.

Anforderung

Die Anforderung dieser Operation hat die folgenden Parameter:

`componentName` (Python: `component_name`)

Der Name der Komponente.

Antwort

Die Antwort dieser Operation enthält die folgenden Informationen:

`restartStatus` (Python: `restart_status`)

Der Status der Neustartanforderung. Der Anforderungsstatus kann einer der folgenden sein:

- `SUCCEEDED`
- `FAILED`

`message`

Eine Meldung darüber, warum die Komponente nicht neu gestartet werden konnte, falls die Anforderung fehlgeschlagen ist.

StopComponent

Stoppt die Prozesse einer Komponente auf dem Core-Gerät.

> **Note**
>
> Sie können zwar jede Komponente anhalten, wir empfehlen jedoch, nur generische Komponenten anzuhalten.

Diese Operation bietet die gleiche Funktionalität wie der Befehl zum Stoppen von Komponenten in der Greengrass-CLI.

Anforderung

Die Anforderung dieser Operation hat die folgenden Parameter:

`componentName` (Python: `component_name`)

Der Name der Komponente.

Antwort

Die Antwort dieser Operation enthält die folgenden Informationen:

`stopStatus` (Python: `stop_status`)

Der Status der Stopp-Anforderung. Der Anforderungsstatus kann einer der folgenden sein:

- `SUCCEEDED`
- `FAILED`

`message`

Eine Meldung darüber, warum die Komponente nicht gestoppt werden konnte, falls die Anforderung fehlgeschlagen ist.

CreateDebugPassword

Generiert ein zufälliges Passwort, mit dem Sie sich bei der lokalen Debug-Konsolenkomponente anmelden können. Das Passwort läuft 8 Stunden nach seiner Generierung ab.

Diese Operation bietet die gleiche Funktionalität wie der get-debug-password Befehl in der Greengrass-CLI.

Anforderung

Die Anforderung dieser Operation hat keine Parameter.

Antwort

Die Antwort dieser Operation enthält die folgenden Informationen:

`username`

Der Benutzername, der für die Anmeldung verwendet werden soll.

`password`

Das Passwort, das für die Anmeldung verwendet werden soll.

`passwordExpiration` (Python: `password_expiration`)

Der Zeitpunkt, zu dem das Passwort abläuft.

`certificateSHA256Hash` (Python: `certificate_sha256_hash`)

Der SHA-256-Fingerabdruck für das selbstsignierte Zertifikat, das die lokale Debug-Konsole verwendet, wenn HTTPS aktiviert ist. Wenn Sie die lokale Debug-Konsole öffnen, verwenden Sie diesen Fingerabdruck, um zu überprüfen, ob das Zertifikat berechtigt und die Verbindung sicher ist.

`certificateSHA1Hash` (Python: `certificate_sha1_hash`)

Der SHA-1-Fingerabdruck für das selbstsignierte Zertifikat, das die lokale Debug-Konsole verwendet, wenn HTTPS aktiviert ist. Wenn Sie die lokale Debug-Konsole öffnen, verwenden Sie diesen Fingerabdruck, um zu überprüfen, ob das Zertifikat berechtigt und die Verbindung sicher ist.

Authentifizieren und Autorisieren von Client-Geräten

> **Note**
>
> Diese Funktion ist für v2.6.0 und höher der Greengrass-Kernkomponente verfügbar.

Verwenden Sie den Client-Geräte-Auth-IPK-Service, um eine benutzerdefinierte lokale Broker-Komponente zu entwickeln, mit der lokale IoT-Geräte, wie Client-Geräte, eine Verbindung herstellen können.

Um diese IPC-Operationen zu verwenden, fügen Sie Version 2.2.0 oder höher der Client-Geräteauthentifizierungskomponente als Abhängigkeit in Ihre benutzerdefinierte Komponente ein. Anschließend können Sie IPC-Operationen in Ihren benutzerdefinierten Komponenten verwenden, um Folgendes zu tun:

- Überprüfen Sie die Identität der Client-Geräte, die eine Verbindung zum Core-Gerät herstellen.
- Erstellen Sie eine Sitzung, damit ein Client-Gerät eine Verbindung zum Core-Gerät herstellen kann.

- Überprüfen Sie, ob ein Client-Gerät über die Berechtigung zum Ausführen einer Aktion verfügt.
- Erhalten Sie eine Benachrichtigung, wenn das Serverzertifikat des Core-Geräts rotiert.

Themen

SDK-Mindestversionen

In der folgenden Tabelle sind die Mindestversionen der aufgeführtAWS IoT Device SDK, die Sie für die Interaktion mit dem IPC-Service für die Authentifizierung des Client-Geräts verwenden müssen.

SDK	Mindestversion	
AWS IoT Device SDK für Java v2	v1.9.3	
AWS IoT Device SDK für Python v2	v1.11.3	
AWS IoT Device SDK für C++ v2	v1.18.3	
AWS IoT Device SDK für JavaScript v2	v1.12.0	

Autorisierung

Um den Client-Geräte-Authentifizierungs-IPK-Service in einer benutzerdefinierten Komponente zu verwenden, müssen Sie Autorisierungsrichtlinien definieren, die es Ihrer Komponente ermöglichen,

diese Vorgänge auszuführen. Informationen zum Definieren von Autorisierungsrichtlinien finden Sie unter Autorisieren Sie Komponenten zur Ausführung von IPC-Vorgängen.

Autorisierungsrichtlinien für die Authentifizierung und Autorisierung von Client-Geräten haben die folgenden Eigenschaften.

IPC-Service-ID: `aws.greengrass.clientdevices.Auth`

Operation	Beschreibung	Ressourcen
`aws.greengrass#VerifyClientDeviceIdentity`	Ermöglicht einer Komponente, die Identität eines Client-Geräts zu überprüfen.	*
`aws.greengrass#GetClientDeviceAuthToken`	Ermöglicht einer Komponente, die Anmeldeinformationen eines Client-Geräts zu validieren und eine Sitzung für dieses Client-Gerät zu erstellen.	*
`aws.greengrass#AuthorizeClientDeviceAction`	Ermöglicht einer Komponente zu überprüfen, ob ein Client-Gerät über die Berechtigung zum Ausführen einer Aktion verfügt.	*
`aws.greengrass#SubscribeToCertificateUpdates`	Ermöglicht einer Komponente, Benachrichtigungen zu erhalten, wenn das Serverzertifikat des Core-Geräts rotiert.	*
*	Ermöglicht einer Komponente, alle IPC-Service-Operationen zur Authentifizierung des Client-Geräts durchzuführen.	*

Beispiele für Autorisierungsrichtlinien

Sie können auf das folgende Beispiel für eine Autorisierungsrichtlinie verweisen, um Ihnen bei der Konfiguration von Autorisierungsrichtlinien für Ihre Komponenten zu helfen.

Example Beispiel für eine Autorisierungsrichtlinie

Die folgende Beispielautorisierungsrichtlinie ermöglicht es einer Komponente, alle IPC-Operationen zur Authentifizierung von Client-Geräten durchzuführen.

```
{
  "accessControl": {
    "aws.greengrass.clientdevices.Auth": {
      "com.example.MyLocalBrokerComponent:clientdevices:1": {
        "policyDescription": "Allows access to authenticate and authorize client
 devices.",
        "operations": [
          "aws.greengrass#VerifyClientDeviceIdentity",
          "aws.greengrass#GetClientDeviceAuthToken",
          "aws.greengrass#AuthorizeClientDeviceAction",
          "aws.greengrass#SubscribeToCertificateUpdates"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

VerifyClientDeviceIdentity

Überprüfen Sie die Identität eines Client-Geräts. Diese Operation überprüft, ob das Client-Gerät ein gültiges AWS IoT Objekt ist.

Anforderung

Die Anforderung dieser Operation hat die folgenden Parameter:

`credential`

Die Anmeldeinformationen des Client-Geräts. Dieses Objekt, `ClientDeviceCredential`, enthält die folgenden Informationen:

`clientDeviceCertificate` (Python: `client_device_certificate`)

Das X.509-Gerätezertifikat des Client-Geräts.

Antwort

Die Antwort dieser Operation enthält die folgenden Informationen:

`isValidClientDevice` (Python: `is_valid_client_device`)

Gibt an, ob die Identität des Client-Geräts gültig ist.

GetClientDeviceAuthToken

Validiert die Anmeldeinformationen eines Client-Geräts und erstellt eine Sitzung für das Client-Gerät. Dieser Vorgang gibt ein Sitzungstoken zurück, das Sie in nachfolgenden Anforderungen verwenden können, um Client-Geräteaktionen zu autorisieren.

Um ein Client-Gerät erfolgreich zu verbinden, muss die Authentifizierungskomponente des Client-Geräts die -`mqtt:connect`Berechtigung für die Client-ID erteilen, die das Client-Gerät verwendet.

Anforderung

Die Anforderung dieser Operation hat die folgenden Parameter:

`credential`

Die Anmeldeinformationen des Client-Geräts. Dieses Objekt, `CredentialDocument`, enthält die folgenden Informationen:

`mqttCredential` (Python: `mqtt_credential`)

Die MQTT-Anmeldeinformationen des Client-Geräts. Geben Sie die Client-ID und das Zertifikat an, die das Client-Gerät für die Verbindung verwendet. Dieses Objekt, `MQTTCredential`, enthält die folgenden Informationen:

`clientId` (Python: `client_id`)

Die Client-ID, die für die Verbindung verwendet werden soll.

`certificatePem` (Python: `certificate_pem`)

Das X.509-Gerätezertifikat, das für die Verbindung verwendet werden soll.

`username`

> **Note**
>
> Diese Eigenschaft wird derzeit nicht verwendet.

`password`

> **Note**
>
> Diese Eigenschaft wird derzeit nicht verwendet.

Antwort

Die Antwort dieser Operation enthält die folgenden Informationen:

`clientDeviceAuthToken` (Python: `client_device_auth_token`)

Das Sitzungstoken für das Client-Gerät. Sie können dieses Sitzungstoken in nachfolgenden Anforderungen verwenden, um die Aktionen dieses Client-Geräts zu autorisieren.

AuthorizeClientDeviceAction

Überprüfen Sie, ob ein Client-Gerät berechtigt ist, eine Aktion für eine Ressource auszuführen. Autorisierungsrichtlinien für Client-Geräte geben die Berechtigungen an, die Client-Geräte ausführen können, während sie mit einem Core-Gerät verbunden sind. Sie definieren Autorisierungsrichtlinien für Client-Geräte, wenn Sie die Authentifizierungskomponente des Client-Geräts konfigurieren.

Anforderung

Die Anforderung dieser Operation hat die folgenden Parameter:

`clientDeviceAuthToken` (Python: `client_device_auth_token`)

Das Sitzungstoken für das Client-Gerät.

`operation`

Der zu autorisierende Vorgang.

`resource`

Die Ressource, mit der das Client-Gerät den Vorgang ausführt.

Antwort

Die Antwort dieser Operation enthält die folgenden Informationen:

`isAuthorized` (Python: `is_authorized`)

Gibt an, ob das Client-Gerät berechtigt ist, den Vorgang für die Ressource auszuführen.

SubscribeToCertificateUpdates

Abonnieren Sie bei jeder Rotation, um das neue Serverzertifikat des Core-Geräts zu erhalten. Wenn das Serverzertifikat rotiert, müssen Broker mit dem neuen Serverzertifikat neu laden.

Die Authentifizierungskomponente des Client-Geräts rotiert Serverzertifikate standardmäßig alle 7 Tage. Sie können das Drehungsintervall zwischen 2 und 10 Tagen konfigurieren.

Bei diesem Vorgang handelt es sich um einen Abonnementvorgang, bei dem Sie einen Stream von Ereignisnachrichten abonnieren. Um diesen Vorgang zu verwenden, definieren Sie einen Stream-Antwort-Handler mit Funktionen, die Ereignismeldungen, Fehler und Stream-Schließung verarbeiten. Weitere Informationen finden Sie unter Abonnieren Sie IPC-Event-Streams.

Typ der Ereignisnachricht: `CertificateUpdateEvent`

Anforderung

Die Anforderung dieser Operation hat die folgenden Parameter:

`certificateOptions` (Python: `certificate_options`)

Die Arten von Zertifikatsaktualisierungen, die abonniert werden sollen. Dieses Objekt, `CertificateOptions`, enthält die folgenden Informationen:

`certificateType` (Python: `certificate_type`)

Der Typ der Zertifikatsaktualisierungen, die abonniert werden sollen. Wählen Sie die folgende Option aus:

- `SERVER`

Antwort

Die Antwort dieser Operation enthält die folgenden Informationen:

`messages`

Der Stream von Nachrichten. Dieses Objekt, `CertificateUpdateEvent`, enthält die folgenden Informationen:

`certificateUpdate` (Python: `certificate_update`)

Die Informationen über das neue Zertifikat. Dieses Objekt, `CertificateUpdate`, enthält die folgenden Informationen:

`certificate`

Das Zertifikat.

`privateKey` (Python: `private_key`)

Der private Schlüssel des Zertifikats.

`publicKey` (Python: `public_key`)

Der öffentliche Schlüssel des Zertifikats.

`caCertificates` (Python: `ca_certificates`)

Die Liste der Zertifizierungsstellenzertifikate (CA) in der CA-Zertifikatkette des Zertifikats.

Interagieren mit lokalen IoT-Geräten

Client-Geräte sind lokale IoT-Geräte, die über MQTT eine Verbindung zu einem Greengrass-Core-Gerät herstellen und mit diesem kommunizieren. Sie können Client-Geräte mit -Core-Geräten verbinden, um Folgendes zu tun:

- Interagieren Sie mit MQTT-Nachrichten in Greengrass-Komponenten.
- Weiterleiten von Nachrichten und Daten zwischen Client-Geräten und AWS IoT Core.
- Interagieren Sie mit Client-Geräteschatten in Greengrass-Komponenten.
- Synchronisieren Sie Schatten von Client-Geräten mit AWS IoT Core.

Um eine Verbindung zu einem Core-Gerät herzustellen, können Client-Geräte Cloud Discovery verwenden. Client-Geräte stellen eine Verbindung mit dem AWS IoT Greengrass Cloud-Service her, um Informationen über -Core-Geräte abzurufen, mit denen sie eine Verbindung herstellen können. Anschließend können sie eine Verbindung zu einem Core-Gerät herstellen, um ihre Nachrichten zu verarbeiten und ihre Daten mit dem AWS IoT Core Cloud-Service zu synchronisieren.

Sie können einem Tutorial folgen, in dem beschrieben wird, wie Sie ein Core-Gerät für die Verbindung und Kommunikation mit einem -AWS IoTObjekt konfigurieren. In diesem Tutorial wird auch erläutert, wie Sie eine benutzerdefinierte Greengrass-Komponente entwickeln, die mit Client-Geräten interagiert. Weitere Informationen finden Sie unter Tutorial: Interagieren mit lokalen IoT-Geräten über MQTT.

Themen

AWSVon bereitgestellte Client-Gerätekomponenten

AWS IoT Greengrass stellt die folgenden öffentlichen Komponenten bereit, die Sie auf -Core-Geräten bereitstellen können. Diese Komponenten ermöglichen es Client-Geräten, eine Verbindung herzustellen und mit einem Core-Gerät zu kommunizieren.

> **Note**
>
> Mehrere AWSvon bereitgestellte Komponenten hängen von bestimmten Nebenversionen des Greengrass-Kerns ab. Aufgrund dieser Abhängigkeit müssen Sie diese Komponenten aktualisieren, wenn Sie den Greengrass-Kern auf eine neue Nebenversion aktualisieren. Informationen zu den spezifischen Versionen des Kerns, von denen jede Komponente abhängt, finden Sie im entsprechenden Komponententhema. Weitere Informationen zum Aktualisieren des Kerns finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Wenn eine Komponente sowohl den Komponententyp generisch als auch Lambda hat, ist die aktuelle Version der Komponente der generische Typ und eine frühere Version der Komponente der Lambda-Typ.

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
Authentifizierung auf Client-Geräten	Ermöglicht lokalen IoT-Geräten, so genannte Client-Geräte, eine Verbindung mit dem Core-Gerät herzustellen.	Plug-In	Linux, Windows	Ja

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
IP-Detektor	Meldet MQTT-Broker-Konnektivitätsinformationen an AWS IoT Greengrass, sodass Client-Geräte erkennen können, wie eine Verbindung hergestellt werden soll.	Plug-In	Linux, Windows	Ja
MQTT-Brücke	Weiterleitet MQTT-Nachrichten zwischen Client-Geräten, lokaler AWS IoT Greengrass sVeröffentlichung/Abonnement und AWS IoT Core.	Plug-In	Linux, Windows	Ja

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
MQTT 3.1.1-Broker (Moquette)	Führt einen MQTT-3.1.1-Broker aus, der Nachrichten zwischen Client-Geräten und dem Core-Gerät verarbeitet.	Plug-In	Linux, Windows	Ja
MQTT 5-Broker (EMAX)	Führt einen MQTT 5-Broker aus, der Nachrichten zwischen Client-Geräten und dem Core-Gerät verarbeitet.	Generisch	Linux, Windows	Nein

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
Schattenmanager	Ermöglicht die Interaktion mit Schatten auf dem Core-Gerät. Es verwaltet die Speicherung von Schattendokumenten sowie die Synchronisation lokaler Schattenzustände mit dem AWS IoT Gerätesch attendienst.	Plug-In	Linux, Windows	Ja

Verbinden von Client-Geräten mit -Core-Geräten

Sie können die Cloud-Erkennung konfigurieren, um Client-Geräte mit -Core-Geräten zu verbinden. Wenn Sie die Cloud-Erkennung konfigurieren, können Client-Geräte eine Verbindung mit dem AWS IoT Greengrass Cloud-Service herstellen, um Informationen über -Core-Geräte abzurufen, mit denen sie eine Verbindung herstellen können. Anschließend können die Client-Geräte versuchen, eine Verbindung zu jedem Core-Gerät herzustellen, bis sie erfolgreich eine Verbindung herstellen.

Um die Cloud-Erkennung zu verwenden, müssen Sie Folgendes tun:

- Ordnen Sie Client-Geräte den -Core-Geräten zu, mit denen sie eine Verbindung herstellen können.
- Geben Sie die MQTT-Broker-Endpunkte an, an denen Client-Geräte eine Verbindung zu jedem Core-Gerät herstellen können.

- Stellen Sie Komponenten auf dem Core-Gerät bereit, die Unterstützung für Client-Geräte ermöglichen.

  Sie können auch optionale Komponenten bereitstellen, um Folgendes zu tun:

  - Weiterleiten von Nachrichten zwischen Client-Geräten, Greengrass-Komponenten und dem AWS IoT Core Cloud-Service.
  - Verwalten Sie die MQTT-Broker-Endpunkte des Core-Geräts automatisch für Sie.
  - Verwalten Sie lokale Client-Geräteschatten und synchronisieren Sie Schatten mit dem AWS IoT Core Cloud-Service.

Sie müssen auch die AWS IoT Richtlinie des Core-Geräts überprüfen und aktualisieren, um sicherzustellen, dass es über die erforderlichen Berechtigungen verfügt, um Client-Geräte zu verbinden. Weitere Informationen finden Sie unter Voraussetzungen.

Nachdem Sie die Cloud-Erkennung konfiguriert haben, können Sie die Kommunikation zwischen einem Client-Gerät und einem Core-Gerät testen. Weitere Informationen finden Sie unter Testen der Kommunikation von Client-Geräten.

Themen

Voraussetzungen

Um Client-Geräte mit einem Core-Gerät zu verbinden, benötigen Sie Folgendes:

- Das Core-Gerät muss Greengrass-Kern v2.2.0 oder höher ausführen.
- Die Greengrass-Servicerolle, die Ihrem AWS IoT Greengrass in der AWS Region zugeordnet ist, AWS-Konto in der das Core-Gerät ausgeführt wird. Weitere Informationen finden Sie unter Konfigurieren der Greengrass-Servicerolle.
- Die AWS IoT Richtlinie des Core-Geräts muss die folgenden Berechtigungen zulassen:
  - `greengrass:PutCertificateAuthorities`
  - `greengrass:VerifyClientDeviceIdentity`
  - `greengrass:VerifyClientDeviceIoTCertificateAssociation`
  - `greengrass:GetConnectivityInfo`
  - `greengrass:UpdateConnectivityInfo` – (Optional) Diese Berechtigung ist erforderlich, um die IP-Detektorkomponente zu verwenden, die die Netzwerkkonnektivitätsinformationen des Core-Geräts an den AWS IoT Greengrass Cloud-Service meldet.
  - `iot:GetThingShadow`, und `iot:DeleteThingShadow` – (Optional) Diese Berechtigungen sind erforderlich`iot:UpdateThingShadow`, um die Shadow Manager-Komponente zum Synchronisieren von Client-Geräteschatten mit zu verwendenAWS IoT Core. Für diese Funktion sind Greengrass-Kern v2.6.0 oder höher, Shadow Manager v2.2.0 oder höher und MQTT Bridge v2.2.0 oder höher erforderlich.

  Weitere Informationen finden Sie unter Konfigurieren der AWS IoT Objektrichtlinie.

  > **Note**
  >
  > Wenn Sie bei der Installation der Core-Software die AWS IoT Standardrichtlinie verwendet haben, verfügt das Core-Gerät über eine -AWS IoTRichtlinie, die den Zugriff auf alle AWS IoT Greengrass Aktionen (`greengrass:*`) ermöglicht. AWS IoT Greengrass
- AWS IoT -Objekte, die Sie als Client-Geräte verbinden können. Weitere Informationen finden Sie unter Erstellen von -AWS IoTRessourcen im AWS IoT Core -Entwicklerhandbuch.
- Das Client-Gerät muss eine Verbindung mit einer Client-ID herstellen. Eine Client-ID ist ein Objektname. Es wird keine andere Client-ID akzeptiert.
- Die AWS IoT Richtlinie jedes Client-Geräts muss die -`greengrass:Discover`Berechtigung zulassen. Weitere Informationen finden Sie unter Minimale AWS IoT Richtlinie für Client-Geräte.

Themen

- Konfigurieren der Greengrass-Servicerolle

- Konfigurieren der AWS IoT Objektrichtlinie

Konfigurieren der Greengrass-Servicerolle

Die Greengrass-Servicerolle ist eine AWS Identity and Access Management (IAM)-Servicerolle, die autorisiertAWS IoT Greengrass, in Ihrem Namen auf Ressourcen von -AWSServices zuzugreifen. Diese Rolle ermöglicht es dem , die Identität von Client-Geräten AWS IoT Greengrass zu überprüfen und die Konnektivitätsinformationen des Core-Geräts zu verwalten.

Wenn Sie die Greengrass-Servicerolle noch nicht in dieser Region eingerichtet haben, müssen Sie eine Greengrass-Servicerolle mit AWS IoT Greengrass für Ihr AWS-Konto in dieser Region verknüpfen.

Wenn Sie die Seite Configure core device discovery in der AWS IoT Greengrass Konsole verwenden, AWS IoT Greengrass richtet die Greengrass-Servicerolle für Sie ein. Andernfalls können Sie sie manuell über die AWS IoTKonsole oder AWS IoT Greengrass API einrichten.

In diesem Abschnitt überprüfen Sie, ob die Greengrass-Servicerolle eingerichtet ist. Wenn es nicht eingerichtet ist, erstellen Sie eine neue Greengrass-Servicerolle, die Sie AWS IoT Greengrass Ihrem AWS-Konto in dieser Region zuordnen können.

Konfigurieren der Greengrass-Servicerolle (Konsole)

1. Überprüfen Sie, ob die Greengrass-Servicerolle AWS IoT Greengrass für Ihr AWS-Konto in dieser Region zugeordnet ist. Gehen Sie wie folgt vor:

   a. Navigieren Sie zur AWS IoT-Konsole.
   b. Wählen Sie im Navigationsbereich Settings (Einstellungen).
   c. Suchen Sie im Abschnitt Greengrass-Servicerolle nach Aktuelle Servicerolle, um zu sehen, ob eine Greengrass-Servicerolle zugeordnet ist.

      Wenn Ihnen eine Greengrass-Servicerolle zugeordnet ist, erfüllen Sie diese Anforderung für die Verwendung der IP-Detektorkomponente. Fahren Sie mit Konfigurieren der AWS IoT Objektrichtlinie fort.
2. Wenn die Greengrass-Servicerolle nicht mit AWS IoT Greengrass für Ihr AWS-Konto in dieser Region verknüpft ist, erstellen Sie eine Greengrass-Servicerolle und verknüpfen Sie sie. Gehen Sie wie folgt vor:

   a. Navigieren Sie zur IAM-Konsole.

b. Wählen Sie Roles.

c. Wählen Sie Rolle erstellen aus.

d. Gehen Sie auf der Seite Rolle erstellen wie folgt vor:

   i. Wählen Sie unter Vertrauenswürdiger Entitätstyp ausAWS-Service.

   ii. Wählen Sie unter Anwendungsfall , Anwendungsfälle für andere die AWS-ServicesOption Greengrass und dann Greengrass aus. Diese Option gibt an, dass AWS IoT Greengrass als vertrauenswürdige Entität hinzugefügt werden soll, die diese Rolle übernehmen kann.

   iii. Wählen Sie Weiter aus.

   iv. Wählen Sie unter Berechtigungsrichtlinien die aus, AWSGreengrassResourceAccessRolePolicy die an die Rolle angehängt werden soll.

   v. Wählen Sie Weiter aus.

   vi. Geben Sie unter Rollenname einen Namen für die Rolle ein, z. B. **`Greengrass_ServiceRole`**.

   vii. Wählen Sie Rolle erstellen aus.

e. Navigieren Sie zur AWS IoT-Konsole.

f. Wählen Sie im Navigationsbereich Settings (Einstellungen).

g. Wählen Sie im Abschnitt Greengrass-Servicerolle die Option Rolle anfügen aus.

h. Wählen Sie im Modal Greengrass-Servicerolle aktualisieren die von Ihnen erstellte IAM-Rolle aus und wählen Sie dann Rolle anfügen aus.

Konfigurieren der Greengrass-Servicerolle (AWS CLI)

1. Überprüfen Sie, ob die Greengrass-Servicerolle AWS IoT Greengrass für Ihr AWS-Konto in dieser Region zugeordnet ist.

   ```
   aws greengrassv2 get-service-role-for-account
   ```

   Wenn die Greengrass-Servicerolle zugeordnet ist, gibt die Operation eine Antwort zurück, die Informationen über die Rolle enthält.

   Wenn Ihnen eine Greengrass-Servicerolle zugeordnet ist, erfüllen Sie diese Anforderung für die Verwendung der IP-Detektorkomponente. Fahren Sie mit Konfigurieren der AWS IoT Objektrichtlinie fort.

2. Wenn die Greengrass-Servicerolle nicht mit AWS IoT Greengrass für Ihr AWS-Konto in dieser Region verknüpft ist, erstellen Sie eine Greengrass-Servicerolle und verknüpfen Sie sie. Gehen Sie wie folgt vor:

   a. Erstellen Sie die Rolle mit einer Vertrauensrichtlinie, die AWS IoT Greengrass die Annahme der Rolle erlaubt. In diesem Beispiel wird eine Rolle namens `Greengrass_ServiceRole` erstellt, aber Sie können einen anderen Namen verwenden. Wir empfehlen Ihnen, auch die `aws:SourceAccount` globalen Bedingungskontextschlüssel `aws:SourceArn` und in Ihre Vertrauensrichtlinie aufzunehmen, um das Sicherheitsproblem des verwirrten Stellvertreters zu vermeiden. Die Bedingungskontextschlüssel schränken den Zugriff so ein, dass nur die Anforderungen zugelassen werden, die aus dem angegebenen Konto und dem Greengrass-Workspace stammen. Weitere Informationen zum Confused-Deputy-Problem finden Sie Vermeidung des Problems des verwirrten Stellvertreters (dienstübergreifend).

      Linux or Unix

      ```
      aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-
      document '{
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Principal": {
              "Service": "greengrass.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
              "ArnLike": {
                "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
              },
              "StringEquals": {
                "aws:SourceAccount": "account-id"
              }
            }
          }
        ]
      }'
      ```

Windows Command Prompt (CMD)

```
aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-
document "{\\"Version\\":\\"2012-10-17\\",\\"Statement\\":[{\\"Effect\
\":\\"Allow\\",\\"Principal\\":{\\"Service\\":\\"greengrass.amazonaws.com
\\"},\\"Action\\":\\"sts:AssumeRole\\",\\"Condition\\":{\\"ArnLike\\":
{\\"aws:SourceArn\\":\\"arn:aws:greengrass:region:account-id:*\\"},\
\"StringEquals\\":{\\"aws:SourceAccount\\":\\"account-id\\"}}}]}"
```

PowerShell

```
aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-
document '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "greengrass.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
        },
        "StringEquals": {
          "aws:SourceAccount": "account-id"
        }
      }
    }
  ]
}'
```

b. Kopieren Sie den Rollen-ARN aus den Rollenmetadaten in der Ausgabe. Sie verknüpfen die Servicerolle mithilfe des ARN mit Ihrem Konto.

c. 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
```

d. Ordnen Sie die Greengrass-Servicerolle AWS IoT Greengrass Ihrem zuAWS-Konto. Ersetzen Sie *role-arn* durch den ARN der Servicerolle.

```
aws greengrassv2 associate-service-role-to-account --role-arn role-arn
```

Die Operation gibt die folgende Antwort zurück, wenn sie erfolgreich ist.

```
{
  "associatedAt": "timestamp"
}
```

Konfigurieren der AWS IoT Objektrichtlinie

Core-Geräte verwenden X.509-Gerätezertifikate, um Verbindungen zu zu autorisierenAWS. Sie fügen AWS IoT Richtlinien an Gerätezertifikate an, um die Berechtigungen für ein Core-Gerät zu definieren. Weitere Informationen finden Sie unter AWS IoT-Richtlinien für Operationen auf Datenebene und Minimale AWS IoT Richtlinie zur Unterstützung von Client-Geräten.

Um Client-Geräte mit einem Core-Gerät zu verbinden, muss die AWS IoT Richtlinie des Core-Geräts die folgenden Berechtigungen zulassen:

- `greengrass:PutCertificateAuthorities`
- `greengrass:VerifyClientDeviceIdentity`
- `greengrass:VerifyClientDeviceIoTCertificateAssociation`
- `greengrass:GetConnectivityInfo`
- `greengrass:UpdateConnectivityInfo` – (Optional) Diese Berechtigung ist erforderlich, um die IP-Detektorkomponente zu verwenden, die die Netzwerkkonnektivitätsinformationen des Core-Geräts an den AWS IoT Greengrass Cloud-Service meldet.
- `iot:GetThingShadow`, und `iot:DeleteThingShadow` – (Optional) Diese Berechtigungen sind erforderlich`iot:UpdateThingShadow`, um die Shadow Manager-Komponente zum Synchronisieren von Client-Geräteschatten mit zu verwendenAWS IoT Core. Für diese Funktion sind Greengrass-Kern v2.6.0 oder höher, Shadow Manager v2.2.0 oder höher und MQTT Bridge v2.2.0 oder höher erforderlich.

In diesem Abschnitt überprüfen Sie die AWS IoT Richtlinien für Ihr Core-Gerät und fügen alle erforderlichen Berechtigungen hinzu, die fehlen. Wenn Sie das AWS IoT Greengrass -Core-

Softwareinstallationsprogramm zur Bereitstellung von Ressourcen verwendet haben, verfügt Ihr -Core-Gerät über eine -AWS IoTRichtlinie, die den Zugriff auf alle -AWS IoT GreengrassAktionen ermöglicht (`greengrass:*`). In diesem Fall müssen Sie die AWS IoT Richtlinie nur aktualisieren, wenn Sie die Shadow Manager-Komponente bereitstellen möchten, um Geräteschatten mit zu synchronisierenAWS IoT Core. Andernfalls können Sie diesen Abschnitt überspringen.

Konfigurieren der AWS IoT Objektrichtlinie (Konsole)

1. Wählen Sie im Navigationsmenü der AWS IoT Greengrass Konsole die Option Core-Geräte aus.
2. Wählen Sie auf der Seite Core-Geräte das zu aktualisierende Core-Gerät aus.
3. Wählen Sie auf der Detailseite des Core-Geräts den Link zum Objekt des Core-Geräts aus. Dieser Link öffnet die Seite mit den Objektdetails in der -AWS IoTKonsole.
4. Wählen Sie auf der Seite mit den Objektdetails die Option Zertifikate aus.
5. Wählen Sie auf der Registerkarte Zertifikate das aktive Zertifikat des Objekts aus.
6. Wählen Sie auf der Seite mit den Zertifikatsdetails Richtlinien aus.
7. Wählen Sie auf der Registerkarte Richtlinien die zu überprüfende und zu aktualisierende AWS IoT Richtlinie aus. Sie können die erforderlichen Berechtigungen zu jeder Richtlinie hinzufügen, die dem aktiven Zertifikat des Core-Geräts angefügt ist.

   > **Note**
   >
   > Wenn Sie das AWS IoT Greengrass-Core-Softwareinstallationsprogramm zur Bereitstellung von Ressourcen verwendet haben, haben Sie zwei AWS IoT Richtlinien. Wir empfehlen Ihnen, die Richtlinie mit dem Namen auszuwählenGreengrassV2IoTThingPolicy, falls vorhanden. Core-Geräte, die Sie mit dem Schnellinstallationsprogramm erstellen, verwenden diesen Richtliniennamen standardmäßig. Wenn Sie dieser Richtlinie Berechtigungen hinzufügen, erteilen Sie diese Berechtigungen auch anderen -Core-Geräten, die diese Richtlinie verwenden.

8. Wählen Sie in der Richtlinienübersicht die Option Aktive Version bearbeiten aus.
9. Überprüfen Sie die Richtlinie auf die erforderlichen Berechtigungen und fügen Sie alle erforderlichen Berechtigungen hinzu, die fehlen.

   - `greengrass:PutCertificateAuthorities`
   - `greengrass:VerifyClientDeviceIdentity`
   - `greengrass:VerifyClientDeviceIoTCertificateAssociation`

- `greengrass:GetConnectivityInfo`
- `greengrass:UpdateConnectivityInfo` – (Optional) Diese Berechtigung ist erforderlich, um die IP-Detektorkomponente zu verwenden, die die Netzwerkkonnektivitätsinformationen des Core-Geräts an den AWS IoT Greengrass Cloud-Service meldet.
- `iot:GetThingShadow`, und `iot:DeleteThingShadow` – (Optional) Diese Berechtigungen sind erforderlich`iot:UpdateThingShadow`, um die Shadow Manager-Komponente zum Synchronisieren von Client-Geräteschatten mit zu verwendenAWS IoT Core. Für diese Funktion sind Greengrass-Kern v2.6.0 oder höher, Shadow Manager v2.2.0 oder höher und MQTT Bridge v2.2.0 oder höher erforderlich.

10. (Optional) Damit das Core-Gerät Schatten mit synchronisieren kannAWS IoT Core, fügen Sie der Richtlinie die folgende Anweisung hinzu. Wenn Sie vorhaben, mit Client-Geräteschatten zu interagieren, sie aber nicht mit synchronisierenAWS IoT Core, überspringen Sie diesen Schritt. Ersetzen Sie *`region`* und *`account-id`* durch die von Ihnen verwendete Region und Ihre -AWS-KontoNummer.

    - Diese Beispielanweisung ermöglicht den Zugriff auf die Geräteschatten aller Objekte. Um bewährte Sicherheitsmethoden zu befolgen, können Sie den Zugriff nur auf das Core-Gerät und die Client-Geräte beschränken, die Sie mit dem Core-Gerät verbinden. Weitere Informationen finden Sie unter Minimale AWS IoT Richtlinie zur Unterstützung von Client-Geräten.

    ```
    {
      "Effect": "Allow",
      "Action": [
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:DeleteThingShadow"
      ],
      "Resource": [
        "arn:aws:iot:region:account-id:thing/*"
      ]
    }
    ```

    Nachdem Sie diese Anweisung hinzugefügt haben, könnte das Richtliniendokument dem folgenden Beispiel ähneln.

    ```
    {
    ```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect",
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive",
        "greengrass:*"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:DeleteThingShadow"
      ],
      "Resource": [
        "arn:aws:iot:region:account-id:thing/*"
      ]
    }
  ]
}
```

11. 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.
12. Wählen Sie Als neue Version speichern aus.

Konfigurieren der AWS IoT Objektrichtlinie (AWS CLI)

1. Listen Sie die Prinzipale für das AWS IoT Objekt des Core-Geräts auf. Objektprinzipale können X.509-Gerätezertifikate oder andere Identifikatoren sein. Führen Sie den folgenden Befehl aus und ersetzen Sie *MyGreengrassCore* durch den Namen des Core-Geräts.

   ```
   aws iot list-thing-principals --thing-name MyGreengrassCore
   ```

   Der Vorgang gibt eine Antwort zurück, die die Objektprinzipale des Core-Geräts auflistet.

```
{
    "principals": [
        "arn:aws:iot:us-west-2:123456789012:cert/certificateId"
    ]
}
```

2. Identifizieren Sie das aktive Zertifikat des Core-Geräts. Führen Sie den folgenden Befehl aus und ersetzen Sie *certificateId* durch die ID jedes Zertifikats aus dem vorherigen Schritt, bis Sie das aktive Zertifikat finden. Die Zertifikat-ID ist die hexadezimale Zeichenfolge am Ende des Zertifikat-ARN. Das `--query` Argument gibt an, dass nur der Status des Zertifikats ausgegeben wird.

   ```
   aws iot describe-certificate --certificate-id certificateId --query
    'certificateDescription.status'
   ```

   Der Vorgang gibt den Zertifikatsstatus als Zeichenfolge zurück. Wenn das Zertifikat beispielsweise aktiv ist, gibt dieser Vorgang aus`"ACTIVE"`.

3. Listen Sie die AWS IoT Richtlinien auf, die dem Zertifikat angefügt sind. Führen Sie den folgenden Befehl aus und ersetzen Sie den Zertifikat-ARN durch den ARN des Zertifikats.

   ```
   aws iot list-principal-policies --principal arn:aws:iot:us-
   west-2:123456789012:cert/certificateId
   ```

   Der Vorgang gibt eine Antwort zurück, in der die AWS IoT Richtlinien aufgeführt sind, die dem Zertifikat angefügt sind.

   ```
   {
       "policies": [
           {
               "policyName":
    "GreengrassTESCertificatePolicyMyGreengrassCoreTokenExchangeRoleAlias",
               "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
   GreengrassTESCertificatePolicyMyGreengrassCoreTokenExchangeRoleAlias"
           },
           {
               "policyName": "GreengrassV2IoTThingPolicy",
               "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
   GreengrassV2IoTThingPolicy"
           }
   ```

```
    ]
}
```

4. Wählen Sie die anzuzeigende und zu aktualisierende Richtlinie aus.

> **Note**
>
> Wenn Sie das AWS IoT Greengrass-Core-Softwareinstallationsprogramm zur Bereitstellung von Ressourcen verwendet haben, haben Sie zwei AWS IoT Richtlinien. Wir empfehlen Ihnen, die Richtlinie mit dem Namen auszuwählenGreengrassV2IoTThingPolicy, falls sie vorhanden ist. Core-Geräte, die Sie mit dem Schnellinstallationsprogramm erstellen, verwenden diesen Richtliniennamen standardmäßig. Wenn Sie dieser Richtlinie Berechtigungen hinzufügen, erteilen Sie diese Berechtigungen auch anderen -Core-Geräten, die diese Richtlinie verwenden.

5. Rufen Sie das Dokument der Richtlinie ab. Führen Sie den folgenden Befehl aus und ersetzen Sie *GreengrassV2IoTThingPolicy* durch den Namen der Richtlinie.

```
aws iot get-policy --policy-name GreengrassV2IoTThingPolicy
```

Die Operation gibt eine Antwort zurück, die das Dokument der Richtlinie und andere Informationen über die Richtlinie enthält. Das Richtliniendokument ist ein JSON-Objekt, das als Zeichenfolge serialisiert wird.

```
{
    "policyName": "GreengrassV2IoTThingPolicy",
    "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
    "policyDocument": "{\
  \\"Version\\": \\"2012-10-17\\",\
  \\"Statement\\": [\
    {\
      \\"Effect\\": \\"Allow\\",\
      \\"Action\\": [\
                \\"iot:Connect\\",\
                \\"iot:Publish\\",\
                \\"iot:Subscribe\\",\
                \\"iot:Receive\\",\
                \\"greengrass:*\\"\
],\
```

```
      \\"Resource\\": \\"*\\"\
    }\
  ]\
}",
    "defaultVersionId": "1",
    "creationDate": "2021-02-05T16:03:14.098000-08:00",
    "lastModifiedDate": "2021-02-05T16:03:14.098000-08:00",
    "generationId":
 "f19144b798534f52c619d44f771a354f1b957dfa2b850625d9f1d0fde530e75f"
}
```

6. Verwenden Sie einen Online-Konverter oder ein anderes Tool, um die Richtliniendokumentzeichenfolge in ein JSON-Objekt zu konvertieren, und speichern Sie sie dann in einer Datei mit dem Namen `iot-policy.json`.

   Wenn Sie beispielsweise das Tool jq installiert haben, können Sie den folgenden Befehl ausführen, um das Richtliniendokument abzurufen, es in ein JSON-Objekt zu konvertieren und das Richtliniendokument als JSON-Objekt zu speichern.

   ```
   aws iot get-policy --policy-name GreengrassV2IoTThingPolicy --query
    'policyDocument' | jq fromjson >> iot-policy.json
   ```

7. Überprüfen Sie die Richtlinie auf die erforderlichen Berechtigungen und fügen Sie alle erforderlichen Berechtigungen hinzu, die fehlen.

   Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Öffnen der Datei zu verwenden.

   ```
   nano iot-policy.json
   ```

   - `greengrass:PutCertificateAuthorities`
   - `greengrass:VerifyClientDeviceIdentity`
   - `greengrass:VerifyClientDeviceIoTCertificateAssociation`
   - `greengrass:GetConnectivityInfo`
   - `greengrass:UpdateConnectivityInfo` – (Optional) Diese Berechtigung ist erforderlich, um die IP-Detektorkomponente zu verwenden, die die Netzwerkkonnektivitätsinformationen des Core-Geräts an den AWS IoT Greengrass Cloud-Service meldet.
   - `iot:GetThingShadow`, und `iot:DeleteThingShadow` – (Optional) Diese Berechtigungen sind erforderlich`iot:UpdateThingShadow`, um die Shadow Manager-Komponente zum

Synchronisieren von Client-Geräteschatten mit zu verwendenAWS IoT Core. Für diese Funktion sind Greengrass-Kern v2.6.0 oder höher, Shadow Manager v2.2.0 oder höher und MQTT Bridge v2.2.0 oder höher erforderlich.

8. Speichern Sie die Änderungen als neue Version der Richtlinie. Führen Sie den folgenden Befehl aus und ersetzen Sie *GreengrassV2IoTThingPolicy* durch den Namen der Richtlinie.

```
aws iot create-policy-version --policy-name GreengrassV2IoTThingPolicy --policy-
document file://iot-policy.json --set-as-default
```

Die Operation gibt eine Antwort ähnlich dem folgenden Beispiel zurück, wenn sie erfolgreich ist.

```
{
    "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
    "policyDocument": "{\
  \\"Version\\": \\"2012-10-17\\",\
  \\"Statement\\": [\
    {\
      \\"Effect\\": \\"Allow\\",\
      \\"Action\\": [\
\\t\\t\\"iot:Connect\\",\
\\t\\t\\"iot:Publish\\",\
\\t\\t\\"iot:Subscribe\\",\
\\t\\t\\"iot:Receive\\",\
\\t\\t\\"greengrass:*\\"\
      ],\
      \\"Resource\\": \\"*\\"\
    }\
  ]\
}",
    "policyVersionId": "2",
    "isDefaultVersion": true
}
```

Greengrass-Komponenten für die Unterstützung von Client-Geräten

> **Important**
>
> Das Core-Gerät muss Greengrass-Kern v2.2.0 oder höher ausführen, um Client-Geräte zu unterstützen.

Damit Client-Geräte eine Verbindung zu einem Core-Gerät herstellen und mit diesem kommunizieren können, stellen Sie die folgenden Greengrass-Komponenten auf dem Core-Gerät bereit:

- Authentifizierung auf Client-Geräten (`aws.greengrass.clientdevices.Auth`)

  Stellen Sie die Authentifizierungskomponente des Client-Geräts bereit, um Client-Geräte zu authentifizieren und Client-Geräteaktionen zu autorisieren. Diese Komponente ermöglicht es Ihren AWS IoT Objekten, eine Verbindung zu einem Core-Gerät herzustellen.

  Diese Komponente erfordert eine gewisse Konfiguration, um sie zu verwenden. Sie müssen Gruppen von Client-Geräten und die Operationen angeben, zu deren Ausführung jede Gruppe berechtigt ist, z. B. zum Herstellen einer Verbindung und Kommunikation über MQTT. Weitere Informationen finden Sie unter Konfiguration der Client-Geräte-Authentifizierungskomponente .
- MQTT 3.1.1-Broker (Moquette) (`aws.greengrass.clientdevices.mqtt.Moquette`)

  Stellen Sie die Moquette MQTT-Brokerkomponente bereit, um einen leichtgewichtigen MQTT-Broker auszuführen. Der Moquette MQTT-Broker ist mit MQTT 3.1.1 konform und bietet lokale Unterstützung für QoS 0, QoS 1, QoS 2, beibehaltene Nachrichten, Last-Will-Nachrichten und persistente Abonnements.

  Sie müssen diese Komponente nicht für ihre Verwendung konfigurieren. Sie können jedoch den Port konfigurieren, an dem diese Komponente den MQTT-Broker betreibt. Standardmäßig wird Port 8883 verwendet.
- MQTT 5-Broker (EMAX) (`aws.greengrass.clientdevices.mqtt.EMQX`)

  > **Note**
  >
  > Um den EMQX MQTT 5-Broker verwenden zu können, müssen Sie Greengrass kernus v2.6.0 oder höher und die Client-Geräte-Authentifizierung v2.2.0 oder höher verwenden.

Stellen Sie die EMQX MQTT-Brokerkomponente bereit, um MQTT 5.0-Funktionen bei der Kommunikation zwischen Client-Geräten und dem Core-Gerät zu verwenden. Der EMQX MQTT-Broker ist mit MQTT 5.0 kompatibel und bietet Unterstützung für Sitzungs- und Nachrichtenablaufintervalle, Benutzereigenschaften, freigegebene Abonnements, Themenaliasnamen und mehr.

Sie müssen diese Komponente nicht für ihre Verwendung konfigurieren. Sie können jedoch den Port konfigurieren, an dem diese Komponente den MQTT-Broker betreibt. Standardmäßig wird Port 8883 verwendet.

- MQTT-Brücke (`aws.greengrass.clientdevices.mqtt.Bridge`)

  (Optional) Stellen Sie die MQTT-Bridge-Komponente bereit, um Nachrichten zwischen Client-Geräten (lokales MQTT), lokaler Veröffentlichung/Abonnement und AWS IoT Core MQTT weiterzuleiten. Konfigurieren Sie diese Komponente, um Client-Geräte mit zu synchronisieren AWS IoT Core und von Greengrass-Komponenten aus mit Client-Geräten zu interagieren.

  Für diese Komponente ist eine Konfiguration erforderlich. Sie müssen die Themenzuordnungen angeben, an die diese Komponente Nachrichten weiterleitet. Weitere Informationen finden Sie unter Konfiguration der MQTT-Bridge-Komponente .

- IP-Detektor (`aws.greengrass.clientdevices.IPDetector`)

  (Optional) Stellen Sie die IP-Detektorkomponente bereit, um die MQTT-Broker-Endpunkte des Core-Geräts automatisch an den AWS IoT Greengrass Cloud-Service zu melden. Sie können diese Komponente nicht verwenden, wenn Sie über eine komplexe Netzwerkeinrichtung verfügen, z. B. eine, bei der ein Router den MQTT-Broker-Port an das Core-Gerät weiterleitet.

  Sie müssen diese Komponente nicht für ihre Verwendung konfigurieren.

- Schattenmanager (`aws.greengrass.ShadowManager`)

  > **Note**
  >
  > Um Client-Geräteschatten zu verwalten, müssen Sie den Greengrass-Kern v2.6.0 oder höher, den Schattenmanager v2.2.0 oder höher und die MQTT-Bridge v2.2.0 oder höher verwenden.

(Optional) Stellen Sie die Shadow Manager-Komponente bereit, um die Schatten des Client-Geräts auf dem Core-Gerät zu verwalten. Greengrass-Komponenten können Schatten von Client-Geräten abrufen, aktualisieren und löschen, um mit Client-Geräten zu interagieren. Sie können die Shadow Manager-Komponente auch so konfigurieren, dass die Schatten von Client-Geräten mit dem AWS IoT Core Cloud-Service synchronisiert werden.

Um diese Komponente mit Client-Geräteschatten zu verwenden, müssen Sie die MQTT-Bridge-Komponente so konfigurieren, dass Nachrichten zwischen Client-Geräten und dem Shadow Manager weitergeleitet werden, der lokales Veröffentlichen/Abonnement verwendet. Andernfalls erfordert diese Komponente keine Konfiguration, um sie zu verwenden, aber sie erfordert eine Konfiguration, um Geräteschatten zu synchronisieren.

> **Note**
>
> Wir empfehlen, nur eine MQTT-Brokerkomponente bereitzustellen. Die MQTT-Bridge- und IP-Detektorkomponenten funktionieren jeweils nur mit einer MQTT-Brokerkomponente. Wenn Sie mehrere MQTT-Brokerkomponenten bereitstellen, müssen Sie sie so konfigurieren, dass unterschiedliche Ports verwendet werden.

Konfigurieren der Cloud-Erkennung (Konsole)

Sie können die -AWS IoT GreengrassKonsole verwenden, um Client-Geräte zuzuordnen, Core-Geräteendpunkte zu verwalten und Komponenten bereitzustellen, um die Client-Geräteunterstützung zu ermöglichen. Weitere Informationen finden Sie unter Schritt 2: Aktivieren der Client-Geräteunterstützung.

Konfigurieren der Cloud-Erkennung (AWS CLI)

Sie können die AWS Command Line Interface (AWS CLI) verwenden, um Client-Geräte zuzuordnen, Core-Geräteendpunkte zu verwalten und Komponenten bereitzustellen, um Client-Geräteunterstützung zu ermöglichen. Weitere Informationen finden Sie hier:

- Verwalten von Client-Gerätezuordnungen (AWS CLI)
- Verwalten von -Core-Geräteendpunkten
- AWSVon bereitgestellte Client-Gerätekomponenten

- Erstellen von Bereitstellungen

Zuordnen von Client-Geräten

Um die Cloud-Erkennung zu verwenden, verknüpfen Sie Client-Geräte mit einem Core-Gerät, damit sie das Core-Gerät erkennen können. Anschließend können sie die Greengrass-Discovery-API verwenden, um Konnektivitätsinformationen und Zertifikate für ihre zugehörigen Core-Geräte abzurufen.

Trennen Sie Client-Geräte ebenfalls von einem Core-Gerät, damit sie das Core-Gerät nicht erkennen können.

Themen

- Verwalten von Client-Gerätezuordnungen (Konsole)
- Verwalten von Client-Gerätezuordnungen (AWS CLI)
- Verwalten von Client-Gerätezuordnungen (API)

Verwalten von Client-Gerätezuordnungen (Konsole)

Sie können die AWS IoT Greengrass Konsole verwenden, um Client-Gerätezuordnungen anzuzeigen, hinzuzufügen und zu löschen.

So zeigen Sie Client-Gerätezuordnungen für ein Core-Gerät an (Konsole)

1. Navigieren Sie zur AWS IoT Greengrass-Konsole.
2. Wählen Sie Core-Geräte aus.
3. Wählen Sie das zu verwaltende Core-Gerät aus.
4. Wählen Sie auf der Detailseite des Core-Geräts die Registerkarte Client-Geräte aus.
5. Im Abschnitt Zugeordnete Client-Geräte können Sie sehen, welche Client-Geräte (AWS IoT-Objekte) dem Core-Gerät zugeordnet sind.

So verknüpfen Sie Client-Geräte mit einem Core-Gerät (Konsole)

1. Navigieren Sie zur AWS IoT Greengrass-Konsole.
2. Wählen Sie Core-Geräte aus.

3. Wählen Sie das zu verwaltende Core-Gerät aus.
4. Wählen Sie auf der Detailseite des Core-Geräts die Registerkarte Client-Geräte aus.
5. Wählen Sie im Abschnitt Zugeordnete Client-Geräte die Option Client-Geräte zuordnen aus.
6. Gehen Sie im Modal Client-Geräte dem Core-Gerät zuordnen für jedes Client-Gerät wie folgt vor:
   a. Geben Sie den Namen des AWS IoT Objekts ein, das als Client-Gerät zugeordnet werden soll.
   b. Wählen Sie Hinzufügen aus.
7. Wählen Sie Associate aus.

   Die Client-Geräte, die Sie zugeordnet haben, können jetzt die Greengrass-Discovery-API verwenden, um dieses Core-Gerät zu erkennen.

So trennen Sie Clientgeräte von einem Core-Gerät (Konsole)

1. Navigieren Sie zur AWS IoT Greengrass-Konsole.
2. Wählen Sie Core-Geräte aus.
3. Wählen Sie das zu verwaltende Core-Gerät aus.
4. Wählen Sie auf der Detailseite des Core-Geräts die Registerkarte Client-Geräte aus.
5. Wählen Sie im Abschnitt Zugeordnete Client-Geräte jedes Client-Gerät aus, dessen Zuordnung aufgehoben werden soll.
6. Wählen Sie Disassociate (Zuordnung aufheben) aus.
7. Wählen Sie im Bestätigungsmodal die Option Zuordnung aufheben aus.

   Die Client-Geräte, deren Zuordnung Sie aufgehoben haben, können die Greengrass-Erkennungs-API nicht mehr verwenden, um dieses Core-Gerät zu erkennen.

Verwalten von Client-Gerätezuordnungen (AWS CLI)

Sie können die AWS Command Line Interface (AWS CLI) verwenden, um Client-Gerätezuordnungen für ein Core-Gerät zu verwalten.

So zeigen Sie Client-Gerätezuordnungen für ein Core-Gerät an (AWS CLI)

- Verwenden Sie den folgenden Befehl: list-client-devices-associated-with-core-device.

So verknüpfen Sie Client-Geräte mit einem Core-Gerät (AWS CLI)

- Verwenden Sie den folgenden Befehl: batch-associate-client-device-with-core-device.

So trennen Sie Client-Geräte von einem Core-Gerät (AWS CLI)

- Verwenden Sie den folgenden Befehl: batch-disassociate-client-device-from-core-device.

Verwalten von Client-Gerätezuordnungen (API)

Sie können die AWS-API verwenden, um Client-Gerätezuordnungen für ein Core-Gerät zu verwalten.

So zeigen Sie Client-Gerätezuordnungen für ein Core-Gerät an (AWS-API)

- Verwenden Sie die folgende Operation: ListClientDevicesAssociatedWithCoreDevice.

So verknüpfen Sie Client-Geräte mit einem Core-Gerät (AWS-API)

- Verwenden Sie die folgende Operation: BatchAssociateClientDeviceWithCoreDevice.

So trennen Sie Client-Geräte von einem Core-Gerät (AWS-API)

- Verwenden Sie die folgende Operation: BatchDisassociateClientDeviceFromCoreDevice.

Authentifizierung von Clients im Offlinemodus

Mit der Offline-Authentifizierung können Sie Ihr AWS IoT Greengrass Core-Gerät so konfigurieren, dass Client-Geräte eine Verbindung zu einem Core-Gerät herstellen können, auch wenn das Core-Gerät nicht mit der Cloud verbunden ist. Wenn Sie die Offline-Authentifizierung verwenden, können Ihre Greengrass-Geräte weiterhin in einer teilweise Offline-Umgebung arbeiten.

Um die Offline-Authentifizierung für ein Client-Gerät mit einer Verbindung zur Cloud zu verwenden, benötigen Sie Folgendes:

- Ein AWS IoT Greengrass Core-Gerät, auf dem die Authentifizierung auf Client-Geräten Komponente bereitgestellt ist. Sie müssen Version 2.3.0 oder höher für die Offline-Authentifizierung verwenden.
- Eine Cloud-Verbindung für das Kerngerät während der ersten Verbindung der Client-Geräte.

Speichern von Kundenanmeldedaten

Wenn ein Client-Gerät zum ersten Mal eine Verbindung zu einem Core-Gerät herstellt, ruft das Core-Gerät den AWS IoT Greengrass Dienst auf. Wenn Greengrass aufgerufen wird, validiert es die Registrierung des Client-Geräts als eine AWS IoT Sache. Es bestätigt auch, dass das Gerät über ein gültiges Zertifikat verfügt. Das Kerngerät speichert diese Informationen dann lokal.

Wenn das Gerät das nächste Mal eine Verbindung herstellt, versucht das Greengrass-Core-Gerät, das Client-Gerät mit dem AWS IoT Greengrass Dienst zu validieren. Wenn keine Verbindung hergestellt werden kann AWS IoT Greengrass, verwendet das Kerngerät seine lokal gespeicherten Geräteinformationen, um das Client-Gerät zu validieren.

Sie können die Dauer konfigurieren, für die das Greengrass-Core-Gerät Anmeldeinformationen speichert. Sie können das Timeout von einer Minute bis 2.147.483.647 Minuten festlegen, indem Sie die `clientDeviceTrustDurationMinutes` Konfigurationsoption in der Konfiguration der Authentifizierungskomponente für das Client-Gerät festlegen. Die Standardeinstellung ist eine Minute, wodurch die Offline-Authentifizierung effektiv ausgeschaltet wird. Wenn Sie dieses Timeout festlegen, empfehlen wir Ihnen, Ihre Sicherheitsanforderungen zu berücksichtigen. Sie sollten auch berücksichtigen, wie lange Sie erwarten, dass die Kerngeräte laufen, wenn sie nicht mit der Cloud verbunden sind.

Das Kerngerät aktualisiert seinen Anmeldeinformationsspeicher dreimal:

1. Wenn ein Gerät zum ersten Mal eine Verbindung zum Kerngerät herstellt.
2. Wenn das Kerngerät mit der Cloud verbunden ist, wenn ein Client-Gerät erneut eine Verbindung zum Kerngerät herstellt.
3. Wenn das Kerngerät mit der Cloud verbunden ist, einmal täglich, um den gesamten Anmeldeinformationsspeicher zu aktualisieren.

Wenn das Greengrass-Core-Gerät seinen Anmeldeinformationsspeicher aktualisiert, verwendet es den Vorgang. ListClientDevicesAssociatedWithCoreDevice Greengrass aktualisiert nur die Geräte, die durch diesen Vorgang zurückgegeben wurden. Informationen zum Zuordnen eines Client-Geräts zu einem Core-Gerät finden Sie unter. Zuordnen von Client-Geräten

Um den `ListClientDevicesAssociatedWithCoreDevice` Vorgang verwenden zu können, müssen Sie der AWS Identity and Access Management (IAM-) Rolle, die dem ausgeführten Vorgang zugeordnet ist AWS-Konto , die Berechtigung für den Vorgang hinzufügen. AWS IoT Greengrass

Weitere Informationen finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

Verwalten von -Core-Geräteendpunkten

Wenn Sie Cloud Discovery verwenden, speichern Sie MQTT-Broker-Endpunkte für -Core-Geräte im AWS IoT Greengrass Cloud-Service. Client-Geräte stellen eine Verbindung zu herAWS IoT Greengrass, um diese Endpunkte und andere Informationen für ihre zugehörigen Core-Geräte abzurufen.

Für jedes Core-Gerät können Sie Endpunkte automatisch oder manuell verwalten.

- Automatische Verwaltung von Endpunkten mit IP-Detektor

  Sie können die IP-Detektorkomponente bereitstellen, um Core-Geräteendpunkte automatisch für Sie zu verwalten, wenn Sie über eine nicht komplexe Netzwerkeinrichtung verfügen, z. B. wenn sich die Clientgeräte im selben Netzwerk wie das Core-Gerät befinden. Sie können die IP-Detektorkomponente nicht verwenden, wenn sich das Core-Gerät hinter einem Router befindet, der beispielsweise den MQTT-Broker-Port an das Core-Gerät weiterleitet.

  Die IP-Detektorkomponente ist auch nützlich, wenn Sie in Objektgruppen bereitstellen, da sie die Endpunkte für alle Core-Geräte in der Objektgruppe verwaltet. Weitere Informationen finden Sie unter Verwenden Sie den IP-Detektor, um Endpunkte automatisch zu verwalten.
- Manuelles Verwalten von Endpunkten

  Wenn Sie die IP-Detektorkomponente nicht verwenden können, müssen Sie die Endpunkte des Core-Geräts manuell verwalten. Sie können diese Endpunkte mit der Konsole oder der API aktualisieren. Weitere Informationen finden Sie unter Manuelles Verwalten von Endpunkten.

Themen

Verwenden Sie den IP-Detektor, um Endpunkte automatisch zu verwalten

Wenn Sie eine einfache Netzwerkeinrichtung haben, z. B. die Client-Geräte im selben Netzwerk wie das Core-Gerät, können Sie die IP-Detektorkomponente bereitstellen, um Folgendes zu tun:

- Überwachen Sie die lokalen Netzwerkkonnektivitätsinformationen des Greengrass-Core-Geräts. Zu diesen Informationen gehören die Netzwerkendpunkte des Core-Geräts und der Port, an dem der MQTT-Broker arbeitet.
- Melden Sie die Konnektivitätsinformationen des Core-Geräts an den AWS IoT Greengrass Cloud-Service.

Die IP-Detektorkomponente überschreibt Endpunkte, die Sie manuell festlegen.

> **Important**
>
> Die AWS IoT Richtlinie des Core-Geräts muss der `greengrass:UpdateConnectivityInfo` Berechtigung zur Verwendung der IP-Detektorkomponente erteilen. Weitere Informationen finden Sie unter AWS IoT-Richtlinien für Operationen auf Datenebene und Konfigurieren der AWS IoT Objektrichtlinie.

Sie können einen der folgenden Schritte ausführen, um die IP-Detektorkomponente bereitzustellen:

- Verwenden Sie die Seite Erkennung konfigurieren in der -Konsole. Weitere Informationen finden Sie unter Konfigurieren der Cloud-Erkennung (Konsole).
- Erstellen und überarbeiten Sie Bereitstellungen, um den IP-Detektor einzuschließen. Sie können die Konsole, oder AWS API verwendenAWS CLI, um Bereitstellungen zu verwalten. Weitere Informationen finden Sie unter Erstellen von Bereitstellungen.

Bereitstellen der IP-Detektorkomponente (Konsole)

1. Wählen Sie im Navigationsmenü der AWS IoT GreengrassKonsole Komponenten aus.
2. Wählen Sie auf der Seite Komponenten die Registerkarte Öffentliche Komponenten und dann ausaws.greengrass.clientdevices.IPDetector.
3. Wählen Sie auf der aws.greengrass.clientdevices.IPDetector Seite Bereitstellen aus.
4. Wählen Sie unter Zu Bereitstellung hinzufügen eine vorhandene Bereitstellung aus, die Sie überarbeiten möchten, oder wählen Sie aus, eine neue Bereitstellung zu erstellen, und wählen Sie dann Weiter aus.
5. Wenn Sie eine neue Bereitstellung erstellen möchten, wählen Sie das Ziel-Core-Gerät oder die Objektgruppe für die Bereitstellung aus. Wählen Sie auf der Seite Ziel angeben unter Bereitstellungsziel ein Core-Gerät oder eine Objektgruppe und dann Weiter aus.

6. Überprüfen Sie auf der Seite Komponenten auswählen, ob die aws.greengrass.clientdevices.IPDetector Komponente ausgewählt ist, und wählen Sie Weiter aus.
7. Wählen Sie auf der Seite Komponenten konfigurieren die Option aus aws.greengrass.clientdevices.IPDetectorund gehen Sie dann wie folgt vor:

   a. Wählen Sie Komponente konfigurieren aus.

   b. Im Modal konfigurieren aws.greengrass.clientdevices.IPDetector können Sie unter Konfigurationsaktualisierung unter Konfiguration zum Zusammenführen von ein Konfigurationsupdate eingeben, um die IP-Detektorkomponente zu konfigurieren. Sie können eine der folgenden Konfigurationsoptionen angeben:

      - `defaultPort` – (Optional) Der MQTT-Broker-Port, der gemeldet werden soll, wenn diese Komponente IP-Adressen erkennt. Sie müssen diesen Parameter angeben, wenn Sie den MQTT-Broker so konfigurieren, dass er einen anderen Port als den Standardport 8883 verwendet.
      - `includeIPv4LoopbackAddrs` – (Optional) Sie können diese Option aktivieren, um IPv4-Loopback-Adressen zu erkennen und zu melden. Dies sind IP-Adressen, wie z. B. `localhost`, bei denen ein Gerät mit sich selbst kommunizieren kann. Verwenden Sie diese Option in Testumgebungen, in denen das Core-Gerät und das Client-Gerät auf demselben System ausgeführt werden.
      - `includeIPv4LinkLocalAddrs` – (Optional) Sie können diese Option aktivieren, um Link-lokale IPv4-Adressen zu erkennen und zu melden. https://en.wikipedia.org/wiki/Link-local_address Verwenden Sie diese Option, wenn das Netzwerk des Core-Geräts nicht über das Dynamic Host Configuration Protocol (DHCP) oder statisch zugewiesene IP-Adressen verfügt.

      Das Konfigurationsupdate könnte dem folgenden Beispiel ähneln.

      ```
      {
        "defaultPort": "8883",
        "includeIPv4LoopbackAddrs": false,
        "includeIPv4LinkLocalAddrs": false
      }
      ```

   c. Wählen Sie Bestätigen, um das Modal zu schließen, und wählen Sie dann Weiter aus.

8. Behalten Sie auf der Seite Erweiterte Einstellungen konfigurieren die Standardkonfigurationseinstellungen bei und wählen Sie Weiter.
9. Wählen Sie auf der Seite Review (Prüfen) die Option Deploy (Bereitstellen) aus.

   Die Bereitstellung kann bis zu einer Minute dauern.

Bereitstellen der IP-Detektorkomponente (AWS CLI)

Um die IP-Detektorkomponente bereitzustellen, erstellen Sie ein Bereitstellungsdokument, das `aws.greengrass.clientdevices.IPDetector` in das `components` Objekt einschließt, und geben Sie das Konfigurationsupdate für die Komponente an. Folgen Sie den Anweisungen unter Erstellen von Bereitstellungen, um eine neue Bereitstellung zu erstellen oder eine vorhandene Bereitstellung zu überarbeiten.

Sie können beim Erstellen des Bereitstellungsdokuments eine der folgenden Optionen angeben, um die IP-Detektorkomponente zu konfigurieren:

- `defaultPort` – (Optional) Der MQTT-Broker-Port, der gemeldet werden soll, wenn diese Komponente IP-Adressen erkennt. Sie müssen diesen Parameter angeben, wenn Sie den MQTT-Broker so konfigurieren, dass er einen anderen Port als den Standardport 8883 verwendet.
- `includeIPv4LoopbackAddrs` – (Optional) Sie können diese Option aktivieren, um IPv4-Loopback-Adressen zu erkennen und zu melden. Dies sind IP-Adressen, wie z. B. `localhost`, bei denen ein Gerät mit sich selbst kommunizieren kann. Verwenden Sie diese Option in Testumgebungen, in denen das Core-Gerät und das Client-Gerät auf demselben System ausgeführt werden.
- `includeIPv4LinkLocalAddrs` – (Optional) Sie können diese Option aktivieren, um Link-lokale IPv4-Adressen zu erkennen und zu melden. https://en.wikipedia.org/wiki/Link-local_address Verwenden Sie diese Option, wenn das Netzwerk des Core-Geräts nicht über das Dynamic Host Configuration Protocol (DHCP) oder statisch zugewiesene IP-Adressen verfügt.

Das folgende Beispiel für ein partielles Bereitstellungsdokument gibt an, Port 8883 als MQTT-Broker-Port zu melden.

```
{
  ...,
  "components": {
    ...,
    "aws.greengrass.clientdevices.IPDetector": {
```

```
      "componentVersion": "2.1.1",
      "configurationUpdate": {
        "merge": "{\"defaultPort\":\"8883\",}"
      }
    }
  }
}
```

Manuelles Verwalten von Endpunkten

Sie können MQTT-Broker-Endpunkte für -Core-Geräte manuell verwalten.

Jeder MQTT-Broker-Endpunkt enthält die folgenden Informationen:

Endpunkt (`HostAddress`)

Eine IP-Adresse oder DNS-Adresse, an der Client-Geräte eine Verbindung zu einem MQTT-Broker auf dem Core-Gerät herstellen können.

Port (`PortNumber`)

Der Port, an dem der MQTT-Broker auf dem Core-Gerät arbeitet.

Sie können diesen Port auf der Moquette-MQTT-Brokerkomponente konfigurieren, die standardmäßig Port 8883 verwendet.

Metadaten (`Metadata`)

Zusätzliche Metadaten, die Client-Geräten bereitgestellt werden, die eine Verbindung zu diesem Endpunkt herstellen.

Themen

- Verwalten von Endpunkten (Konsole)
- Verwalten von Endpunkten (AWS CLI)
- Verwalten von Endpunkten (API)

Verwalten von Endpunkten (Konsole)

Sie können die AWS IoT Greengrass Konsole verwenden, um Endpunkte für ein Core-Gerät anzuzeigen, zu aktualisieren und zu entfernen.

So verwalten Sie Endpunkte für ein Core-Gerät (Konsole)

1. Navigieren Sie zur AWS IoT Greengrass-Konsole.
2. Wählen Sie Core-Geräte aus.
3. Wählen Sie das zu verwaltende Core-Gerät aus.
4. Wählen Sie auf der Detailseite des Core-Geräts die Registerkarte Client-Geräte aus.
5. Im Abschnitt MQTT-Broker-Endpunkte können Sie die MQTT-Broker-Endpunkte des Core-Geräts sehen. Wählen Sie Endpunkte verwalten aus.
6. Fügen Sie im Modal Endpunkte verwalten MQTT-Broker-Endpunkte für das Core-Gerät hinzu oder entfernen Sie sie.
7. Wählen Sie Aktualisieren.

Verwalten von Endpunkten (AWS CLI)

Sie können die AWS Command Line Interface (AWS CLI) verwenden, um Endpunkte für ein Core-Gerät zu verwalten.

> **Note**
>
> Da die Client-Geräteunterstützung in abwärtskompatibel mit AWS IoT Greengrass V2 istAWS IoT Greengrass V1, können Sie - AWS IoT Greengrass V2 oder AWS IoT Greengrass V1-API-Operationen verwenden, um Core-Geräteendpunkte zu verwalten.

So rufen Sie Endpunkte für ein Core-Gerät ab (AWS CLI)

- Verwenden Sie einen der folgenden Befehle:
  - Greengrassv2: get-connectivity-info
  - Greengrass: get-connectivity-info

So aktualisieren Sie Endpunkte für ein Core-Gerät (AWS CLI)

- Verwenden Sie einen der folgenden Befehle:
  - Greengrassv2: update-connectivity-info
  - Greengrass: update-connectivity-info

Verwalten von Endpunkten (API)

Sie können die AWS-API verwenden, um Endpunkte für ein Core-Gerät zu verwalten.

> **Note**
>
> Da die Client-Geräteunterstützung in abwärtskompatibel mit AWS IoT Greengrass V2 istAWS IoT Greengrass V1, können Sie - AWS IoT Greengrass V2 oder AWS IoT Greengrass V1-API-Operationen verwenden, um Core-Geräteendpunkte zu verwalten.

So rufen Sie Endpunkte für ein Core-Gerät ab (AWS-API)

- Verwenden Sie eine der folgenden Operationen:
  - V2: GetConnectivityInfo
  - V1: GetConnectivityInfo

So aktualisieren Sie Endpunkte für ein Core-Gerät (AWS-API)

- Verwenden Sie eine der folgenden Operationen:
  - V2: UpdateConnectivityInfo
  - V1: UpdateConnectivityInfo

Auswählen eines MQTT-Brokers

AWS IoT Greengrass bietet Optionen, mit denen Sie auswählen können, welcher lokale MQTT-Broker auf Ihren -Core-Geräten ausgeführt werden soll. Client-Geräte stellen eine Verbindung mit dem MQTT-Broker her, der auf einem Core-Gerät ausgeführt wird. Wählen Sie daher einen MQTT-Broker aus, der mit den Client-Geräten kompatibel ist, die Sie verbinden möchten.

> **Note**
>
> Wir empfehlen, nur eine MQTT-Brokerkomponente bereitzustellen. Die MQTT-Bridge- und IP-Detektorkomponenten funktionieren jeweils nur mit einer MQTT-Brokerkomponente. Wenn Sie mehrere MQTT-Brokerkomponenten bereitstellen, müssen Sie sie so konfigurieren, dass unterschiedliche Ports verwendet werden.

Sie können aus den folgenden MQTT-Brokern wählen:

- MQTT-3.1.1-Broker (Moquette) – `aws.greengrass.clientdevices.mqtt.Moquette`

  Wählen Sie diese Option für einen leichtgewichtigen MQTT-Broker, der mit dem MQTT-Standard 3.1.1 kompatibel ist. Der AWS IoT Core MQTT-Broker und AWS IoT Device SDK sind auch mit dem MQTT-Standard 3.1.1 kompatibel, sodass Sie diese Funktionen verwenden können, um eine Anwendung zu erstellen, die MQTT 3.1.1 auf Ihren Geräten und der verwendetAWS Cloud.
- MQTT-5-Broker (EMQX) – `aws.greengrass.clientdevices.mqtt.EMQX`

  Wählen Sie diese Option, um MQTT 5-Funktionen für die Kommunikation zwischen -Core-Geräten und Client-Geräten zu verwenden. Diese Komponente verbraucht mehr Ressourcen als der Moquette MQTT 3.1.1-Broker. Auf Linux-Core-Geräten erfordert sie Docker.

  MQTT 5 ist abwärtskompatibel mit MQTT 3.1.1, sodass Sie Client-Geräte, die MQTT 3.1.1 verwenden, mit diesem Broker verbinden können. Wenn Sie den Moquette MQTT 3.1.1-Broker ausführen, können Sie ihn durch den EMQX MQTT 5-Broker ersetzen, und Client-Geräte können weiterhin wie gewohnt eine Verbindung herstellen und arbeiten.

- Implementieren eines benutzerdefinierten Brokers

  Wählen Sie diese Option, um eine benutzerdefinierte lokale Broker-Komponente für die Kommunikation mit Client-Geräten zu erstellen. Sie können einen benutzerdefinierten lokalen Broker erstellen, der ein anderes Protokoll als MQTT verwendet. AWS IoT Greengrass bietet ein Komponenten-SDK, mit dem Sie Client-Geräte authentifizieren und autorisieren können. Weitere Informationen finden Sie unter Verwenden Sie den AWS IoT Device SDK , um mit dem Greengrass-Kern und anderen Komponenten zu kommunizieren und AWS IoT Core und Authentifizieren und Autorisieren von Client-Geräten.

Client-Geräte mit einemAWS IoT Greengrass Core-Gerät über einen MQTT-Broker verbinden

Wenn Sie einen MQTT-Broker auf IhremAWS IoT Greengrass Core-Gerät verwenden, verwendet das Gerät eine für das Gerät einzigartige Core-Device Certificate Authority (CA), um dem Broker ein Zertifikat für die Herstellung gegenseitiger TLS-Verbindungen mit Clients auszustellen.

AWS IoT Greengrassgeneriert automatisch eine eigene Zertifizierungsstelle für das Kerngerät. Die CA des Kerngeräts wird registriertAWS IoT Greengrass, wenn dieAuthentifizierung auf Client-Geräten

Komponente angeschlossen ist. Die automatisch generierte Zertifizierungsstelle für das Kerngerät ist persistent. Das Gerät verwendet weiterhin dieselbe CA, solange die Authentifizierungskomponente für das Clientgerät konfiguriert ist.

Wenn der MQTT-Broker startet, fordert er ein Zertifikat an. Die Authentifizierungskomponente für das Clientgerät stellt ein X.509-Zertifikat unter Verwendung der Kerngeräte-CA aus. Das Zertifikat wird rotiert, wenn der Broker gestartet wird, wenn das Zertifikat abläuft oder wenn sich Verbindungsinformationen wie die IP-Adresse ändern. Weitere Informationen finden Sie unter Zertifikatrotation auf dem lokalen MQTT-Broker.

Um einen Client mit dem MQTT-Broker zu verbinden, benötigen Sie Folgendes:

- Das Client-Gerät muss über dieAWS IoT Greengrass Core-Geräte-CA verfügen. Sie können diese CA über Cloud Discovery oder durch manuelles Bereitstellen der CA abrufen. Weitere Informationen finden Sie unter Verwenden Sie Ihre eigene Zertifizierungsstelle.
- Der vollqualifizierte Domänenname (FQDN) oder die IP-Adresse des Kerngeräts müssen im von der zentralen Gerätezertifizierungsstelle ausgestellten Brokerzertifikat enthalten sein. Sie stellen dies sicher, indem Sie dieIP-Detektor Komponente verwenden oder die IP-Adresse manuell konfigurieren. Weitere Informationen finden Sie unter Verwalten von -Core-Geräteendpunkten.
- Die Authentifizierungskomponente für das Client-Gerät muss dem Client-Gerät die Berechtigung geben, eine Verbindung zum Greengrass-Core-Gerät herzustellen. Weitere Informationen finden Sie unter Authentifizierung auf Client-Geräten.

Verwenden Sie Ihre eigene Zertifizierungsstelle

Wenn Ihre Client-Geräte nicht auf die Cloud zugreifen können, um Ihr Kerngerät zu finden, können Sie eine zentrale Gerätezertifizierungsstelle (CA) angeben. Ihr Greengrass-Core-Gerät verwendet die Core-Device-CA, um Zertifikate für Ihren MQTT-Broker auszustellen. Sobald Sie das Kerngerät konfiguriert und Ihr Client-Gerät mit seiner CA ausgestattet haben, können Ihre Client-Geräte eine Verbindung zum Endpunkt herstellen und den TLS-Handshake mithilfe der zentralen Geräte-CA (eigene, von Ihnen bereitgestellte oder automatisch generierte) verifizieren.

Um dieAuthentifizierung auf Client-Geräten Komponente so zu konfigurieren, dass sie Ihre zentrale Geräte-CA verwendet, legen Sie den`certificateAuthority` Konfigurationsparameter fest, wenn Sie die Komponente bereitstellen. Sie müssen bei der Konfiguration angeben:

- Der Standort eines CA-Zertifikats für das Kerngerät.
- Der private Schlüssel des CA-Zertifikats des Kerngeräts.

- (Optional) Die Zertifikatskette zum Stammzertifikat, wenn es sich bei der Kerngeräte-CA um eine Zwischenzertifizierungsstelle handelt.

Wenn Sie eine Zertifizierungsstelle für das Kerngerät angeben, wird die CA in der CloudAWS IoT Greengrass registriert.

Sie können Ihre Zertifikate in einem Hardware-Sicherheitsmodul oder im Dateisystem speichern. Das folgende Beispiel zeigt eine`certificateAuthority` Konfiguration für eine Zwischenzertifizierungsstelle, die mit HSM/TPM gespeichert wurde. Beachten Sie, dass die Zertifikatskette nur auf der Festplatte gespeichert werden kann.

```
"certificateAuthority": {
    "certificateUri": "pkcs11:object=CustomerIntermediateCA;type=cert",
    "privateKeyUri": "pkcs11:object=CustomerIntermediateCA;type=private"
    "certificateChainUri": "file:///home/ec2-user/creds/certificateChain.pem",
  }
```

In diesem Beispiel konfiguriert der`certificateAuthority` Konfigurationsparameter die Authentifizierungskomponente für das Clientgerät so, dass sie eine Zwischenzertifizierungsstelle aus dem Dateisystem verwendet:

```
"certificateAuthority": {
    "certificateUri": "file:///home/ec2-user/creds/intermediateCA.pem",
    "privateKeyUri": "file:///home/ec2-user/creds/intermediateCA.privateKey.pem",
    "certificateChainUri": "file:///home/ec2-user/creds/certificateChain.pem",
  }
```

Gehen Sie wie folgt vor, um die Geräte mit IhremAWS IoT Greengrass Core-Gerät zu verbinden:

1. Verwenden der eigenen Greengrass eigenen eigenen eigenen eigenen eigenen eigenen eigenen eigenen eigenen eigenen eigenen eigenen eigenen eigenen eigenen eigenen eigenen eigenen eigenen Wir empfehlen die Verwendung einer Zwischenzertifizierungsstelle als bewährte Sicherheitsmethode.
2. Stellen Sie das CA-Zwischenzertifikat, den privaten Schlüssel und die Zertifikatskette Ihrer Stammzertifizierungsstelle für das Greengrass-Core-Gerät zur Verfügung. Weitere Informationen finden Sie unter Authentifizierung auf Client-Geräten. Die Zwischenzertifizierungsstelle wird zur zentralen Geräte-CA für das Greengrass-Core-Gerät, und das Gerät registriert die CA beiAWS IoT Greengrass.

3. Registrieren Sie das Client-Gerät alsAWS IoT Ding. Weitere Informationen finden Sie im AWS IoT CoreDeveloper Guide unter Erstellen eines Dingobjekts. Fügen Sie Ihrem Client-Gerät den privaten Schlüssel, den öffentlichen Schlüssel, das Gerätezertifikat und das Root-CA-Zertifikat hinzu. Wie Sie die Informationen hinzufügen, hängt von Ihrem Gerät und Ihrer Software ab.

Sobald Sie Ihr Gerät konfiguriert haben, können Sie das Zertifikat und die öffentliche Schlüsselkette verwenden, um eine Verbindung zum Greengrass-Core-Gerät herzustellen. Ihre Software ist dafür verantwortlich, die wichtigsten Geräteendpunkte zu finden. Sie können den Endpunkt für das Kerngerät manuell festlegen. Weitere Informationen finden Sie unter Manuelles Verwalten von Endpunkten.

Testen der Kommunikation von Client-Geräten

Client-Geräte können die verwenden, AWS IoT Device SDK um ein Core-Gerät zu erkennen, eine Verbindung herzustellen und mit ihm zu kommunizieren. Sie können den Greengrass-Discovery-Client in der verwenden, AWS IoT Device SDK um die Greengrass-Discovery-API zu verwenden, die Informationen über Core-Geräte zurückgibt, mit denen ein Client-Gerät eine Verbindung herstellen kann. Die API-Antwort enthält MQTT-Broker-Endpunkte zum Herstellen einer Verbindung und Zertifikate zum Überprüfen der Identität jedes Core-Geräts. Anschließend kann das Client-Gerät jeden Endpunkt ausprobieren, bis es erfolgreich eine Verbindung zu einem Core-Gerät herstellt.

Client-Geräte können nur -Core-Geräte erkennen, denen Sie sie zuordnen. Bevor Sie die Kommunikation zwischen einem Client-Gerät und einem Core-Gerät testen, müssen Sie das Client-Gerät dem Core-Gerät zuordnen. Weitere Informationen finden Sie unter Zuordnen von Client-Geräten.

Die Greengrass Discovery API gibt die MQTT-Broker-Endpunkte des Core-Geräts zurück, die Sie angeben. Sie können die IP-Detektorkomponente verwenden, um diese Endpunkte für Sie zu verwalten, oder Sie können sie für jedes Kerngerät manuell verwalten. Weitere Informationen finden Sie unter Verwalten von -Core-Geräteendpunkten.

> **Note**
>
> Um die Greengrass-Erkennungs-API verwenden zu können, muss ein Client-Gerät über die `-greengrass:Discover`Berechtigung verfügen. Weitere Informationen finden Sie unter Minimale AWS IoT Richtlinie für Client-Geräte.

Die AWS IoT Device SDK ist in mehreren Programmiersprachen verfügbar. Weitere Informationen finden Sie unter AWS IoT Geräte-SDKs im AWS IoT Core -Entwicklerhandbuch.

Themen

Testen der Kommunikation (Python)

In diesem Abschnitt verwenden Sie das Greengrass-Erkennungsbeispiel in der AWS IoT Device SDK v2 für Python, um die Kommunikation zwischen einem Client-Gerät und einem Core-Gerät zu testen.

> **Important**
>
> Um AWS IoT Device SDK v2 für Python verwenden zu können, muss ein Gerät Python 3.6 oder höher ausführen.

So testen Sie die Kommunikation (AWS IoT Device SDK v2 für Python)

1. Laden Sie AWS IoT Device SDK v2 für Python herunter und installieren Sie es auf dem AWS IoT Objekt, um eine Verbindung als Client-Gerät herzustellen.

   Gehen Sie auf dem Client-Gerät wie folgt vor:

   a. Klonen Sie das Repository AWS IoT Device SDK v2 für Python, um es herunterzuladen.

   ```
   git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
   ```

   b. Installieren Sie AWS IoT Device SDK v2 für Python.

   ```
   python3 -m pip install --user ./aws-iot-device-sdk-python-v2
   ```

2. Wechseln Sie in den Ordner Beispiele in AWS IoT Device SDK v2 für Python.

   ```
   cd aws-iot-device-sdk-python-v2/samples
   ```

3. Führen Sie die Greengrass-Erkennungs-Beispielanwendung aus. Diese Anwendung erwartet Argumente, die den Objektnamen des Client-Geräts, das zu verwendende MQTT-Thema und die Nachricht sowie die Zertifikate angeben, die die Verbindung authentifizieren und sichern. Im folgenden Beispiel wird eine Hello World-Nachricht an das `clients/`*`MyClientDevice1`*`/hello/world` Thema gesendet.

   - Ersetzen Sie *`MyClientDevice1`* durch den Objektnamen des Client-Geräts.
   - Ersetzen Sie *`~/certs/AmazonRootCA1.pem`* durch den Pfad zum Amazon-Root-CA-Zertifikat auf dem Client-Gerät.
   - Ersetzen Sie *`~/certs/device.pem.crt`* durch den Pfad zum Gerätezertifikat auf dem Client-Gerät.
   - Ersetzen Sie *`~/certs/private.pem.key`* durch den Pfad zur Datei mit dem privaten Schlüssel auf dem Client-Gerät.
   - Ersetzen Sie *`us-east-1`* durch die AWS Region, in der Ihr Client-Gerät und Ihr Core-Gerät betrieben werden.

   ```
   python3 basic_discovery.py \\
     --thing_name MyClientDevice1 \\
     --topic 'clients/MyClientDevice1/hello/world' \\
     --message 'Hello World!' \\
     --ca_file ~/certs/AmazonRootCA1.pem \\
     --cert ~/certs/device.pem.crt \\
     --key ~/certs/private.pem.key \\
     --region us-east-1 \\
     --verbosity Warn
   ```

   Die Discovery-Beispielanwendung sendet die Nachricht 10 Mal und trennt die Verbindung. Es abonniert auch dasselbe Thema, in dem es Nachrichten veröffentlicht. Wenn die Ausgabe angibt, dass die Anwendung MQTT-Nachrichten zum Thema erhalten hat, kann das Client-Gerät erfolgreich mit dem Core-Gerät kommunizieren.

   ```
   Performing greengrass discovery...
   awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup
   coreDevice-MyGreengrassCore',
    cores=[awsiot.greengrass_discovery.GGCore(thing_arn='arn:aws:iot:us-
   east-1:123456789012:thing/MyGreengrassCore',
    connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='203.0.113.0',
   ```

```
 host_address='203.0.113.0', metadata='', port=8883)])],
 certificate_authorities=['-----BEGIN CERTIFICATE-----\
MIICiT...EXAMPLE=\
-----END CERTIFICATE-----\
'])])
Trying core arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore at host
 203.0.113.0 port 8883
Connected!
Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 0}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 0}'
Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 1}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 1}'

...

Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 9}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 9}'
```

Wenn die Anwendung stattdessen einen Fehler ausgibt, finden Sie weitere Informationen unter Fehlerbehebung bei Greengrass-Erkennungsproblemen.

Sie können auch die Greengrass-Protokolle auf dem Core-Gerät anzeigen, um zu überprüfen, ob das Client-Gerät erfolgreich eine Verbindung herstellt und Nachrichten sendet. Weitere Informationen finden Sie unter Überwachen von AWS IoT Greengrass Protokollen.

Testen der Kommunikation (C++)

In diesem Abschnitt verwenden Sie das Greengrass-Erkennungsbeispiel in der AWS IoT Device SDK v2 für C++, um die Kommunikation zwischen einem Client-Gerät und einem Core-Gerät zu testen.

Um AWS IoT Device SDK v2 für C++ zu erstellen, muss ein Gerät über die folgenden Tools verfügen:

- C++ 11 oder höher

- CMake 3.1 oder höher
- Einer der folgenden Compiler:
  - GCC 4.8 oder höher
  - Clang 3.9 oder höher
  - MSVC 2015 oder höher

So testen Sie die Kommunikation (AWS IoT Device SDK v2 für C++)

1. Laden Sie AWS IoT Device SDK v2 für C++ herunter und erstellen Sie es auf dem AWS IoT Objekt, um eine Verbindung als Client-Gerät herzustellen.

   Gehen Sie auf dem Client-Gerät wie folgt vor:

   a. Erstellen Sie einen Ordner für den Workspace AWS IoT Device SDK v2 für C++ und wechseln Sie zu diesem.

   ```
   cd
   mkdir iot-device-sdk-cpp
   cd iot-device-sdk-cpp
   ```

   b. Klonen Sie das Repository AWS IoT Device SDK v2 für C++, um es herunterzuladen. Das `--recursive` Flag gibt an, dass Submodule heruntergeladen werden sollen.

   ```
   git clone --recursive https://github.com/aws/aws-iot-device-sdk-cpp-v2.git
   ```

   c. Erstellen Sie einen Ordner für die Build-Ausgabe AWS IoT Device SDK v2 für C++ und wechseln Sie zu diesem.

   ```
   mkdir aws-iot-device-sdk-cpp-v2-build
   cd aws-iot-device-sdk-cpp-v2-build
   ```

   d. Erstellen Sie AWS IoT Device SDK v2 für C++.

   ```
   cmake -DCMAKE_INSTALL_PREFIX="~/iot-device-sdk-cpp" -
   DCMAKE_BUILD_TYPE="Release" ../aws-iot-device-sdk-cpp-v2
   cmake --build . --target install
   ```

2. Erstellen Sie die Greengrass Discovery-Beispielanwendung in der AWS IoT Device SDK v2 für C ++. Gehen Sie wie folgt vor:

a. Wechseln Sie in den Greengrass Discovery-Beispielordner in AWS IoT Device SDK v2 für C ++.

```
cd ../aws-iot-device-sdk-cpp-v2/samples/greengrass/basic_discovery
```

b. Erstellen Sie einen Ordner für die Greengrass-Erkennungsbeispiel-Build-Ausgabe und ändern Sie in diesen.

```
mkdir build
cd build
```

c. Erstellen Sie die Greengrass Discovery-Beispielanwendung.

```
cmake -DCMAKE_PREFIX_PATH="~/iot-device-sdk-cpp" -
DCMAKE_BUILD_TYPE="Release" ..
cmake --build . --config "Release"
```

3. Führen Sie die Greengrass-Erkennungs-Beispielanwendung aus. Diese Anwendung erwartet Argumente, die den Objektnamen des Client-Geräts, das zu verwendende MQTT-Thema und die Zertifikate angeben, die die Verbindung authentifizieren und sichern. Im folgenden Beispiel wird das `clients/`*`MyClientDevice1`*`/hello/world` Thema abonniert und eine Nachricht veröffentlicht, die Sie in der Befehlszeile für dasselbe Thema eingeben.

   - Ersetzen Sie *`MyClientDevice1`* durch den Objektnamen des Client-Geräts.
   - Ersetzen Sie *`~/certs/AmazonRootCA1.pem`* durch den Pfad zum Amazon-Root-CA-Zertifikat auf dem Client-Gerät.
   - Ersetzen Sie *`~/certs/device.pem.crt`* durch den Pfad zum Gerätezertifikat auf dem Client-Gerät.
   - Ersetzen Sie *`~/certs/private.pem.key`* durch den Pfad zur Datei mit dem privaten Schlüssel auf dem Client-Gerät.
   - Ersetzen Sie *`us-east-1`* durch die AWS Region, in der Ihr Client-Gerät und Ihr Core-Gerät betrieben werden.

```
./basic-discovery \
  --thing_name MyClientDevice1 \
  --topic 'clients/MyClientDevice1/hello/world' \
  --ca_file ~/certs/AmazonRootCA1.pem \
```

```
  --cert ~/certs/device.pem.crt \
  --key ~/certs/private.pem.key \
  --region us-east-1
```

Die Discovery-Beispielanwendung abonniert das Thema und fordert Sie auf, eine Nachricht zur Veröffentlichung einzugeben.

```
Connecting to group greengrassV2-coreDevice-MyGreengrassCore with thing arn
 arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore, using endpoint
 203.0.113.0:8883
Connected to group greengrassV2-coreDevice-MyGreengrassCore, using connection to
 203.0.113.0:8883
Successfully subscribed to clients/MyClientDevice1/hello/world
Enter the message you want to publish to topic clients/MyClientDevice1/hello/world
 and press enter. Enter 'exit' to exit this program.
```

Wenn die Anwendung stattdessen einen Fehler ausgibt, finden Sie weitere Informationen unter Fehlerbehebung bei Greengrass-Erkennungsproblemen.

4. Geben Sie eine Nachricht ein, z. B. **`Hello World!`**.

```
Enter the message you want to publish to topic clients/MyClientDevice1/hello/world
 and press enter. Enter 'exit' to exit this program.
Hello World!
```

Wenn die Ausgabe angibt, dass die Anwendung die MQTT-Nachricht zum Thema erhalten hat, kann das Client-Gerät erfolgreich mit dem Core-Gerät kommunizieren.

```
Operation on packetId 2 Succeeded
Publish received on topic clients/MyClientDevice1/hello/world
Message:
Hello World!
```

Sie können auch die Greengrass-Protokolle auf dem Core-Gerät anzeigen, um zu überprüfen, ob das Client-Gerät erfolgreich eine Verbindung herstellt und Nachrichten sendet. Weitere Informationen finden Sie unter Überwachen von AWS IoT Greengrass Protokollen.

Testen der Kommunikation (JavaScript)

In diesem Abschnitt verwenden Sie das Greengrass-Erkennungsbeispiel in der AWS IoT Device SDK v2 für JavaScript , um die Kommunikation zwischen einem Client-Gerät und einem Core-Gerät zu testen.

> **Important**
>
> Um AWS IoT Device SDK v2 für verwenden zu können JavaScript, muss ein Gerät Node v10.0 oder höher ausführen.

So testen Sie die Kommunikation (AWS IoT Device SDK v2 für JavaScript)

1. Laden Sie AWS IoT Device SDK v2 für JavaScript herunter und installieren Sie es auf dem AWS IoT Objekt, um eine Verbindung als Client-Gerät herzustellen.

   Gehen Sie auf dem Client-Gerät wie folgt vor:

   a. Klonen Sie das AWS IoT Device SDK v2 für das JavaScript Repository, um es herunterzuladen.

   ```
   git clone https://github.com/aws/aws-iot-device-sdk-js-v2.git
   ```

   b. Installieren Sie AWS IoT Device SDK v2 für JavaScript.

   ```
   cd aws-iot-device-sdk-js-v2
   npm install
   ```

2. Wechseln Sie in den Greengrass Discovery-Beispielordner in der AWS IoT Device SDK v2 für JavaScript.

   ```
   cd samples/node/basic_discovery
   ```

3. Installieren Sie die Greengrass-Discovery-Beispielanwendung.

   ```
   npm install
   ```

4. Führen Sie die Greengrass-Erkennungs-Beispielanwendung aus. Diese Anwendung erwartet Argumente, die den Objektnamen des Client-Geräts, das zu verwendende MQTT-Thema und die Nachricht sowie die Zertifikate angeben, die die Verbindung authentifizieren und sichern.

Im folgenden Beispiel wird eine Hello World-Nachricht an das `clients/MyClientDevice1/hello/world` Thema gesendet.

- Ersetzen Sie *MyClientDevice1* durch den Objektnamen des Client-Geräts.
- Ersetzen Sie *~/certs/AmazonRootCA1.pem* durch den Pfad zum Amazon-Root-CA-Zertifikat auf dem Client-Gerät.
- Ersetzen Sie *~/certs/device.pem.crt* durch den Pfad zum Gerätezertifikat auf dem Client-Gerät.
- Ersetzen Sie *~/certs/private.pem.key* durch den Pfad zur Datei mit dem privaten Schlüssel auf dem Client-Gerät.
- Ersetzen Sie *us-east-1* durch die AWS Region, in der Ihr Client-Gerät und Ihr Core-Gerät betrieben werden.

```
node dist/index.js \
  --thing_name MyClientDevice1 \
  --topic 'clients/MyClientDevice1/hello/world' \
  --message 'Hello World!' \
  --ca_file ~/certs/AmazonRootCA1.pem \
  --cert ~/certs/device.pem.crt \
  --key ~/certs/private.pem.key \
  --region us-east-1 \
  --verbose warn
```

Die Discovery-Beispielanwendung sendet die Nachricht 10 Mal und trennt die Verbindung. Es abonniert auch dasselbe Thema, in dem es Nachrichten veröffentlicht. Wenn die Ausgabe angibt, dass die Anwendung MQTT-Nachrichten zum Thema erhalten hat, kann das Client-Gerät erfolgreich mit dem Core-Gerät kommunizieren.

```
Discovery Response:
{"gg_groups":[{"gg_group_id":"greengrassV2-coreDevice-
MyGreengrassCore","cores":[{"thing_arn":"arn:aws:iot:us-
east-1:123456789012:thing/MyGreengrassCore","connectivity":
[{"id":"203.0.113.0","host_address":"203.0.113.0","port":8883,"metadata":""}]}],"certificat
["-----BEGIN CERTIFICATE-----\nMIICiT...EXAMPLE=\n-----END CERTIFICATE-----\n"]}]}
Trying
 endpoint={"id":"203.0.113.0","host_address":"203.0.113.0","port":8883,"metadata":""}
```

```
[WARN] [2021-06-12T00:46:45Z] [00007f90c0e8d700] [socket] - id=0x7f90b8018710
 fd=26: setsockopt() for NO_SIGNAL failed with errno 92. If you are having SIGPIPE
 signals thrown, you may want to install a signal trap in your application layer.
Connected to
 endpoint={"id":"203.0.113.0","host_address":"203.0.113.0","port":8883,"metadata":""}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":1}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":2}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":3}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":4}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":5}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":6}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":7}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":8}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":9}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":10}
Complete!
```

Wenn die Anwendung stattdessen einen Fehler ausgibt, finden Sie weitere Informationen unter Fehlerbehebung bei Greengrass-Erkennungsproblemen.

Sie können auch die Greengrass-Protokolle auf dem Core-Gerät anzeigen, um zu überprüfen, ob das Client-Gerät erfolgreich eine Verbindung herstellt und Nachrichten sendet. Weitere Informationen finden Sie unter Überwachen von AWS IoT Greengrass Protokollen.

Testen der Kommunikation (Java)

In diesem Abschnitt verwenden Sie das Greengrass-Erkennungsbeispiel in AWS IoT Device SDK v2 für Java, um die Kommunikation zwischen einem Client-Gerät und einem Core-Gerät zu testen.

> **Important**
>
> Um AWS IoT Device SDK v2 für Java zu erstellen, muss ein Gerät über die folgenden Tools verfügen:
>
> - Java 8 oder höher, wobei auf den Java-Ordner `JAVA_HOME` verweist.
> - Apache Maven

So testen Sie die Kommunikation (AWS IoT Device SDK v2 für Java)

1. Laden Sie AWS IoT Device SDK v2 für Java herunter und erstellen Sie es auf dem AWS IoT Objekt, um eine Verbindung als Client-Gerät herzustellen.

   Gehen Sie auf dem Client-Gerät wie folgt vor:

   a. Klonen Sie das Repository AWS IoT Device SDK v2 für Java, um es herunterzuladen.

   ```
   git clone https://github.com/aws/aws-iot-device-sdk-java-v2.git
   ```

   b. Wechseln Sie zum Ordner AWS IoT Device SDK v2 für Java.

   c. Erstellen Sie AWS IoT Device SDK v2 für Java.

   ```
   cd aws-iot-device-sdk-java-v2
   mvn versions:use-latest-versions -Dincludes="software.amazon.awssdk.crt*"
   mvn clean install
   ```

2. Führen Sie die Greengrass-Erkennungs-Beispielanwendung aus. Diese Anwendung erwartet Argumente, die den Objektnamen des Client-Geräts, das zu verwendende MQTT-Thema und die Zertifikate angeben, die die Verbindung authentifizieren und sichern. Im folgenden Beispiel wird das `clients/`*`MyClientDevice1`*`/hello/world` Thema abonniert und eine Nachricht veröffentlicht, die Sie in der Befehlszeile für dasselbe Thema eingeben.

   - Ersetzen Sie beide Instances von *`MyClientDevice1`* durch den Objektnamen des Client-Geräts.

- Ersetzen Sie *$HOME/certs/AmazonRootCA1.pem* durch den Pfad zum Amazon-Root-CA-Zertifikat auf dem Client-Gerät.
- Ersetzen Sie *$HOME/certs/device.pem.crt* durch den Pfad zum Gerätezertifikat auf dem Client-Gerät.
- Ersetzen Sie *$HOME/certs/private.pem.key* durch den Pfad zur Datei mit dem privaten Schlüssel auf dem Client-Gerät.
- Ersetzen Sie *us-east-1* durch die AWS-Region, in der Ihr Client-Gerät und Ihr Core-Gerät ausgeführt werden.

```
DISCOVERY_SAMPLE_ARGS="--thing_name MyClientDevice1 \
  --topic 'clients/MyClientDevice1/hello/world' \
  --ca_file $HOME/certs/AmazonRootCA1.pem \
  --cert $HOME/certs/device.pem.crt \
  --key $HOME/certs/private.pem.key \
  --region us-east-1"

mvn exec:java -pl samples/Greengrass \
  -Dexec.mainClass=greengrass.BasicDiscovery \
  -Dexec.args="$DISCOVERY_SAMPLE_ARGS"
```

Die Discovery-Beispielanwendung abonniert das Thema und fordert Sie auf, eine Nachricht zur Veröffentlichung einzugeben.

```
Connecting to group ID greengrassV2-coreDevice-MyGreengrassCore, with thing
 arn arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore, using endpoint
 203.0.113.0:8883
Started a clean session
Enter the message you want to publish to topic clients/MyClientDevice1/hello/world
 and press Enter. Type 'exit' or 'quit' to exit this program:
```

Wenn die Anwendung stattdessen einen Fehler ausgibt, finden Sie weitere Informationen unter Fehlerbehebung bei Greengrass-Erkennungsproblemen.

3. Geben Sie eine Nachricht ein, z. B. **`Hello World!`**.

```
Enter the message you want to publish to topic clients/MyClientDevice1/hello/world
 and press Enter. Type 'exit' or 'quit' to exit this program:
Hello World!
```

Wenn die Ausgabe angibt, dass die Anwendung die MQTT-Nachricht zum Thema erhalten hat, kann das Client-Gerät erfolgreich mit dem Core-Gerät kommunizieren.

```
Message received on topic clients/MyClientDevice1/hello/world: Hello World!
```

Sie können auch die Greengrass-Protokolle auf dem Core-Gerät anzeigen, um zu überprüfen, ob das Client-Gerät erfolgreich eine Verbindung herstellt und Nachrichten sendet. Weitere Informationen finden Sie unter Überwachen von AWS IoT Greengrass Protokollen.

RESTful-API zur Greengrass-Erkennung

AWS IoT Greengrass stellt den `Discover` API-Vorgang bereit, mit dem Client-Geräte Greengrass-Core-Geräte identifizieren können, mit denen sie eine Verbindung herstellen können. Client-Geräte verwenden diesen Vorgang auf Datenebene, um Informationen abzurufen, die für die Verbindung mit Greengrass-Core-Geräten erforderlich sind, bei denen Sie sie dem BatchAssociateClientDeviceWithCoreDevice API-Vorgang zuordnen. Wenn ein Client-Gerät online geht, kann es eine Verbindung zum AWS IoT Greengrass Cloud-Service herstellen und die Erkennungs-API verwenden, um Folgendes zu finden:

- Die IP-Adresse und der Port für jedes zugehörige Greengrass-Kerngerät.
- Das CA-Zertifikat des Core-Geräts, das Client-Geräte zur Authentifizierung des Greengrass-Core-Geräts verwenden können.

> **Note**
>
> Client-Geräte können den Discovery-Client auch in der verwendenAWS IoT Device SDK, um Konnektivitätsinformationen für Greengrass-Core-Geräte zu ermitteln. Der Discovery-Client verwendet die Discovery-API. Weitere Informationen finden Sie hier:
>
> - Testen der Kommunikation von Client-Geräten
> - Greengrass Discovery RESTful API im AWS IoT Greengrass Version 1 Entwicklerhandbuch für .

Um diesen API-Vorgang zu verwenden, senden Sie HTTP-Anforderungen an die Discovery-API auf dem Greengrass-Datenebene-Endpunkt. Dieser API-Endpunkt hat das folgende Format.

```
https://greengrass-ats.iot.region.amazonaws.com:port/greengrass/discover/thing/thing-name
```

Eine Liste der unterstützten AWS-Regionen und Endpunkte für die AWS IoT Greengrass Discovery-API finden Sie unter -AWS IoT Greengrass V2Endpunkte und -Kontingente im Allgemeine AWS-Referenz. Diese API-Operation ist nur auf dem Greengrass-Endpunkt der Datenebene verfügbar. Der Endpunkt der Steuerebene, den Sie zum Verwalten von Komponenten und Bereitstellungen verwenden, unterscheidet sich vom Endpunkt der Datenebene.

> **Note**
>
> Die Erkennungs-API ist für AWS IoT Greengrass V1 und identischAWS IoT Greengrass V2. Wenn Sie Client-Geräte haben, die eine Verbindung zu einem -AWS IoT Greengrass V1Core herstellen, können Sie sie mit AWS IoT Greengrass V2-Core-Geräten verbinden, ohne den Code auf den Client-Geräten zu ändern. Weitere Informationen finden Sie unter Greengrass Discovery RESTful API im AWS IoT Greengrass Version 1 -Entwicklerhandbuch.

Themen

Erkennungsauthentifizierung und -autorisierung

Um die Discovery-API zum Abrufen von Konnektivitätsinformationen zu verwenden, muss ein Client-Gerät die gegenseitige TLS-Authentifizierung mit einem X.509-Client-Zertifikat zur Authentifizierung verwenden. Weitere Informationen finden Sie unter X.509-Clientzertifikate im AWS IoT Core Entwicklerhandbuch für .

Ein Client-Gerät muss auch über die Berechtigung zum Ausführen der `greengrass:Discover` Aktion verfügen. Die folgende AWS IoT Beispielrichtlinie erlaubt es einem -AWS IoTObjekt mit dem Namen `MyClientDevice1` für `Discover` sich selbst auszuführen.

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "greengrass:Discover",
      "Resource": [
        "arn:aws:iot:us-west-2:123456789012:thing/MyClientDevice1"
      ]
    }
  ]
}
```

> **Important**
>
> Objektrichtlinienvariablen (`iot:Connection.Thing.*`) werden für in AWS IoT Richtlinien für -Core-Geräte oder Greengrass-Operationen auf Datenebene nicht unterstützt. Stattdessen können Sie einen Platzhalter verwenden, der mehreren Geräten entspricht, die ähnliche Namen haben. Sie können beispielsweise angeben, `MyGreengrassDevice*` dass mit `MyGreengrassDevice1`, `MyGreengrassDevice2`usw. übereinstimmt.

Weitere Informationen finden Sie unter -AWS IoT CoreRichtlinien im AWS IoT Core - Entwicklerhandbuch.

Anforderung

Die Anforderung enthält die Standard-HTTP-Header und wird an den Greengrass-Erkennungsendpunkt gesendet, wie in den folgenden Beispielen gezeigt.

Die Portnummer hängt davon ab, ob das Core-Gerät so konfiguriert ist, dass HTTPS-Datenverkehr über Port 8443 oder Port 443 gesendet wird. Weitere Informationen finden Sie unter the section called “Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy”.

> **Note**
>
> In diesen Beispielen wird der Amazon Trust Services (ATS)-Endpunkt verwendet, der mit den empfohlenen ATS-Stammzertifizierungsstellenzertifikaten funktioniert. Endpunkte müssen dem Typ des Stammzertifizierungsstellenzertifikats entsprechen.

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
```

> **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` in der übergeben`ProtocolNameList`. Weitere Informationen finden Sie unter Protokolle im AWS IoT -Entwicklerhandbuch.

Antwort

Bei Erfolg enthält der Antwort-Header den HTTP 200-Statuscode und der Antworttext enthält das Discover-Antwortdokument.

> **Note**
>
> Da dieselbe Erkennungs-API wie AWS IoT Greengrass V2 verwendetAWS IoT Greengrass V1, organisiert die Antwort Informationen nach AWS IoT Greengrass V1 Konzepten wie Greengrass-Gruppen. Die Antwort enthält eine Liste von Greengrass-Gruppen. In befindet sich AWS IoT Greengrass V2jedes Core-Gerät in einer eigenen Gruppe, wobei die Gruppe nur dieses Core-Gerät und seine Konnektivitätsinformationen enthält.

Beispieldokumente für Discovery-Antwort

Das folgende Dokument zeigt die Antwort für ein Client-Gerät, das einem Greengrass-Kerngerät zugeordnet ist. Das Core-Gerät verfügt über einen Endpunkt und ein CA-Zertifikat.

```
{
  "GGGroups": [
```

```
    {
      "GGGroupId": "greengrassV2-coreDevice-core-device-01-thing-name",
      "Cores": [
        {
          "thingArn": "core-device-01-thing-arn",
          "Connectivity": [
            {
              "id": "core-device-01-connection-id",
              "hostAddress": "core-device-01-address",
              "portNumber": core-device-01-port,
              "metadata": "core-device-01-description"
            }
          ]
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
      ]
    }
  ]
}
```

Das folgende Dokument zeigt die Antwort für ein Client-Gerät, das zwei -Core-Geräten zugeordnet ist. Die -Core-Geräte verfügen über mehrere Endpunkte und mehrere Gruppen-CA-Zertifikate.

```
{
  "GGGroups": [
    {
      "GGGroupId": "greengrassV2-coreDevice-core-device-01-thing-name",
      "Cores": [
        {
          "thingArn": "core-device-01-thing-arn",
          "Connectivity": [
            {
              "id": "core-device-01-connection-id",
              "hostAddress": "core-device-01-address",
              "portNumber": core-device-01-port,
              "metadata": "core-device-01-connection-1-description"
            },
            {
              "id": "core-device-01-connection-id-2",
              "hostAddress": "core-device-01-address-2",
              "portNumber": core-device-01-port-2,
```

```
              "metadata": "core-device-01-connection-2-description"
            }
          ]
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
      ]
    },
    {
      "GGGroupId": "greengrassV2-coreDevice-core-device-02-thing-name",
      "Cores": [
        {
          "thingArn":"core-device-02-thing-arn",
          "Connectivity" : [
            {
              "id": "core-device-02-connection-id",
              "hostAddress": "core-device-02-address",
              "portNumber": core-device-02-port,
              "metadata": "core-device-02-connection-1-description"
            }
          ]
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
      ]
    }
  ]
}
```

Testen der Discovery-API mit cURL

Wenn Sie cURL installiert haben, können Sie die Discovery-API testen. Das folgende Beispiel gibt die Zertifikate eines Client-Geräts an, um eine Anforderung an den Greengrass-Discovery-API-Endpunkt zu authentifizieren.

```
curl -i \
  --cert 1a23bc4d56.cert.pem \
```

```
  --key 1a23bc4d56.private.key \
  https://greengrass-ats.iot.us-west-2.amazonaws.com:8443/greengrass/discover/
thing/MyClientDevice1
```

> **Note**
>
> Das `-i` Argument gibt an, dass HTTP-Antwort-Header ausgegeben werden sollen. Sie können diese Option verwenden, um Fehler zu identifizieren.

Wenn die Anforderung erfolgreich ist, gibt dieser Befehl eine Antwort ähnlich dem folgenden Beispiel aus.

```
{
  "GGGroups": [
    {
      "GGGroupId": "greengrassV2-coreDevice-MyGreengrassCore",
      "Cores": [
        {
          "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
          "Connectivity": [
            {
              "Id": "AUTOIP_192.168.1.4_1",
              "HostAddress": "192.168.1.5",
              "PortNumber": 8883,
              "Metadata": ""
            }
          ]
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----\ncert-contents\n-----END CERTIFICATE-----\n"
      ]
    }
  ]
}
```

Wenn der Befehl einen Fehler ausgibt, finden Sie weitere Informationen unter Behebung von Greengrass-Erkennungsproblemen.

Weiterleiten von MQTT-Nachrichten zwischen Client-Geräten und AWS IoT Core

Sie können MQTT-Nachrichten und andere Daten zwischen Client-Geräten und weiterleitenAWS IoT Core. Client-Geräte stellen eine Verbindung mit der MQTT-Brokerkomponente her, die auf dem Core-Gerät ausgeführt wird. Standardmäßig leiten -Core-Geräte keine MQTT-Nachrichten oder -Daten zwischen Client-Geräten und weiterAWS IoT Core. Client-Geräte können standardmäßig nur über MQTT miteinander kommunizieren.

Um MQTT-Nachrichten zwischen Client-Geräten und weiterzuleitenAWS IoT Core, konfigurieren Sie die MQTT-Bridge-Komponente so, dass sie Folgendes tut:

- Weiterleiten von Nachrichten von Client-Geräten an AWS IoT Core.
- Weiterleiten von Nachrichten von AWS IoT Core an Client-Geräte.

> **Note**
>
> Die MQTT-Brücke verwendet QoS 1 zum Veröffentlichen und Abonnieren von AWS IoT Core, auch wenn ein Client-Gerät QoS 0 zum Veröffentlichen und Abonnieren des lokalen MQTT-Brokers verwendet. Infolgedessen können Sie eine zusätzliche Latenz beobachten, wenn Sie MQTT-Nachrichten von Client-Geräten auf dem lokalen MQTT-Broker an weiterleitenAWS IoT Core. Weitere Informationen zur MQTT-Konfiguration auf -Core-Geräten finden Sie unter Konfigurieren Sie MQTT-Timeouts und Cache-Einstellungen.

Themen

Konfigurieren und Bereitstellen der MQTT-Bridge-Komponente

Die MQTT-Bridge-Komponente verwendet eine Liste von Themenzuordnungen, die jeweils eine Nachrichtenquelle und ein Nachrichtenziel angeben. Um Nachrichten zwischen Client-Geräten und weiterzuleitenAWS IoT Core, stellen Sie die MQTT-Bridge-Komponente bereit und geben Sie jedes Quell- und Zielthema in der Komponentenkonfiguration an.

Um die MQTT-Bridge-Komponente auf einem Core-Gerät oder einer Gruppe von Core-Geräten bereitzustellen, erstellen Sie eine Bereitstellung, die die `aws.greengrass.clientdevices.mqtt.Bridge` Komponente enthält. Geben Sie die Themenzuordnungen `mqttTopicMapping`in der Konfiguration der MQTT-Bridge-Komponente in der Bereitstellung an.

Im folgenden Beispiel wird eine Bereitstellung definiert, die die MQTT-Bridge-Komponente so konfiguriert, dass Nachrichten zu Themen weitergeleitet werden, die dem `clients/+/hello/world` Themenfilter von Client-Geräten an entsprechenAWS IoT Core. Das `merge` Konfigurationsupdate erfordert ein serialisiertes JSON-Objekt. Weitere Informationen finden Sie unter Komponentenkonfigurationen aktualisieren.

Console

```
{
  "mqttTopicMapping": {
    "HelloWorldIotCore": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

AWS CLI

```
{
  "components": {
    "aws.greengrass.clientdevices.mqtt.Bridge": {
      "version": "2.0.0",
      "configurationUpdate": {
        "merge": "{\"mqttTopicMapping\":{\"HelloWorldIotCore\":{\"topic\":\"clients/
+/hello/world\",\"source\":\"LocalMqtt\",\"target\":\"IotCore\"}}}"
      }
    }
    ...
  }
}
```

Weiterleiten von MQTT-Nachrichten

Um MQTT-Nachrichten zwischen Client-Geräten und weiterzuleitenAWS IoT Core, konfigurieren und stellen Sie die MQTT-Bridge-Komponente bereit und geben Sie die Themen an, die weitergeleitet werden sollen.

Example Beispiel: Weiterleiten von Nachrichten zu einem Thema von Client-Geräten an AWS IoT Core

Die folgende Konfiguration der MQTT-Bridge-Komponente legt die Weiterleitung von Nachrichten zu Themen fest, die dem `clients/+/hello/world/event` Themenfilter von Client-Geräten an entsprechenAWS IoT Core.

```
{
  "mqttTopicMapping": {
    "HelloWorldEvent": {
      "topic": "clients/+/hello/world/event",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

Example Beispiel: Weiterleiten von Nachrichten zu einem Thema von AWS IoT Core an Client-Geräte

Die folgende Konfiguration der MQTT-Bridge-Komponente legt die Weiterleitung von Nachrichten zu Themen fest, die dem `clients/+/hello/world/event/response` Themenfilter von AWS IoT Core an Client-Geräte entsprechen.

```
{
  "mqttTopicMapping": {
    "HelloWorldEventConfirmation": {
      "topic": "clients/+/hello/world/event/response",
      "source": "IotCore",
      "target": "LocalMqtt"
    }
  }
}
```

Interagieren mit Client-Geräten in Komponenten

Sie können benutzerdefinierte Greengrass-Komponenten entwickeln, die mit Clientgeräten interagieren, die mit einem Core-Gerät verbunden sind. Sie können beispielsweise Komponenten entwickeln, die Folgendes tun:

- Reagieren Sie auf MQTT-Nachrichten von Client-Geräten und senden Sie Daten an AWS Cloud Ziele.
- Senden Sie MQTT-Nachrichten an Client-Geräte, um Aktionen zu initiieren.

Client-Geräte stellen über die MQTT-Brokerkomponente, die auf dem Core-Gerät ausgeführt wird, eine Verbindung zu einem Core-Gerät her und kommunizieren mit diesem. Standardmäßig können Client-Geräte nur über MQTT miteinander kommunizieren, und Greengrass-Komponenten können diese MQTT-Nachrichten nicht empfangen oder Nachrichten an Client-Geräte senden.

Greengrass-Komponenten verwenden die lokale Publish/Subscribe-Schnittstelle, um auf einem Core-Gerät zu kommunizieren. Um mit Client-Geräten in Greengrass-Komponenten zu kommunizieren, konfigurieren Sie die MQTT-Bridge-Komponente so, dass sie Folgendes tut:

- Weiterleiten von MQTT-Nachrichten von Client-Geräten an die lokale Veröffentlichung/das lokale Abonnieren.
- Weiterleiten von MQTT-Nachrichten von der lokalen Veröffentlichung/dem Abonnieren von Client-Geräten.

Sie können auch mit Client-Geräteschatten in Greengrass-Komponenten interagieren. Weitere Informationen finden Sie unter Interagieren und Synchronisieren von Client-Geräteschatten.

Themen

Konfigurieren und Bereitstellen der MQTT-Bridge-Komponente

Die MQTT-Bridge-Komponente verwendet eine Liste von Themenzuordnungen, die jeweils eine Nachrichtenquelle und ein Nachrichtenziel angeben. Um mit Client-Geräten zu kommunizieren,

stellen Sie die MQTT-Bridge-Komponente bereit und geben Sie jedes Quell- und Zielthema in der Komponentenkonfiguration an.

Um die MQTT-Bridge-Komponente auf einem Core-Gerät oder einer Gruppe von Core-Geräten bereitzustellen, erstellen Sie eine Bereitstellung, die die `aws.greengrass.clientdevices.mqtt.Bridge` Komponente enthält. Geben Sie die Themenzuordnungen `mqttTopicMapping`in der Konfiguration der MQTT-Bridge-Komponente in der Bereitstellung an.

Das folgende Beispiel definiert eine Bereitstellung, die die MQTT-Bridge-Komponente so konfiguriert, dass das `clients/MyClientDevice1/hello/world` Thema von Client-Geräten an den lokalen Publish/Subscribe-Broker weitergeleitet wird. Das `merge` Konfigurationsupdate erfordert ein serialisiertes JSON-Objekt. Weitere Informationen finden Sie unter Komponentenkonfigurationen aktualisieren.

Console

```
{
  "mqttTopicMapping": {
    "HelloWorldPubsub": {
      "topic": "clients/MyClientDevice1/hello/world",
      "source": "LocalMqtt",
      "target": "Pubsub"
    }
  }
}
```

AWS CLI

```
{
  "components": {
    "aws.greengrass.clientdevices.mqtt.Bridge": {
      "version": "2.0.0",
      "configurationUpdate": {
        "merge": "\"mqttTopicMapping\":{\"HelloWorldPubsub\":{\"topic\":\"clients/
MyClientDevice1/hello/world\",\"source\":\"LocalMqtt\",\"target\":\"Pubsub\"}}}"
      }
    }
    ...
  }
}
```

Sie können MQTT-Themen-Platzhalter verwenden, um Nachrichten zu Themen weiterzuleiten, die einem Themenfilter entsprechen. Wenn Sie MQTT Bridge v2.2.0 oder höher verwenden, können Sie MQTT-Themen-Platzhalter in Themenfiltern verwenden, wenn der Quell-Broker lokale Veröffentlichung/Abonnement ist. Weitere Informationen finden Sie unter Konfiguration der MQTT-Bridge-Komponente .

Empfangen von MQTT-Nachrichten von Client-Geräten

Sie können die lokalen Themen zum Veröffentlichen/Abonnieren abonnieren, die Sie für die MQTT-Bridge-Komponente konfigurieren, um Nachrichten von Client-Geräten zu empfangen.

So empfangen Sie MQTT-Nachrichten von Client-Geräten in benutzerdefinierten Komponenten

1. Konfigurieren Sie die MQTT-Bridge-Komponente und stellen Sie sie bereit, um Nachrichten aus einem MQTT-Thema weiterzuleiten, in dem Client-Geräte zu einem lokalen Veröffentlichungs-/Abonnementthema veröffentlichen.
2. Verwenden Sie die lokale IPC-Schnittstelle zum Veröffentlichen/Abonnieren, um das Thema zu abonnieren, an das die MQTT-Brücke Nachrichten weiterleitet. Weitere Informationen finden Sie unter Lokale Nachrichten veröffentlichen/abonnieren und SubscribeToTopic.

Das Tutorial Client-Geräte verbinden und testen enthält einen Abschnitt, in dem Sie eine Komponente entwickeln, die Nachrichten von einem Client-Gerät abonniert. Weitere Informationen finden Sie unter Schritt 4: Entwickeln einer Komponente, die mit Client-Geräten kommuniziert.

Senden von MQTT-Nachrichten an Client-Geräte

Sie können in den lokalen Themen zum Veröffentlichen/Abonnieren veröffentlichen, die Sie für die MQTT-Bridge-Komponente konfigurieren, um Nachrichten an Client-Geräte zu senden.

So veröffentlichen Sie MQTT-Nachrichten auf Client-Geräten in benutzerdefinierten Komponenten

1. Konfigurieren Sie die MQTT-Bridge-Komponente und stellen Sie sie bereit, um Nachrichten aus einem lokalen Veröffentlichungs-/Abonnementthema an ein MQTT-Thema weiterzuleiten, in dem Client-Geräte abonnieren.
2. Verwenden Sie die lokale IPC-Schnittstelle zum Veröffentlichen/Abonnieren, um zu dem Thema zu veröffentlichen, an das die MQTT-Brücke Nachrichten weiterleitet. Weitere Informationen finden Sie unter Lokale Nachrichten veröffentlichen/abonnieren und PublishToTopic.

Interagieren und Synchronisieren von Client-Geräteschatten

Sie können die Shadow Manager-Komponente verwenden, um lokale Schatten zu verwalten, einschließlich Client-Geräteschatten. Sie können den Shadow Manager verwenden, um Folgendes zu tun:

- Interagieren Sie mit Client-Geräteschatten in Greengrass-Komponenten.
- Synchronisieren Sie Schatten von Client-Geräten mit AWS IoT Core.

> **Note**
>
> Die Shadow-Manager-Komponente synchronisiert AWS IoT Core standardmäßig keine Schatten mit . Sie müssen die Shadow Manager-Komponente konfigurieren, um anzugeben, welche Client-Geräteschatten synchronisiert werden sollen.

Themen

Voraussetzungen

Um mit Client-Geräteschatten zu interagieren und Client-Geräteschatten mit zu synchronisierenAWS IoT Core, muss ein Core-Gerät die folgenden Anforderungen erfüllen:

- Das Core-Gerät muss zusätzlich zu den Greengrass-Komponenten für die Client-Geräteunterstützung die folgenden Komponenten ausführen:
  - Greengrass-Kern v2.6.0 oder höher
  - Shadow Manager v2.2.0 oder höher
  - MQTT Bridge v2.2.0 oder höher
- Die Authentifizierungskomponente des Client-Geräts muss so konfiguriert sein, dass Client-Geräte über Geräteschattenthemen kommunizieren können.

Aktivieren der Kommunikation zwischen Shadow Manager und Client-Geräten

Standardmäßig verwaltet die Shadow Manager-Komponente keine Client-Geräteschatten. Um dieses Feature zu aktivieren, müssen Sie MQTT-Nachrichten zwischen Client-Geräten und der Shadow-Manager-Komponente weiterleiten. Client-Geräte verwenden MQTT-Nachrichten, um Geräteschattenaktualisierungen zu empfangen und zu senden. Die Shadow Manager-Komponente abonniert die lokale Greengrass-Publish/Subscribe-Schnittstelle, sodass Sie die MQTT-Bridge-Komponente so konfigurieren können, dass MQTT-Nachrichten an Geräteschattenthemen weitergeleitet werden.

Die MQTT-Bridge-Komponente verbraucht eine Liste von Themenzuordnungen, die jeweils eine Nachrichtenquelle und ein Nachrichtenziel angeben. Damit die Shadow-Manager-Komponente Client-Geräteschatten verwalten kann, stellen Sie die MQTT-Bridge-Komponente bereit und geben Sie die Schattenthemen für die Client-Geräteschatten an. Sie müssen die Brücke so konfigurieren, dass Nachrichten in beide Richtungen zwischen lokaler MQTT und lokaler Veröffentlichung/dem lokalen Abonnement weitergeleitet werden.

Um die MQTT-Bridge-Komponente auf einem Core-Gerät oder einer Gruppe von Core-Geräten bereitzustellen, erstellen Sie eine Bereitstellung, die die `aws.greengrass.clientdevices.mqtt.Bridge` Komponente enthält. Geben Sie die Themenzuordnungen `mqttTopicMapping`in der Konfiguration der MQTT-Bridge-Komponente in der Bereitstellung an.

Verwenden Sie die folgenden Beispiele, um die MQTT-Bridge-Komponente so zu konfigurieren, dass sie die Kommunikation zwischen Client-Geräten und der Shadow-Manager-Komponente ermöglicht.

> **Note**
>
> Sie können diese Konfigurationsbeispiele in der -AWS IoT GreengrassKonsole verwenden. Wenn Sie die AWS IoT Greengrass API verwenden, erfordert das `merge` Konfigurationsupdate ein serialisiertes JSON-Objekt, daher müssen Sie die folgenden JSON-Objekte in Zeichenfolgen serialisieren. Weitere Informationen finden Sie unter Komponentenkonfigurationen aktualisieren.

Example Beispiel: Verwalten aller Client-Geräteschatten

Das folgende Beispiel für eine MQTT-Bridge-Konfiguration ermöglicht es dem Shadow Manager, alle Schatten für alle Client-Geräte zu verwalten.

```
{
  "mqttTopicMapping": {
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/#",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Example Beispiel: Verwalten von Schatten für ein Client-Gerät

Das folgende Beispiel für eine MQTT-Bridge-Konfiguration ermöglicht es dem Shadow Manager, alle Schatten für ein Client-Gerät mit dem Namen zu verwalten`MyClientDevice`.

```
{
  "mqttTopicMapping": {
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/MyClientDevice/shadow/#",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/MyClientDevice/shadow/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Example Beispiel: Verwalten eines benannten Schattens für alle Client-Geräte

Das folgende Beispiel für eine MQTT-Bridge-Konfiguration ermöglicht es dem Shadow Manager, einen Schatten mit dem Namen `DeviceConfiguration` für alle Client-Geräte zu verwalten.

```
{
  "mqttTopicMapping": {
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/name/DeviceConfiguration/#",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/name/DeviceConfiguration/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Example Beispiel: Verwalten der unbenannten Schatten aller Client-Geräte

Das folgende Beispiel für eine MQTT-Bridge-Konfiguration ermöglicht es dem Shadow Manager, unbenannte Schatten, aber keine benannten Schatten, für alle Client-Geräte zu verwalten.

```
{
  "mqttTopicMapping": {
    "DeleteShadowLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/delete",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "DeleteShadowPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/delete/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    },
    "GetShadowLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/get",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "GetShadowPubsubToLocalMqtt": {
```

```
      "topic": "$aws/things/+/shadow/get/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    },
    "UpdateShadowLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/update",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "UpdateShadowPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/update/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Interagieren mit Client-Geräteschatten in Komponenten

Sie können benutzerdefinierte Komponenten entwickeln, die den lokalen Schattenservice verwenden, um die lokalen Schattendokumente von Client-Geräten zu lesen und zu ändern. Weitere Informationen finden Sie unter Interagieren mit Schatten in Komponenten.

Schatten von Client-Geräten mit synchronisieren AWS IoT Core

Sie können die Shadow Manager-Komponente so konfigurieren, dass lokale Client-Geräteschattenstatus mit synchronisiert werdenAWS IoT Core. Weitere Informationen finden Sie unter Lokale Geräteschatten mit synchronisieren AWS IoT Core.

Fehlerbehebung bei Clientgeräten

Verwenden Sie die Informationen und Lösungen zur Fehlerbehebung in diesem Abschnitt, um Probleme mit Greengrass-Clientgeräten und Client-Gerätekomponenten zu beheben.

Themen

- Probleme bei der Greengrass-Erkennung
- Verbindungsprobleme mit MQTT

Probleme bei der Greengrass-Erkennung

Verwenden Sie die folgenden Informationen, um Probleme mit der Greengrass-Erkennung zu beheben. Diese Probleme können auftreten, wenn Client-Geräte die Greengrass-Erkennungs-API verwenden, um ein Greengrass-Kerngerät zu identifizieren, mit dem sie eine Verbindung herstellen können.

Themen

Probleme mit der Greengrass-Erkennung (HTTP-API)

Verwenden Sie die folgenden Informationen, um Probleme mit der Greengrass-Erkennung zu beheben. Diese Fehler können auftreten, wenn Sie die Discovery-API mit cURL testen.

Themen

curl: (52) Empty reply from server

Dieser Fehler wird möglicherweise angezeigt, wenn Sie in der Anforderung ein inaktives AWS IoT Zertifikat angeben.

Überprüfen Sie, ob dem Client-Gerät ein Zertifikat angefügt ist und ob das Zertifikat aktiv ist. Weitere Informationen finden Sie unter Anfügen eines Objekts oder einer Richtlinie an ein Client-Zertifikat und Aktivieren oder Deaktivieren eines Client-Zertifikats im AWS IoT Core -Entwicklerhandbuch.

HTTP 403: {"message":null,"traceId":"a1b2c3d4-5678-90ab-cdef-11111EXAMPLE"}

Dieser Fehler wird möglicherweise angezeigt, wenn das Client-Gerät nicht berechtigt ist, `greengrass:Discover` für sich selbst aufzurufen.

Überprüfen Sie, ob das Zertifikat des Client-Geräts über eine Richtlinie verfügt, die zulässt`greengrass:Discover`. Für diese Berechtigung können Sie keine Objektrichtlinienvariablen (`iot:Connection.Thing.*`) im `Resource` Abschnitt verwenden. Weitere Informationen finden Sie unter Erkennungsauthentifizierung und -autorisierung.

HTTP 404: {"errorMessage":"The thing provided for discovery was not found"}

Dieser Fehler kann in den folgenden Fällen auftreten:

- Das Client-Gerät ist keinen Greengrass-Core-Geräten oder -AWS IoT Greengrass V1Gruppen zugeordnet.
- Keines der zugeordneten Greengrass-Core-Geräte oder -AWS IoT Greengrass V1Gruppen des Client-Geräts hat einen MQTT-Broker-Endpunkt.
- Keines der zugeordneten Greengrass-Core-Geräte des Client-Geräts führt die Authentifizierungskomponente des Client-Geräts aus.

Überprüfen Sie, ob das Client-Gerät dem Core-Gerät zugeordnet ist, mit dem es eine Verbindung herstellen soll. Überprüfen Sie dann, ob das Core-Gerät die Authentifizierungskomponente des Client-Geräts ausführt und mindestens einen MQTT-Broker-Endpunkt hat. Weitere Informationen finden Sie hier:

- Zuordnen von Client-Geräten
- Verwalten von -Core-Geräteendpunkten
- Konfigurieren der Cloud-Erkennung (Konsole)

Probleme bei der Greengrass-Erkennung (AWS IoT Device SDK v2 für Python)

Verwenden Sie die folgenden Informationen, um Probleme mit der Greengrass-Erkennung in vAWS IoT Device SDK2 für Python zu beheben.

Themen

- awscrt.exceptions.AwsCrtError: AWS_ERROR_HTTP_CONNECTION_CLOSED: The connection has closed or is closing.
- awsiot.greengrass_discovery.DiscoveryException: ('Error during discover call: response_code=403', 403)
- awsiot.greengrass_discovery.DiscoveryException: ('Error during discover call: response_code=404', 404)

awscrt.exceptions.AwsCrtError: AWS_ERROR_HTTP_CONNECTION_CLOSED: The connection has closed or is closing.

Dieser Fehler wird möglicherweise angezeigt, wenn Sie in der Anforderung ein inaktives AWS IoT Zertifikat angeben.

Überprüfen Sie, ob dem Client-Gerät ein Zertifikat angefügt ist und ob das Zertifikat aktiv ist. Weitere Informationen finden Sie unter Anfügen eines Objekts oder einer Richtlinie an ein Client-Zertifikat und Aktivieren oder Deaktivieren eines Client-Zertifikats im AWS IoT Core -Entwicklerhandbuch.

awsiot.greengrass_discovery.DiscoveryException: ('Error during discover call: response_code=403', 403)

Dieser Fehler wird möglicherweise angezeigt, wenn das Client-Gerät nicht berechtigt ist, `greengrass:Discover` für sich selbst aufzurufen.

Überprüfen Sie, ob das Zertifikat des Client-Geräts über eine Richtlinie verfügt, die zulässt`greengrass:Discover`. Für diese Berechtigung können Sie keine Objektrichtlinienvariablen (`iot:Connection.Thing.*`) im `Resource` Abschnitt verwenden. Weitere Informationen finden Sie unter Erkennungsauthentifizierung und -autorisierung.

awsiot.greengrass_discovery.DiscoveryException: ('Error during discover call: response_code=404', 404)

Dieser Fehler kann in den folgenden Fällen auftreten:

- Das Client-Gerät ist keinen Greengrass-Core-Geräten oder -AWS IoT Greengrass V1Gruppen zugeordnet.
- Keines der zugeordneten Greengrass-Core-Geräte oder -AWS IoT Greengrass V1Gruppen des Client-Geräts hat einen MQTT-Broker-Endpunkt.
- Keines der zugeordneten Greengrass-Core-Geräte des Client-Geräts führt die Authentifizierungskomponente des Client-Geräts aus.

Überprüfen Sie, ob das Client-Gerät dem Core-Gerät zugeordnet ist, mit dem es eine Verbindung herstellen soll. Überprüfen Sie dann, ob das Core-Gerät die Authentifizierungskomponente des Client-Geräts ausführt und mindestens einen MQTT-Broker-Endpunkt hat. Weitere Informationen finden Sie hier:

- Zuordnen von Client-Geräten

- Verwalten von -Core-Geräteendpunkten
- Konfigurieren der Cloud-Erkennung (Konsole)

Probleme bei der Greengrass-Erkennung (AWS IoT Device SDK v2 für C++)

Verwenden Sie die folgenden Informationen, um Probleme mit der Greengrass-Erkennung in AWS IoT Device SDK v2 für C++ zu beheben.

Themen

- aws-c-http: AWS_ERROR_HTTP_CONNECTION_CLOSED, The connection has closed or is closing.
- aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. (HTTP 403)
- aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. (HTTP 404)

aws-c-http: AWS_ERROR_HTTP_CONNECTION_CLOSED, The connection has closed or is closing.

Dieser Fehler wird möglicherweise angezeigt, wenn Sie in der Anforderung ein inaktives AWS IoT Zertifikat angeben.

Überprüfen Sie, ob dem Client-Gerät ein Zertifikat angefügt ist und ob das Zertifikat aktiv ist. Weitere Informationen finden Sie unter Anfügen eines Objekts oder einer Richtlinie an ein Client-Zertifikat und Aktivieren oder Deaktivieren eines Client-Zertifikats im AWS IoT Core -Entwicklerhandbuch.

aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. (HTTP 403)

Dieser Fehler wird möglicherweise angezeigt, wenn das Client-Gerät nicht berechtigt ist, `greengrass:Discover` für sich selbst aufzurufen.

Überprüfen Sie, ob das Zertifikat des Client-Geräts über eine Richtlinie verfügt, die zulässt`greengrass:Discover`. Für diese Berechtigung können Sie keine Objektrichtlinienvariablen (`iot:Connection.Thing.*`) im `Resource` Abschnitt verwenden. Weitere Informationen finden Sie unter Erkennungsauthentifizierung und -autorisierung.

aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. (HTTP 404)

Dieser Fehler kann in den folgenden Fällen auftreten:

- Das Client-Gerät ist keinen Greengrass-Core-Geräten oder -AWS IoT Greengrass V1Gruppen zugeordnet.

- Keines der zugeordneten Greengrass-Core-Geräte oder -AWS IoT Greengrass V1Gruppen des Client-Geräts hat einen MQTT-Broker-Endpunkt.
- Keines der zugeordneten Greengrass-Core-Geräte des Client-Geräts führt die Authentifizierungskomponente des Client-Geräts aus.

Überprüfen Sie, ob das Client-Gerät dem Core-Gerät zugeordnet ist, mit dem es eine Verbindung herstellen soll. Überprüfen Sie dann, ob das Core-Gerät die Authentifizierungskomponente des Client-Geräts ausführt und mindestens einen MQTT-Broker-Endpunkt hat. Weitere Informationen finden Sie hier:

- Zuordnen von Client-Geräten
- Verwalten von -Core-Geräteendpunkten
- Konfigurieren der Cloud-Erkennung (Konsole)

Probleme bei der Greengrass-Erkennung (AWS IoT Device SDK v2 für JavaScript)

Verwenden Sie die folgenden Informationen, um Probleme mit der Greengrass-Erkennung in der AWS IoT Device SDK v2 für zu JavaScriptbeheben.

Themen

- Error: aws-c-http: AWS_ERROR_HTTP_CONNECTION_CLOSED, The connection has closed or is closing.
- Error: Discovery failed (headers: [object Object]) { response_code: 403 }
- Error: Discovery failed (headers: [object Object]) { response_code: 404 }
- Error: Discovery failed (headers: [object Object])

Error: aws-c-http: AWS_ERROR_HTTP_CONNECTION_CLOSED, The connection has closed or is closing.

Dieser Fehler wird möglicherweise angezeigt, wenn Sie in der Anforderung ein inaktives AWS IoT Zertifikat angeben.

Überprüfen Sie, ob dem Client-Gerät ein Zertifikat angefügt ist und ob das Zertifikat aktiv ist. Weitere Informationen finden Sie unter Anfügen eines Objekts oder einer Richtlinie an ein Client-Zertifikat und Aktivieren oder Deaktivieren eines Client-Zertifikats im AWS IoT Core -Entwicklerhandbuch.

Error: Discovery failed (headers: [object Object]) { response_code: 403 }

Dieser Fehler wird möglicherweise angezeigt, wenn das Client-Gerät nicht berechtigt ist, `greengrass:Discover` für sich selbst aufzurufen.

Überprüfen Sie, ob das Zertifikat des Client-Geräts über eine Richtlinie verfügt, die zulässt`greengrass:Discover`. Für diese Berechtigung können Sie keine Objektrichtlinienvariablen (`iot:Connection.Thing.*`) im `Resource` Abschnitt verwenden. Weitere Informationen finden Sie unter Erkennungsauthentifizierung und -autorisierung.

Error: Discovery failed (headers: [object Object]) { response_code: 404 }

Dieser Fehler kann in den folgenden Fällen auftreten:

- Das Client-Gerät ist keinen Greengrass-Core-Geräten oder -AWS IoT Greengrass V1Gruppen zugeordnet.
- Keines der zugeordneten Greengrass-Core-Geräte oder -AWS IoT Greengrass V1Gruppen des Client-Geräts hat einen MQTT-Broker-Endpunkt.
- Keines der zugeordneten Greengrass-Core-Geräte des Client-Geräts führt die Authentifizierungskomponente des Client-Geräts aus.

Überprüfen Sie, ob das Client-Gerät dem Core-Gerät zugeordnet ist, mit dem es eine Verbindung herstellen soll. Überprüfen Sie dann, ob das Core-Gerät die Authentifizierungskomponente des Client-Geräts ausführt und mindestens einen MQTT-Broker-Endpunkt hat. Weitere Informationen finden Sie hier:

- Zuordnen von Client-Geräten
- Verwalten von -Core-Geräteendpunkten
- Konfigurieren der Cloud-Erkennung (Konsole)

Error: Discovery failed (headers: [object Object])

Dieser Fehler (ohne HTTP-Antwortcode) wird möglicherweise angezeigt, wenn Sie das Greengrass-Erkennungsbeispiel ausführen. Dieser Fehler kann aus mehreren Gründen auftreten.

- Dieser Fehler wird möglicherweise angezeigt, wenn das Client-Gerät nicht berechtigt ist, `greengrass:Discover` für sich selbst aufzurufen.

Überprüfen Sie, ob das Zertifikat des Client-Geräts über eine Richtlinie verfügt, die zulässt`greengrass:Discover`. Für diese Berechtigung können Sie keine Objektrichtlinienvariablen (`iot:Connection.Thing.*`) im `Resource` Abschnitt verwenden. Weitere Informationen finden Sie unter Erkennungsauthentifizierung und -autorisierung.

- Dieser Fehler kann in den folgenden Fällen auftreten:
  - Das Client-Gerät ist keinen Greengrass-Core-Geräten oder -AWS IoT Greengrass V1Gruppen zugeordnet.
  - Keines der zugeordneten Greengrass-Core-Geräte oder -AWS IoT Greengrass V1Gruppen des Client-Geräts hat einen MQTT-Broker-Endpunkt.
  - Keines der zugeordneten Greengrass-Core-Geräte des Client-Geräts führt die Authentifizierungskomponente des Client-Geräts aus.

  Überprüfen Sie, ob das Client-Gerät dem Core-Gerät zugeordnet ist, mit dem es eine Verbindung herstellen soll. Überprüfen Sie dann, ob das Core-Gerät die Authentifizierungskomponente des Client-Geräts ausführt und mindestens einen MQTT-Broker-Endpunkt hat. Weitere Informationen finden Sie hier:
  - Zuordnen von Client-Geräten
  - Verwalten von -Core-Geräteendpunkten
  - Konfigurieren der Cloud-Erkennung (Konsole)

Probleme bei der Greengrass-Erkennung (AWS IoT Device SDK v2 für Java)

Verwenden Sie die folgenden Informationen, um Probleme mit der Greengrass-Erkennung in vAWS IoT Device SDK2 für Java zu beheben.

Themen

- software.amazon.awssdk.crt.CrtRuntimeException: Error Getting Response Status Code from HttpStream. (aws_last_error: AWS_ERROR_HTTP_DATA_NOT_AVAILABLE(2062), This data is not yet available.)
- java.lang.RuntimeException: Error x-amzn-ErrorType(403)
- java.lang.RuntimeException: Error x-amzn-ErrorType(404)

software.amazon.awssdk.crt.CrtRuntimeException: Error Getting Response Status Code from HttpStream. (aws_last_error: AWS_ERROR_HTTP_DATA_NOT_AVAILABLE(2062), This data is not yet available.)

Dieser Fehler wird möglicherweise angezeigt, wenn Sie in der Anforderung ein inaktives AWS IoT Zertifikat angeben.

Überprüfen Sie, ob dem Client-Gerät ein Zertifikat angefügt ist und ob das Zertifikat aktiv ist. Weitere Informationen finden Sie unter Anfügen eines Objekts oder einer Richtlinie an ein Client-Zertifikat und Aktivieren oder Deaktivieren eines Client-Zertifikats im AWS IoT Core -Entwicklerhandbuch.

java.lang.RuntimeException: Error x-amzn-ErrorType(403)

Dieser Fehler wird möglicherweise angezeigt, wenn das Client-Gerät nicht berechtigt ist, `greengrass:Discover` für sich selbst aufzurufen.

Überprüfen Sie, ob das Zertifikat des Client-Geräts über eine Richtlinie verfügt, die zulässt`greengrass:Discover`. Für diese Berechtigung können Sie keine Objektrichtlinienvariablen (`iot:Connection.Thing.*`) im `Resource` Abschnitt verwenden. Weitere Informationen finden Sie unter Erkennungsauthentifizierung und -autorisierung.

java.lang.RuntimeException: Error x-amzn-ErrorType(404)

Dieser Fehler kann in den folgenden Fällen auftreten:

- Das Client-Gerät ist keinen Greengrass-Core-Geräten oder -AWS IoT Greengrass V1Gruppen zugeordnet.
- Keines der zugeordneten Greengrass-Core-Geräte oder -AWS IoT Greengrass V1Gruppen des Client-Geräts hat einen MQTT-Broker-Endpunkt.
- Keines der zugeordneten Greengrass-Core-Geräte des Client-Geräts führt die Authentifizierungskomponente des Client-Geräts aus.

Überprüfen Sie, ob das Client-Gerät dem Core-Gerät zugeordnet ist, mit dem es eine Verbindung herstellen soll. Überprüfen Sie dann, ob das Core-Gerät die Authentifizierungskomponente des Client-Geräts ausführt und mindestens einen MQTT-Broker-Endpunkt hat. Weitere Informationen finden Sie hier:

- Zuordnen von Client-Geräten
- Verwalten von -Core-Geräteendpunkten

- Konfigurieren der Cloud-Erkennung (Konsole)

Verbindungsprobleme mit MQTT

Verwenden Sie die folgenden Informationen, um Probleme mit MQTT-Verbindungen von Client-Geräten zu beheben. Diese Probleme können auftreten, wenn Client-Geräte versuchen, über MQTT eine Verbindung zu einem Core-Gerät herzustellen.

Themen

- io.moquette.broker.Authorizator: Client does not have read permissions on the topic
- MQTT-Verbindungsprobleme (Python)
- Verbindungsprobleme mit MQTT (C++)
- MQTT-Verbindungsprobleme (Java)
- Verbindungsprobleme mit MQTT (JavaScript)

io.moquette.broker.Authorizator: Client does not have read permissions on the topic

Dieser Fehler wird möglicherweise in den Greengrass-Protokollen angezeigt, wenn ein Client-Gerät versucht, ein MQTT-Thema zu abonnieren, bei dem es keine Berechtigung hat. Die Fehlermeldung enthält das Thema .

Überprüfen Sie, ob die Konfiguration der Authentifizierungskomponente des Client-Geräts Folgendes enthält:

- Eine Gerätegruppe, die dem Client-Gerät entspricht.
- Eine Client-Geräteautorisierungsrichtlinie für diese Gerätegruppe, die die `-mqtt:subscribe`Berechtigung für das Thema erteilt.

Weitere Informationen zum Bereitstellen und Konfigurieren der Authentifizierungskomponente für Client-Geräte finden Sie im Folgenden:

- Konfigurieren der Cloud-Erkennung (Konsole)
- Authentifizierung auf Client-Geräten
- Erstellen von Bereitstellungen

MQTT-Verbindungsprobleme (Python)

Verwenden Sie die folgenden Informationen, um Probleme mit MQTT-Verbindungen von Client-Geräten zu beheben, wenn Sie AWS IoT Device SDK v2 für Python verwenden.

Themen

AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred

Dieser Fehler wird möglicherweise angezeigt, wenn die Authentifizierungskomponente des Client-Geräts keine Client-Geräteautorisierungsrichtlinie definiert, die dem Client-Gerät die Berechtigung zum Herstellen einer Verbindung erteilt.

Überprüfen Sie, ob die Konfiguration der Client-Geräte-Authentifizierungskomponente Folgendes enthält:

- Eine Gerätegruppe, die dem Client-Gerät entspricht.
- Eine Client-Geräteautorisierungsrichtlinie für diese Gerätegruppe, die die `-mqtt:connect`Berechtigung für das Client-Gerät erteilt.

Weitere Informationen zum Bereitstellen und Konfigurieren der Authentifizierungskomponente für Client-Geräte finden Sie im Folgenden:

- Konfigurieren der Cloud-Erkennung (Konsole)
- Authentifizierung auf Client-Geräten
- Erstellen von Bereitstellungen

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

Dieser Fehler kann auftreten, wenn die Authentifizierungskomponente des Client-Geräts keine Client-Geräteautorisierungsrichtlinie definiert, die dem Client-Gerät die Berechtigung zum Herstellen einer Verbindung gewährt.

Überprüfen Sie, ob die Konfiguration der Client-Geräte-Authentifizierungskomponente Folgendes enthält:

- Eine Gerätegruppe, die dem Client-Gerät entspricht.
- Eine Client-Geräteautorisierungsrichtlinie für diese Gerätegruppe, die die `-mqtt:connect`Berechtigung für das Client-Gerät erteilt.

Weitere Informationen zum Bereitstellen und Konfigurieren der Authentifizierungskomponente für Client-Geräte finden Sie im Folgenden:

- Konfigurieren der Cloud-Erkennung (Konsole)
- Authentifizierung auf Client-Geräten
- Erstellen von Bereitstellungen

Verbindungsprobleme mit MQTT (C++)

Verwenden Sie die folgenden Informationen, um Probleme mit MQTT-Verbindungen des Client-Geräts zu beheben, wenn Sie AWS IoT Device SDK v2 für C++ verwenden.

Themen

- AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred
- AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred

Dieser Fehler wird möglicherweise angezeigt, wenn die Authentifizierungskomponente des Client-Geräts keine Client-Geräteautorisierungsrichtlinie definiert, die dem Client-Gerät die Berechtigung zum Herstellen einer Verbindung erteilt.

Überprüfen Sie, ob die Konfiguration der Client-Geräte-Authentifizierungskomponente Folgendes enthält:

- Eine Gerätegruppe, die dem Client-Gerät entspricht.
- Eine Client-Geräteautorisierungsrichtlinie für diese Gerätegruppe, die die `-mqtt:connect`Berechtigung für das Client-Gerät erteilt.

Weitere Informationen zum Bereitstellen und Konfigurieren der Authentifizierungskomponente für Client-Geräte finden Sie im Folgenden:

- Konfigurieren der Cloud-Erkennung (Konsole)
- Authentifizierung auf Client-Geräten
- Erstellen von Bereitstellungen

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

Dieser Fehler wird möglicherweise angezeigt, wenn die Authentifizierungskomponente des Client-Geräts keine Client-Geräteautorisierungsrichtlinie definiert, die dem Client-Gerät die Berechtigung zum Herstellen einer Verbindung erteilt.

Überprüfen Sie, ob die Konfiguration der Client-Geräte-Authentifizierungskomponente Folgendes enthält:

- Eine Gerätegruppe, die dem Client-Gerät entspricht.
- Eine Client-Geräteautorisierungsrichtlinie für diese Gerätegruppe, die die `-mqtt:connect`Berechtigung für das Client-Gerät erteilt.

Weitere Informationen zum Bereitstellen und Konfigurieren der Authentifizierungskomponente für Client-Geräte finden Sie im Folgenden:

- Konfigurieren der Cloud-Erkennung (Konsole)
- Authentifizierung auf Client-Geräten
- Erstellen von Bereitstellungen

MQTT-Verbindungsprobleme (Java)

Verwenden Sie die folgenden Informationen, um Probleme mit MQTT-Verbindungen von Client-Geräten zu beheben, wenn Sie AWS IoT Device SDK v2 für Java verwenden.

Themen

- software.amazon.awssdk.crt.mqtt.MqttException: Protocol error occurred
- AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

software.amazon.awssdk.crt.mqtt.MqttException: Protocol error occurred

Dieser Fehler wird möglicherweise angezeigt, wenn die Authentifizierungskomponente des Client-Geräts keine Client-Geräteautorisierungsrichtlinie definiert, die dem Client-Gerät die Berechtigung zum Herstellen einer Verbindung erteilt.

Überprüfen Sie, ob die Konfiguration der Client-Geräte-Authentifizierungskomponente Folgendes enthält:

- Eine Gerätegruppe, die dem Client-Gerät entspricht.
- Eine Client-Geräteautorisierungsrichtlinie für diese Gerätegruppe, die die `-mqtt:connect`Berechtigung für das Client-Gerät erteilt.

Weitere Informationen zum Bereitstellen und Konfigurieren der Authentifizierungskomponente für Client-Geräte finden Sie im Folgenden:

- Konfigurieren der Cloud-Erkennung (Konsole)
- Authentifizierung auf Client-Geräten
- Erstellen von Bereitstellungen

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

Dieser Fehler kann auftreten, wenn die Authentifizierungskomponente des Client-Geräts keine Client-Geräteautorisierungsrichtlinie definiert, die dem Client-Gerät die Berechtigung zum Herstellen einer Verbindung gewährt.

Überprüfen Sie, ob die Konfiguration der Client-Geräte-Authentifizierungskomponente Folgendes enthält:

- Eine Gerätegruppe, die dem Client-Gerät entspricht.
- Eine Client-Geräteautorisierungsrichtlinie für diese Gerätegruppe, die die `-mqtt:connect`Berechtigung für das Client-Gerät erteilt.

Weitere Informationen zum Bereitstellen und Konfigurieren der Authentifizierungskomponente für Client-Geräte finden Sie im Folgenden:

- Konfigurieren der Cloud-Erkennung (Konsole)
- Authentifizierung auf Client-Geräten

- Erstellen von Bereitstellungen

Verbindungsprobleme mit MQTT (JavaScript)

Verwenden Sie die folgenden Informationen, um Probleme mit MQTT-Verbindungen von Client-Geräten zu beheben, wenn Sie AWS IoT Device SDK v2 für JavaScript verwenden.

Themen

- AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred
- AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred

Dieser Fehler kann auftreten, wenn die Authentifizierungskomponente des Client-Geräts keine Client-Geräteautorisierungsrichtlinie definiert, die dem Client-Gerät die Berechtigung zum Herstellen einer Verbindung gewährt.

Überprüfen Sie, ob die Konfiguration der Client-Geräte-Authentifizierungskomponente Folgendes enthält:

- Eine Gerätegruppe, die dem Client-Gerät entspricht.
- Eine Client-Geräteautorisierungsrichtlinie für diese Gerätegruppe, die die `-mqtt:connect`Berechtigung für das Client-Gerät erteilt.

Weitere Informationen zum Bereitstellen und Konfigurieren der Authentifizierungskomponente für Client-Geräte finden Sie im Folgenden:

- Konfigurieren der Cloud-Erkennung (Konsole)
- Authentifizierung auf Client-Geräten
- Erstellen von Bereitstellungen

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

Dieser Fehler kann auftreten, wenn die Authentifizierungskomponente des Client-Geräts keine Client-Geräteautorisierungsrichtlinie definiert, die dem Client-Gerät die Berechtigung zum Herstellen einer Verbindung gewährt.

Überprüfen Sie, ob die Konfiguration der Client-Geräte-Authentifizierungskomponente Folgendes enthält:

- Eine Gerätegruppe, die dem Client-Gerät entspricht.
- Eine Client-Geräteautorisierungsrichtlinie für diese Gerätegruppe, die die `-mqtt:connect`Berechtigung für das Client-Gerät erteilt.

Weitere Informationen zum Bereitstellen und Konfigurieren der Authentifizierungskomponente für Client-Geräte finden Sie im Folgenden:

- Konfigurieren der Cloud-Erkennung (Konsole)
- Authentifizierung auf Client-Geräten
- Erstellen von Bereitstellungen

Interagieren mit Geräteschatten

Greengrass-Core-Geräte können mithilfe von AWS IoT Komponenten mit Geräteschatten interagieren. Ein Schatten ist ein JSON-Dokument, das die aktuellen oder gewünschten Statusinformationen für ein -AWS IoTObjekt speichert. Schatten können den Status eines Geräts anderen AWS IoT Greengrass Komponenten zur Verfügung stellen, unabhängig davon, ob das Gerät mit verbunden ist AWS IoT oder nicht. Jedes AWS IoT Gerät hat seinen eigenen klassischen, unbenannten Schatten. Sie können auch mehrere benannte Schatten für jedes Gerät erstellen.

Geräte und Services können Cloud-Schatten mithilfe von MQTT und den reservierten MQTT-Schattenthemen , HTTP mithilfe der Device Shadow REST API und der AWS CLI für AWS IoTerstellen, aktualisieren und löschen.

Die Shadow Manager-Komponente ermöglicht es Ihren Greengrass-Komponenten, lokale Schatten zu erstellen, zu aktualisieren und zu löschen, indem sie den lokalen Schattenservice und die lokalen Themen zum Veröffentlichen/Abonnieren von Schatten verwenden. Der Schattenmanager verwaltet auch die Speicherung dieser lokalen Schattendokumente auf Ihrem Core-Gerät und übernimmt die Synchronisation von Schattenstatusinformationen mit Cloud-Schatten.

Sie können die Shadow Manager-Komponente auch verwenden, um lokale Schatten für Client-Geräte zu verwalten, die eine Verbindung zum Core-Gerät herstellen. Damit der Shadow Manager Schatten von Client-Geräten verwalten kann, konfigurieren Sie die MQTT-Bridge-Komponente so, dass Nachrichten zwischen dem lokalen MQTT-Broker und dem lokalen Publish/Subscribe-Service weitergeleitet werden. Weitere Informationen finden Sie unter Interagieren und Synchronisieren von Client-Geräteschatten.

Weitere Informationen zu Konzepten für AWS IoT Geräteschatten finden Sie unter AWS IoT Device Shadow Service im AWS IoT Entwicklerhandbuch für .

Themen

Interagieren mit Schatten in Komponenten

Sie können benutzerdefinierte Komponenten entwickeln, einschließlich Lambda-Funktionskomponenten, die den lokalen Schattenservice verwenden, um lokale Schattendokumente und Schattendokumente für Client-Geräte zu lesen und zu ändern.

Benutzerdefinierte Komponenten interagieren mit dem lokalen Schattenservice mithilfe der AWS IoT Greengrass Core IPC-Bibliotheken in der AWS IoT Device SDK. Die Schattenmanager-Komponente aktiviert den lokalen Schattendienst auf Ihrem Core-Gerät.

Um die Shadow Manager-Komponente auf einem Greengrass-Kerngerät bereitzustellen, erstellen Sie eine Bereitstellung, die die `aws.greengrass.ShadowManager` Komponente enthält.

> **Note**
>
> Standardmäßig ermöglicht die Bereitstellung der Shadow Manager-Komponente nur lokale Schattenoperationen. Damit Schattenstatusinformationen für Kerngeräteschatten oder Schatten für Clientgeräte mit den entsprechenden Cloud-Schattendokumenten in AWS IoT Greengrass synchronisiert werden könnenAWS IoT Core, müssen Sie ein Konfigurationsupdate für die Shadow-Manager-Komponente erstellen, das den `-synchronize`Parameter enthält. Weitere Informationen finden Sie unter Lokale Geräteschatten mit synchronisieren AWS IoT Core.

Themen

Schattenstatus abrufen und ändern

Die Schatten- IPC-Operationen rufen Statusinformationen in lokalen Schattendokumenten ab und aktualisieren sie. Die Schattenmanager-Komponente übernimmt die Speicherung dieser Schattendokumente auf Ihrem Core-Gerät.

So ändern Sie den lokalen Schattenstatus

1. Fügen Sie dem Rezept für Ihre benutzerdefinierte Komponente Autorisierungsrichtlinien hinzu, damit die Komponente Nachrichten zu lokalen Schattenthemen empfangen kann.

Beispiele für Autorisierungsrichtlinien finden Sie unter Beispiele für lokale Schatten-IPK-Autorisierungsrichtlinien.

2. Verwenden Sie die Schatten- IPC-Operationen, um Schattenstatusinformationen abzurufen und zu ändern. Weitere Informationen zur Verwendung von Schatten-IPK-Operationen im Komponentencode finden Sie unter Interagieren mit lokalen Schatten.

> **Note**
>
> Damit ein Core-Gerät mit Client-Geräteschatten interagieren kann, müssen Sie auch die MQTT-Bridge-Komponente konfigurieren und bereitstellen. Weitere Informationen finden Sie unter Schattenmanager für die Kommunikation mit Client-Geräten aktivieren.

Reagieren Sie auf Änderungen des Schattenstatus

Greengrass-Komponenten verwenden die lokale Publish/Subscribe-Schnittstelle, um auf einem Core-Gerät zu kommunizieren. Damit eine benutzerdefinierte Komponente auf Änderungen des Schattenstatus reagieren kann, können Sie die lokalen Themen zum Veröffentlichen/Abonnieren abonnieren abonnieren. Auf diese Weise kann die Komponente Nachrichten zu den lokalen Schattenthemen empfangen und dann auf diese Nachrichten reagieren.

Lokale Schattenthemen verwenden dasselbe Format wie die MQTT-Themen des AWS IoT Geräteschattens. Weitere Informationen zu Schattenthemen finden Sie unter Geräteschatten-MQTT-Themen im AWS IoT -Entwicklerhandbuch.

So reagieren Sie auf Statusänderungen des lokalen Schattens

1. Fügen Sie dem Rezept für Ihre benutzerdefinierte Komponente Zugriffskontrollrichtlinien hinzu, damit die Komponente Nachrichten zu lokalen Schattenthemen empfangen kann.

   Beispiele für Autorisierungsrichtlinien finden Sie unter Beispiele für lokale Schatten-IPK-Autorisierungsrichtlinien.

2. Um eine benutzerdefinierte Aktion in einer Komponente zu initiieren, verwenden Sie `SubscribeToTopic` IPC-Operationen, um die Schattenthemen zu abonnieren, zu denen Sie Nachrichten empfangen möchten. Weitere Informationen zur Verwendung von lokalen IPC-Operationen zum Veröffentlichen/Abonnieren im Komponentencode finden Sie unter Lokale Nachrichten veröffentlichen/abonnieren.

3. Um eine Lambda-Funktion aufzurufen, verwenden Sie die Konfiguration der Ereignisquelle, um den Namen des Schattenthemas anzugeben und anzugeben, dass es sich um ein lokales Veröffentlichungs-/Abonnementthema handelt. Informationen zum Erstellen von Lambda-Funktionskomponenten finden Sie unter Ausführen von -AWS LambdaFunktionen.

> **Note**
>
> Damit ein Core-Gerät mit Client-Geräteschatten interagieren kann, müssen Sie auch die MQTT-Bridge-Komponente konfigurieren und bereitstellen. Weitere Informationen finden Sie unter Shadow Manager die Kommunikation mit Client-Geräten ermöglichen.

Lokale Geräteschatten mit synchronisieren AWS IoT Core

Die Schattenmanager-Komponente ermöglicht es AWS IoT Greengrass, lokale Geräteschattenzustände mit zu synchronisierenAWS IoT Core. Sie müssen die Konfiguration der Shadow-Manager-Komponente so ändern, dass sie den `synchronization` Konfigurationsparameter enthält, und die AWS IoT Objektnamen für Ihre Geräte und die Schatten angeben, die Sie synchronisieren möchten.

Wenn Sie Shadow Manager für die Synchronisierung von Schatten konfigurieren, synchronisiert er alle Statusänderungen für bestimmte Schatten, unabhängig davon, ob die Änderungen in lokalen Schattendokumenten oder in Cloud-Shadow-Dokumenten erfolgen.

Sie können auch angeben, ob die Shadow-Manager-Komponente Schatten in Echtzeit oder in einem regelmäßigen Intervall synchronisiert. Standardmäßig synchronisiert die Shadow-Manager-Komponente Schatten in Echtzeit, sodass das Core-Gerät Schattenaktualisierungen an und von sendet und empfängt, AWS IoT Core wenn jede Aktualisierung erfolgt. Sie können periodische Intervalle konfigurieren, um die Bandbreitennutzung und -gebühren zu reduzieren.

Themen

Voraussetzungen

Um lokale Schatten mit zu synchronisierenAWS IoT Core, müssen Sie die AWS IoT Richtlinie des Greengrass-Core-Geräts so konfigurieren, dass die folgenden AWS IoT Core Schattenrichtlinienaktionen zugelassen werden.

- `iot:GetThingShadow`
- `iot:UpdateThingShadow`
- `iot:DeleteThingShadow`

Weitere Informationen finden Sie hier:

- -AWS IoT CoreRichtlinienaktionen im -AWS IoTEntwicklerhandbuch
- Minimale AWS IoT Richtlinie für -AWS IoT Greengrass V2Core-Geräte
- Aktualisieren der AWS IoT Richtlinie eines Core-Geräts

Konfigurieren der Shadow Manager-Komponente

Der Schattenmanager benötigt eine Liste von Schattennamenzuordnungen, um Schattenstatusinformationen in lokalen Schattendokumenten mit Cloud-Shadow-Dokumenten in zu synchronisierenAWS IoT Core.

Um Schattenstatus zu synchronisieren, erstellen Sie eine Bereitstellung, die die `aws.greengrass.ShadowManager` Komponente enthält, und geben Sie die Schatten an, die Sie im `synchronize` Konfigurationsparameter in der Shadow-Manager-Konfiguration in der Bereitstellung synchronisieren möchten.

> **Note**
>
> Damit ein Core-Gerät mit Client-Geräteschatten interagieren kann, müssen Sie auch die MQTT-Bridge-Komponente konfigurieren und bereitstellen. Weitere Informationen finden Sie unter Schattenmanager für die Kommunikation mit Client-Geräten aktivieren.

Das folgende Beispielkonfigurationsupdate weist die Shadow Manager-Komponente an, die folgenden Schatten mit zu synchronisierenAWS IoT Core:

- Der klassische Schatten für das Core-Gerät
- Der benannte `MyCoreShadow` für das Core-Gerät
- Der klassische Schatten für ein IoT-Objekt mit dem Namen `MyDevice2`
- Die benannten Schatten `MyShadowA` und `MyShadowB` für ein IoT-Objekt mit dem Namen `MyDevice1`

Dieses Konfigurationsupdate gibt an, Schatten mit AWS IoT Core in Echtzeit zu synchronisieren. Wenn Sie Shadow Manager v2.1.0 oder höher verwenden, können Sie die Shadow-Manager-Komponente so konfigurieren, dass Schatten in einem regelmäßigen Intervall synchronisiert werden. Um dieses Feature zu konfigurieren, ändern Sie die Synchronisierungsstrategie in `periodic`und geben Sie einen `delay` in Sekunden für das Intervall an. Weitere Informationen finden Sie im Strategiekonfigurationsparameter der Shadow Manager-Komponente.

Dieses Konfigurationsupdate gibt an, Schatten in beiden Richtungen zwischen AWS IoT Core und dem Core-Gerät zu synchronisieren. Wenn Sie Shadow Manager v2.2.0 oder höher verwenden, können Sie die Shadow-Manager-Komponente so konfigurieren, dass Schatten nur in eine Richtung synchronisiert werden. Um dieses Feature zu konfigurieren, ändern Sie die Synchronisierung `direction` in `deviceToCloud` oder `cloudToDevice`. Weitere Informationen finden Sie im Konfigurationsparameter für die Richtung der Shadow Manager-Komponente.

```
{
  "strategy": {
    "type": "realTime"
  },
  "synchronize": {
    "coreThing": {
      "classic": true,
      "namedShadows": [
        "MyCoreShadow"
      ]
    },
    "shadowDocuments": [
      {
        "thingName": "MyDevice1",
        "classic": false,
        "namedShadows": [
          "MyShadowA",
          "MyShadowB"
        ]
```

```
      },
      {
        "thingName": "MyDevice2",
        "classic": true,
        "namedShadows": [ ]
      }
    ],
    "direction": "betweenDeviceAndCloud"
  }
}
```

Lokale Schatten synchronisieren

Wenn das Greengrass-Core-Gerät mit der AWS IoT Cloud verbunden ist, führt der Shadow Manager die folgenden Aufgaben für die Schatten aus, die Sie in der Komponentenkonfiguration angeben. Das Verhalten hängt von der Konfigurationsoption für die Schattensynchronisierungsrichtung ab, die Sie angeben. Standardmäßig verwendet der Shadow Manager die `betweenDeviceAndCloud` Option , um Schatten in beide Richtungen zu synchronisieren. Wenn Sie Shadow Manager v2.2.0 oder höher verwenden, können Sie das Core-Gerät so konfigurieren, dass Schatten nur in einer Richtung synchronisiert werden, was `cloudToDevice` oder sein kann`deviceToCloud`.

- Wenn die Konfiguration der Schattensynchronisierungsrichtung `betweenDeviceAndCloud` oder ist`cloudToDevice`, ruft der Schattenmanager die gemeldeten Statusinformationen aus dem Cloud-Schattendokument in abAWS IoT Core. Anschließend werden lokal gespeicherte Schattendokumente aktualisiert, um den Gerätestatus zu synchronisieren.
- Wenn die Konfiguration der Schattensynchronisierungsrichtung `betweenDeviceAndCloud` oder ist`deviceToCloud`, veröffentlicht Shadow Manager den aktuellen Status des Geräts im Cloud-Schattendokument.

Verhalten bei der Zusammenführung von Schattenkonflikten

In einigen Fällen, z. B. wenn das Core-Gerät vom Internet getrennt wird, kann sich ein Schatten im lokalen Schattenservice und in der AWS IoT Cloud ändern, bevor der Schattenmanager die Änderungen synchronisiert. Daher unterscheiden sich die gewünschten und gemeldeten Zustände zwischen dem lokalen Schattendienst und der AWS IoT Cloud

Wenn der Schattenmanager den Schatten synchronisiert, führt er die Änderungen gemäß dem folgenden Verhalten zusammen:

- Wenn Sie eine Version von Shadow Manager vor v2.2.0 verwenden oder die `betweenDeviceAndCloud` Schattensynchronisierungsrichtung angeben, gilt das folgende Verhalten:
  - Wenn ein Zusammenführungskonflikt im gewünschten Status eines Schattens vorliegt, überschreibt der Schattenmanager den widersprüchlichen Abschnitt des lokalen Schattendokuments mit dem Wert aus der AWS IoT Cloud.
  - Wenn ein Zusammenführungskonflikt im gemeldeten Status eines Schattens vorliegt, überschreibt der Schattenmanager den widersprüchlichen Abschnitt des Schattens in der AWS IoT Cloud mit dem Wert aus dem lokalen Schattendokument.
- Wenn Sie die Richtung der `deviceToCloud` Schattensynchronisierung angeben, überschreibt der Schattenmanager den widersprüchlichen Abschnitt des Schattens in der AWS IoT Cloud mit dem Wert aus dem lokalen Schattendokument.
- Wenn Sie die Richtung der `cloudToDevice` Schattensynchronisierung angeben, überschreibt der Schattenmanager den widersprüchlichen Abschnitt des lokalen Schattendokuments mit dem Wert aus der AWS IoT Cloud.

Verwalten von Datenströmen auf Greengrass-Core-Geräten

AWS IoT Greengrass Stream Manager macht es effizienter und zuverlässiger, IoT-Daten mit hohem Volumen in die zu übertragenAWS Cloud. Stream Manager verarbeitet Datenströme auf dem AWS IoT Greengrass Core, bevor sie in den exportiert werdenAWS Cloud. Stream Manager lässt sich in gängige Edge-Szenarien integrieren, z. B. in Machine Learning (ML)-Inferenz, in denen das AWS IoT Greengrass Core-Gerät Daten verarbeitet und analysiert, bevor es die Daten in die AWS Cloud oder lokale Speicherziele exportiert.

Stream Manager bietet eine gemeinsame Schnittstelle, um die Entwicklung benutzerdefinierter Komponenten zu vereinfachen, sodass Sie keine benutzerdefinierten Stream-Verwaltungsfunktionen erstellen müssen. Ihre Komponenten können einen standardisierten Mechanismus verwenden, um Streams mit hohem Volumen zu verarbeiten und lokale Datenaufbewahrungsrichtlinien zu verwalten. Sie können Richtlinien für Speichertyp, Größe und Datenaufbewahrung für jeden Stream definieren, um zu steuern, wie der Stream-Manager Daten verarbeitet und exportiert.

Stream Manager funktioniert in Umgebungen mit intermittierender oder eingeschränkter Konnektivität. Sie können die Bandbreitennutzung, das Timeout-Verhalten und die Art und Weise definieren, wie der AWS IoT Greengrass Core Stream-Daten verarbeitet, wenn er verbunden oder getrennt ist. Sie können auch Prioritäten festlegen, um die Reihenfolge zu steuern, in der der AWS IoT Greengrass Core Streams in den exportiertAWS Cloud. Auf diese Weise können Sie kritische Daten früher als andere Daten verarbeiten.

Sie können Stream Manager so konfigurieren, dass Daten zur Speicherung oder Weiterverarbeitung und Analyse automatisch AWS Cloud in den exportiert werden. Stream Manager unterstützt Exporte an die folgenden AWS Cloud Ziele:

- Kanäle in AWS IoT Analytics. AWS IoT Analytics ermöglicht Ihnen eine erweiterte Analyse Ihrer Daten, um Geschäftsentscheidungen zu treffen und Machine-Learning-Modelle zu verbessern. Weitere Informationen finden Sie unter Was ist AWS IoT Analytics? im AWS IoT Analytics-Benutzerhandbuch.
- Streams in Amazon Kinesis Data Streams. Sie können Kinesis Data Streams verwenden, um Daten mit hohem Volumen zu aggregieren und sie in ein Data Warehouse oder einen MapReduce Cluster zu laden. Weitere Informationen finden Sie unter Was ist Amazon Kinesis Data Streams? im Entwicklerhandbuch für Amazon Kinesis Data Streams.

- Asset-Eigenschaften in AWS IoT SiteWise. AWS IoT SiteWise Mit können Sie Daten von Industrieanlagen erfassen, organisieren und analysieren. Weitere Informationen finden Sie unter Was ist AWS IoT SiteWise? im AWS IoT SiteWise-Benutzerhandbuch.
- Objekte in Amazon Simple Storage Service Amazon S3. Sie können Amazon S3 verwenden, um große Datenmengen zu speichern und abzurufen. Weitere Informationen finden Sie unter Was ist Amazon S3? im Amazon Simple Storage Service-Entwicklerhandbuch.

Stream-Management-Workflow

Ihre IoT-Anwendungen interagieren über das Stream Manager SDK mit dem Stream Manager.

In einem einfachen Workflow verbraucht eine Komponente auf dem AWS IoT Greengrass Core IoT-Daten, wie Zeitreihentemperatur- und Druckmetriken. Die Komponente filtert oder komprimiert die Daten und ruft dann das Stream Manager SDK auf, um die Daten in einen Stream im Stream Manager zu schreiben. Stream Manager kann den Stream basierend auf den Richtlinien, die Sie für den Stream definieren, AWS Cloud automatisch in den exportieren. Komponenten können Daten auch direkt an lokale Datenbanken oder Speicher-Repositorys senden.

Ihre IoT-Anwendungen können mehrere benutzerdefinierte Komponenten enthalten, die Streams lesen oder in sie schreiben. Diese Komponenten können Lese- und Schreibvorgänge in Streams ausführen, um Daten auf dem AWS IoT Greengrass Core-Gerät zu filtern, zu aggregieren und zu analysieren. Dadurch ist es möglich, schnell auf lokale Ereignisse zu reagieren und wertvolle Informationen zu extrahieren, bevor die Datenübertragungen vom Kern zum lokalen Ziel AWS Cloud oder erfolgen.

Stellen Sie zunächst die Stream-Manager-Komponente auf Ihrem AWS IoT Greengrass Core-Gerät bereit. Konfigurieren Sie in der Bereitstellung die Stream-Manager-Komponentenparameter, um Einstellungen zu definieren, die für alle Streams auf dem Greengrass-Kerngerät gelten. Verwenden Sie diese Parameter, um zu steuern, wie Stream Manager Streams basierend auf Ihren Geschäftsanforderungen und Umgebungseinschränkungen speichert, verarbeitet und exportiert.

Nachdem Sie den Stream-Manager konfiguriert haben, können Sie Ihre IoT-Anwendungen erstellen und bereitstellen. Dies sind in der Regel benutzerdefinierte Komponenten, die `StreamManagerClient` im Stream Manager SDK verwenden, um Streams zu erstellen und mit ihnen zu interagieren. Wenn Sie einen Stream erstellen, können Sie Richtlinien pro Stream definieren, z. B. Exportziele, Priorität und Persistenz.

Voraussetzungen

Für die Verwendung von Stream Manager gelten die folgenden Anforderungen:

- Stream Manager benötigt zusätzlich zur AWS IoT Greengrass Core-Software mindestens 70 MB RAM. Ihr gesamter Speicherbedarf hängt von Ihrer Arbeitslast ab.
- AWS IoT Greengrass -Komponenten müssen das Stream Manager SDK verwenden, um mit Stream Manager zu interagieren. Das Stream Manager SDK ist in den folgenden Sprachen verfügbar:
  - Stream Manager SDK for Java (v1.1.0 oder höher)
  - Stream Manager SDK for Node.js (v1.1.0 oder höher)
  - Stream Manager SDK für Python (v1.1.0 oder höher)
- AWS IoT Greengrass -Komponenten müssen die Stream-Manager-Komponente (`aws.greengrass.StreamManager`) als Abhängigkeit in ihrem Rezept angeben, um Stream-Manager verwenden zu können.

  > **Note**
  >
  > Wenn Sie Stream Manager verwenden, um Daten in die Cloud zu exportieren, können Sie Version 2.0.7 der Stream Manager-Komponente nicht auf eine Version zwischen v2.0.8 und v2.0.11 aktualisieren. Wenn Sie Stream Manager zum ersten Mal bereitstellen, empfehlen wir dringend, die neueste Version der Stream Manager-Komponente bereitzustellen.

- Wenn Sie AWS Cloud Exportziele für einen Stream definieren, müssen Sie Ihre Exportziele erstellen und Zugriffsberechtigungen in der Greengrass-Geräterolle erteilen. Je nach Ziel können auch andere Anforderungen gelten. Weitere Informationen finden Sie hier:
  - the section called “AWS IoT Analytics-Kanäle”
  - the section called “Amazon Kinesis-Datenströme”
  - the section called “AWS IoT SiteWise Komponenteneigenschaften”
  - the section called “Amazon S3-Objekte”

  Sie sind für die Wartung dieser AWS Cloud Ressourcen verantwortlich.

Datensicherheit

Beachten Sie bei der Verwendung des Stream-Managers die folgenden Sicherheitsüberlegungen.

Lokale Datensicherheit

AWS IoT Greengrass verschlüsselt keine Stream-Daten im Ruhezustand oder während der Übertragung zwischen lokalen Komponenten auf dem Core-Gerät.

- Daten im Ruhezustand. Streamdaten werden lokal in einem Speicherverzeichnis gespeichert. Aus Gründen der Datensicherheit stützt AWS IoT Greengrass sich auf Dateiberechtigungen und vollständige Datenträgerverschlüsselung, falls 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 werden lokal übertragen. verschlüsselt AWS IoT Greengrass keine Stream-Daten während der lokalen Übertragung zwischen Datenquellen, AWS IoT Greengrass Komponenten, dem Stream-Manager-SDK und dem Stream-Manager.
- Daten während der Übertragung zum AWS Cloud. Datenströme, die vom Stream-Manager in exportiert wurden, AWS Cloud verwenden die StandardAWS-Service-Client-Verschlüsselung mit Transport Layer Security (TLS).

Client-Authentifizierung

Stream-Manager-Clients verwenden das Stream-Manager-SDK, um mit dem Stream-Manager zu kommunizieren. Wenn die Client-Authentifizierung aktiviert ist, können nur Greengrass-Komponenten mit Streams im Stream-Manager interagieren. Wenn die Client-Authentifizierung deaktiviert ist, kann jeder Prozess, der auf dem Greengrass-Core-Gerät ausgeführt wird, 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 konfigurieren, wenn Sie die Stream-Manager-Komponente auf -Core-Geräten bereitstellen.

	Enabled	Disabled
Parameterwert	`true` (Standard und empfohlen)	`false`
Zulässige Clients	Greengrass-Komponenten auf dem Core-Gerät	Greengrass-Komponenten auf dem Core-Gerät 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 für die Arbeit mit Streams”
- the section called “Exportkonfigurationen für unterstützte Cloud-Ziele”

Erstellen Sie benutzerdefinierte Komponenten, die Stream Manager verwenden

Verwenden Sie Stream Manager in benutzerdefinierten Greengrass-Komponenten, um IoT-Gerätedaten zu speichern, zu verarbeiten und zu exportieren. Verwenden Sie die Verfahren und Beispiele in diesem Abschnitt, um Komponentenrezepte, Artefakte und Anwendungen zu erstellen, die mit dem Stream Manager funktionieren. Weitere Informationen zum Entwickeln und Testen von Komponenten finden Sie unterErstellen von AWS IoT Greengrass Komponenten.

Themen

- Definieren Sie Komponentenrezepte, die den Stream-Manager verwenden
- Stellen Sie im Anwendungscode eine Connect zum Stream Manager her

Definieren Sie Komponentenrezepte, die den Stream-Manager verwenden

Um den Stream-Manager in einer benutzerdefinierten Komponente zu verwenden, müssen Sie die `aws.greengrass.StreamManager` Komponente als Abhängigkeit definieren. Sie müssen auch das Stream Manager SDK bereitstellen. Führen Sie die folgenden Aufgaben aus, um das Stream Manager SDK in der Sprache Ihrer Wahl herunterzuladen und zu verwenden.

Verwenden Sie das Stream Manager SDK for Java

Das Stream Manager SDK for Java ist als JAR-Datei verfügbar, mit der Sie Ihre Komponente kompilieren können. Anschließend können Sie eine Anwendungs-JAR erstellen, die das Stream Manager-SDK enthält, die Anwendungs-JAR als Komponentenartefakt definieren und die Anwendungs-JAR im Komponentenlebenszyklus ausführen.

So verwenden Sie das Stream Manager SDK for Java

1. Laden Sie die Stream Manager SDK for Java JAR-Datei herunter.
2. Gehen Sie wie folgt vor, um Komponentenartefakte aus Ihrer Java-Anwendung und der Stream Manager SDK-JAR-Datei zu erstellen:
   - Erstellen Sie Ihre Anwendung als JAR-Datei, die das Stream Manager-SDK-JAR enthält, und führen Sie diese JAR-Datei in Ihrem Komponentenrezept aus.
   - Definieren Sie das Stream Manager SDK JAR als Komponentenartefakt. Fügen Sie dieses Artefakt dem Klassenpfad hinzu, wenn Sie Ihre Anwendung in Ihrem Komponentenrezept ausführen.

   Ihr Komponentenrezept könnte wie das folgende Beispiel aussehen. Diese Komponente führt eine modifizierte Version des Beispiels StreamManagerS3.java aus, das das Stream Manager SDK JAR `StreamManagerS3.jar` enthält.

   JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.StreamManagerS3Java",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses stream manager to upload a file to an S3
 bucket.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.StreamManager": {
```

```
      "VersionRequirement": "^2.0.0"
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "java -jar {artifacts:path}/StreamManagerS3.jar"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Java/1.0.0/StreamManagerS3.jar"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.StreamManagerS3Java
ComponentVersion: 1.0.0
ComponentDescription: Uses stream manager to upload a file to an S3 bucket.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.StreamManager:
    VersionRequirement: "^2.0.0"
Manifests:
  - Lifecycle:
      run: java -jar {artifacts:path}/StreamManagerS3.jar
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Java/1.0.0/StreamManagerS3.jar
```

Weitere Informationen zum Entwickeln und Testen von Komponenten finden Sie unterErstellen von AWS IoT Greengrass Komponenten.

Verwenden Sie das Stream Manager SDK für Python

Das Stream Manager SDK für Python ist als Quellcode verfügbar, den Sie in Ihre Komponente aufnehmen können. Erstellen Sie eine ZIP-Datei des Stream Manager SDK, definieren Sie die ZIP-Datei als Komponentenartefakt und installieren Sie die Anforderungen des SDK im Komponentenlebenszyklus.

So verwenden Sie das Stream Manager SDK für Python

1. Klonen Sie das aws-greengrass-stream-manager-sdk-python-Repository oder laden Sie es herunter.

   ```
   git clone git@github.com:aws-greengrass/aws-greengrass-stream-manager-sdk-python.git
   ```

2. Erstellen Sie eine ZIP-Datei, die den `stream_manager` Ordner enthält, der den Quellcode des Stream Manager SDK für Python enthält. Sie können diese ZIP-Datei als Komponentenartefakt bereitstellen, das die AWS IoT Greengrass Core-Software bei der Installation Ihrer Komponente entpackt. Gehen Sie wie folgt vor:

   a. Öffnen Sie den Ordner, der das Repository enthält, das Sie im vorherigen Schritt geklont oder heruntergeladen haben.

      ```
      cd aws-greengrass-stream-manager-sdk-python
      ```

   b. Komprimieren `stream_manager` Sie den Ordner in eine ZIP-Datei mit dem Namen`stream_manager_sdk.zip`.

      Linux or Unix

      ```
      zip -rv stream_manager_sdk.zip stream_manager
      ```

      Windows Command Prompt (CMD)

      ```
      tar -acvf stream_manager_sdk.zip stream_manager
      ```

      PowerShell

      ```
      Compress-Archive stream_manager stream_manager_sdk.zip
      ```

c. Stellen Sie sicher, dass die `stream_manager_sdk.zip` Datei den `stream_manager` Ordner und seinen Inhalt enthält. Führen Sie den folgenden Befehl aus, um den Inhalt der ZIP-Datei aufzulisten.

   Linux or Unix

   ```
   unzip -l stream_manager_sdk.zip
   ```

   Windows Command Prompt (CMD)

   ```
   tar -tf stream_manager_sdk.zip
   ```

   Die Ausgabe sollte in etwa folgendermaßen aussehen:

   ```
   Archive:  aws-greengrass-stream-manager-sdk-python/stream_manager.zip
     Length      Date    Time    Name
   ---------  ---------- -----   ----
           0  02-24-2021 20:45   stream_manager/
         913  02-24-2021 20:45   stream_manager/__init__.py
        9719  02-24-2021 20:45   stream_manager/utilinternal.py
        1412  02-24-2021 20:45   stream_manager/exceptions.py
        1004  02-24-2021 20:45   stream_manager/util.py
           0  02-24-2021 20:45   stream_manager/data/
      254463  02-24-2021 20:45   stream_manager/data/__init__.py
       26515  02-24-2021 20:45   stream_manager/streammanagerclient.py
   ---------                     -------
      294026                     8 files
   ```

3. Kopieren Sie die Stream Manager SDK-Artefakte in den Artefaktordner Ihrer Komponente. Zusätzlich zur Stream Manager SDK-ZIP-Datei verwendet Ihre Komponente die `requirements.txt` SDK-Datei, um die Abhängigkeiten des Stream Manager SDK zu installieren. Ersetzen Sie *`~/greengrass-components`* durch den Pfad zu dem Ordner, den Sie für die lokale Entwicklung verwenden.

   Linux or Unix

   ```
   cp {stream_manager_sdk.zip,requirements.txt} ~/greengrass-components/artifacts/
   com.example.StreamManagerS3Python/1.0.0/
   ```

Windows Command Prompt (CMD)

```
robocopy . %USERPROFILE%\greengrass-components\artifacts
\com.example.StreamManagerS3Python\1.0.0 stream_manager_sdk.zip
robocopy . %USERPROFILE%\greengrass-components\artifacts
\com.example.StreamManagerS3Python\1.0.0 requirements.txt
```

PowerShell

```
cp .\stream_manager_sdk.zip,.\requirements.txt ~\greengrass-components\artifacts
\com.example.StreamManagerS3Python\1.0.0\
```

4. Erstellen Sie Ihr Komponentenrezept. Gehen Sie im Rezept wie folgt vor:

   a. Definieren Sie `stream_manager_sdk.zip` und `requirements.txt` als Artefakte.

   b. Definieren Sie Ihre Python-Anwendung als Artefakt.

   c. Installieren Sie im Installationszyklus die Stream Manager SDK-Anforderungen von`requirements.txt`.

   d. Fügen Sie im Run-Lebenszyklus das Stream Manager-SDK an Ihre Python-Anwendung an PYTHONPATH und führen Sie sie aus.

   Ihr Komponentenrezept könnte wie das folgende Beispiel aussehen. Diese Komponente führt das Beispiel stream_manager_s3.py aus.

   JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.StreamManagerS3Python",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses stream manager to upload a file to an S3
 bucket.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.StreamManager": {
      "VersionRequirement": "^2.0.0"
    }
  },
  "Manifests": [
```

```
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "pip3 install --user -r {artifacts:path}/requirements.txt",
        "run": "export PYTHONPATH=$PYTHONPATH:{artifacts:decompressedPath}/
stream_manager_sdk; python3 {artifacts:path}/stream_manager_s3.py"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip",
          "Unarchive": "ZIP"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "pip3 install --user -r {artifacts:path}/requirements.txt",
        "run": "set \"PYTHONPATH=%PYTHONPATH%;{artifacts:decompressedPath}/
stream_manager_sdk\" & py -3 {artifacts:path}/stream_manager_s3.py"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip",
          "Unarchive": "ZIP"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py"
        },
```

```
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.StreamManagerS3Python
ComponentVersion: 1.0.0
ComponentDescription: Uses stream manager to upload a file to an S3 bucket.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.StreamManager:
    VersionRequirement: "^2.0.0"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: pip3 install --user -r {artifacts:path}/requirements.txt
      run: |
        export PYTHONPATH=$PYTHONPATH:{artifacts:decompressedPath}/
stream_manager_sdk
        python3 {artifacts:path}/stream_manager_s3.py
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip
        Unarchive: ZIP
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt
  - Platform:
      os: windows
    Lifecycle:
      install: pip3 install --user -r {artifacts:path}/requirements.txt
      run: |
```

```
        set "PYTHONPATH=%PYTHONPATH%;{artifacts:decompressedPath}/
stream_manager_sdk"
        py -3 {artifacts:path}/stream_manager_s3.py
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip
        Unarchive: ZIP
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt
```

Weitere Informationen zum Entwickeln und Testen von Komponenten finden Sie unterErstellen von AWS IoT Greengrass Komponenten.

Verwenden Sie das Stream Manager SDK für JavaScript

Das Stream Manager SDK für JavaScript ist als Quellcode verfügbar, den Sie in Ihre Komponente aufnehmen können. Erstellen Sie eine ZIP-Datei des Stream Manager SDK, definieren Sie die ZIP-Datei als Komponentenartefakt und installieren Sie das SDK im Komponentenlebenszyklus.

Um das Stream Manager SDK zu verwenden für JavaScript

1. Klonen Sie das aws-greengrass-stream-manager-sdk-js-Repository oder laden Sie es herunter.

   ```
   git clone git@github.com:aws-greengrass/aws-greengrass-stream-manager-sdk-js.git
   ```

2. Erstellen Sie eine ZIP-Datei, die den `aws-greengrass-stream-manager-sdk` Ordner enthält, der den Quellcode des Stream Manager-SDK für enthält. JavaScript Sie können diese ZIP-Datei als Komponentenartefakt bereitstellen, das die AWS IoT Greengrass Core-Software bei der Installation Ihrer Komponente entpackt. Gehen Sie wie folgt vor:

   a. Öffnen Sie den Ordner, der das Repository enthält, das Sie im vorherigen Schritt geklont oder heruntergeladen haben.

      ```
      cd aws-greengrass-stream-manager-sdk-js
      ```

   b. Komprimieren `aws-greengrass-stream-manager-sdk` Sie den Ordner in eine ZIP-Datei mit dem Namen`stream-manager-sdk.zip`.

Linux or Unix

```
zip -rv stream-manager-sdk.zip aws-greengrass-stream-manager-sdk
```

Windows Command Prompt (CMD)

```
tar -acvf stream-manager-sdk.zip aws-greengrass-stream-manager-sdk
```

PowerShell

```
Compress-Archive aws-greengrass-stream-manager-sdk stream-manager-sdk.zip
```

c. Stellen Sie sicher, dass die `stream-manager-sdk.zip` Datei den `aws-greengrass-stream-manager-sdk` Ordner und seinen Inhalt enthält. Führen Sie den folgenden Befehl aus, um den Inhalt der ZIP-Datei aufzulisten.

Linux or Unix

```
unzip -l stream-manager-sdk.zip
```

Windows Command Prompt (CMD)

```
tar -tf stream-manager-sdk.zip
```

Die Ausgabe sollte in etwa folgendermaßen aussehen:

```
Archive:  stream-manager-sdk.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
        0  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/
      369  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/package.json
     1017  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/util.js
     8374  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/utilInternal.js
     1937  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/exceptions.js
        0  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/data/
   353343  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/data/index.js
    22599  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/client.js
      216  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/index.js
---------                     -------
```

```
  387855                    9 files
```

3. Kopieren Sie das Stream Manager SDK-Artefakt in den Artefaktordner Ihrer Komponente. Ersetzen Sie *~/greengrass-components* durch den Pfad zu dem Ordner, den Sie für die lokale Entwicklung verwenden.

   Linux or Unix

   ```
   cp stream-manager-sdk.zip ~/greengrass-components/artifacts/
   com.example.StreamManagerS3JS/1.0.0/
   ```

   Windows Command Prompt (CMD)

   ```
   robocopy . %USERPROFILE%\greengrass-components\artifacts
   \com.example.StreamManagerS3JS\1.0.0 stream-manager-sdk.zip
   ```

   PowerShell

   ```
   cp .\stream-manager-sdk.zip ~\greengrass-components\artifacts
   \com.example.StreamManagerS3JS\1.0.0\
   ```

4. Erstellen Sie Ihr Komponentenrezept. Gehen Sie im Rezept wie folgt vor:

   a. Definiere `stream-manager-sdk.zip` als Artefakt.

   b. Definieren Sie Ihre JavaScript Anwendung als Artefakt.

   c. Installieren Sie im Installationszyklus das Stream Manager-SDK vom `stream-manager-sdk.zip` Artefakt aus. `npm install`Mit diesem Befehl wird ein `node_modules` Ordner erstellt, der das Stream Manager-SDK und seine Abhängigkeiten enthält.

   d. Fügen Sie im Run-Lebenszyklus den `node_modules` Ordner an `NODE_PATH` und führen Sie Ihre JavaScript Anwendung aus.

   Ihr Komponentenrezept könnte wie das folgende Beispiel aussehen. Diese Komponente führt das StreamManagerS3-Beispiel aus.

   JSON

   ```
   {
     "RecipeFormatVersion": "2020-01-25",
     "ComponentName": "com.example.StreamManagerS3JS",
   ```

```
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses stream manager to upload a file to an S3
 bucket.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.StreamManager": {
      "VersionRequirement": "^2.0.0"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "npm install {artifacts:decompressedPath}/stream-manager-sdk/
aws-greengrass-stream-manager-sdk",
        "run": "export NODE_PATH=$NODE_PATH:{work:path}/node_modules; node
 {artifacts:path}/index.js"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip",
          "Unarchive": "ZIP"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "npm install {artifacts:decompressedPath}/stream-manager-sdk/
aws-greengrass-stream-manager-sdk",
        "run": "set \"NODE_PATH=%NODE_PATH%;{work:path}/node_modules\" & node
 {artifacts:path}/index.js"
      },
      "Artifacts": [
        {
```

```
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip",
          "Unarchive": "ZIP"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.StreamManagerS3JS
ComponentVersion: 1.0.0
ComponentDescription: Uses stream manager to upload a file to an S3 bucket.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.StreamManager:
    VersionRequirement: "^2.0.0"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: npm install {artifacts:decompressedPath}/stream-manager-sdk/aws-
greengrass-stream-manager-sdk
      run: |
        export NODE_PATH=$NODE_PATH:{work:path}/node_modules
        node {artifacts:path}/index.js
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip
        Unarchive: ZIP
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js
  - Platform:
      os: windows
    Lifecycle:
```

```
      install: npm install {artifacts:decompressedPath}/stream-manager-sdk/aws-
greengrass-stream-manager-sdk
      run: |
        set "NODE_PATH=%NODE_PATH%;{work:path}/node_modules"
        node {artifacts:path}/index.js
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip
        Unarchive: ZIP
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js
```

Weitere Informationen zum Entwickeln und Testen von Komponenten finden Sie unterErstellen von AWS IoT Greengrass Komponenten.

Stellen Sie im Anwendungscode eine Connect zum Stream Manager her

Um eine Verbindung zum Stream Manager in Ihrer Anwendung herzustellen, erstellen Sie eine Instanz von `StreamManagerClient` aus dem Stream Manager SDK. Dieser Client stellt über seinen Standardport 8088 oder den von Ihnen angegebenen Port eine Verbindung zur Stream Manager-Komponente her. Weitere Informationen zur Verwendung `StreamManagerClient` nach dem Erstellen einer Instanz finden Sie unterVerwenden von StreamManagerClient für die Arbeit mit Streams.

Example Beispiel: Connect zum Stream-Manager mit Standardport herstellen

Java

```
import com.amazonaws.greengrass.streammanager.client.StreamManagerClient;

public class MyStreamManagerComponent {

    void connectToStreamManagerWithDefaultPort() {
        StreamManagerClient client = StreamManagerClientFactory.standard().build();

        // Use the client.
    }
}
```

Python

```
from stream_manager import (
    StreamManagerClient
)

def connect_to_stream_manager_with_default_port():
    client = StreamManagerClient()

    # Use the client.
```

JavaScript

```
const {
    StreamManagerClient
} = require('aws-greengrass-stream-manager-sdk');

function connectToStreamManagerWithDefaultPort() {
    const client = new StreamManagerClient();

    // Use the client.
}
```

Example Beispiel: Connect zum Stream-Manager mit einem nicht standardmäßigen Port her

Wenn Sie den Stream-Manager mit einem anderen Port als dem Standardport konfigurieren, müssen Sie die Interprozesskommunikation verwenden, um den Port aus der Komponentenkonfiguration abzurufen.

> **Note**
>
> Der `port` Konfigurationsparameter enthält den Wert, den Sie `STREAM_MANAGER_SERVER_PORT` bei der Bereitstellung von Stream Manager angeben.

Java

```
void connectToStreamManagerWithCustomPort() {
    EventStreamRPCConnection eventStreamRpcConnection =
 IPCUtils.getEventStreamRpcConnection();
```

```
    GreengrassCoreIPCClient greengrassCoreIPCClient = new
 GreengrassCoreIPCClient(eventStreamRpcConnection);
    List<String> keyPath = new ArrayList<>();
    keyPath.add("port");

    GetConfigurationRequest request = new GetConfigurationRequest();
    request.setComponentName("aws.greengrass.StreamManager");
    request.setKeyPath(keyPath);
    GetConfigurationResponse response =
            greengrassCoreIPCClient.getConfiguration(request,
 Optional.empty()).getResponse().get();
    String port = response.getValue().get("port").toString();
    System.out.print("Stream Manager is running on port: " + port);

    final StreamManagerClientConfig config = StreamManagerClientConfig.builder()

 .serverInfo(StreamManagerServerInfo.builder().port(Integer.parseInt(port)).build()).build()

    StreamManagerClient client =
 StreamManagerClientFactory.standard().withClientConfig(config).build();

    // Use the client.
}
```

Python

```
import awsiot.greengrasscoreipc
from awsiot.greengrasscoreipc.model import (
    GetConfigurationRequest
)
from stream_manager import (
    StreamManagerClient
)

TIMEOUT = 10

def connect_to_stream_manager_with_custom_port():
    # Use IPC to get the port from the stream manager component configuration.
    ipc_client = awsiot.greengrasscoreipc.connect()
    request = GetConfigurationRequest()
    request.component_name = "aws.greengrass.StreamManager"
    request.key_path = ["port"]
    operation = ipc_client.new_get_configuration()
```

```
    operation.activate(request)
    future_response = operation.get_response()
    response = future_response.result(TIMEOUT)
    stream_manager_port = str(response.value["port"])

    # Use port to create a stream manager client.
    stream_client = StreamManagerClient(port=stream_manager_port)

    # Use the client.
```

Verwenden von StreamManagerClient für die Arbeit mit Streams

Benutzerdefinierte Greengrass-Komponenten, die auf dem Greengrass-Kerngerät ausgeführt werden, können das -`StreamManagerClient`Objekt im Stream Manager SDK verwenden, um Streams im Stream Manager zu erstellen und dann mit den Streams zu interagieren. Wenn eine Komponente einen Stream erstellt, definiert sie die AWS Cloud Ziele, die Priorisierung und andere Export- und Datenaufbewahrungsrichtlinien für den Stream. Um Daten an den Stream-Manager zu senden, hängen Komponenten die Daten an den Stream an. Wenn ein Exportziel für den Stream definiert ist, exportiert Stream Manager den Stream automatisch.

> **Note**
>
> In der Regel sind Clients von Stream Manager benutzerdefinierte Greengrass-Komponenten. Wenn Ihr Geschäftsfall dies erfordert, können Sie auch zulassen, dass Nichtkomponentenprozesse, die auf dem Greengrass-Kern (z. B. einem Docker-Container) ausgeführt werden, mit dem Stream-Manager interagieren. Weitere Informationen finden Sie unter the section called “Client-Authentifizierung”.

Die Ausschnitte in diesem Thema zeigen Ihnen, wie Clients `StreamManagerClient` Methoden aufrufen, um mit Streams zu arbeiten. Für Implementierungsdetails zu den Methoden und ihren Argumenten verwenden Sie die Links zur SDK-Referenz, die nach jedem Ausschnitt aufgeführt ist.

Wenn Sie Stream Manager in einer Lambda-Funktion verwenden, sollte Ihre Lambda-Funktion `StreamManagerClient` außerhalb des Funktionshandlers instanziieren. 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 “Nachrichtenstream aktualisieren”
- the section called “Löschen eines Nachrichten-Streams”

Erstellen eines Nachrichten-Streams

Um einen Stream zu erstellen, ruft eine benutzerdefinierte Greengrass-Komponente die Erstellungsmethode auf und übergibt ein -`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:

- Die Zielziele AWS IoT Analytics, Kinesis Data StreamsAWS IoT SiteWise, und Amazon S3 für automatische Exporte. Weitere Informationen finden Sie unter the section called “Exportkonfigurationen für unterstützte Cloud-Ziele”.
- Export-Priorität. Stream-Manager exportiert Streams mit höherer Priorität vor Streams mit niedrigerer Priorität.
- Maximale Batchgröße und Batch-Intervall für AWS IoT Analytics, Kinesis Data Streams und AWS IoT SiteWise Ziele. Der Stream-Manager exportiert Nachrichten, wenn eine der Bedingungen erfüllt ist.

- T ime-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 finden Sie `MessageStreamDefinitionin` der SDK-Referenz für Ihre Zielsprache:

- MessageStreamDefinition im Java SDK
- MessageStreamDefinition im Node.js SDK
- MessageStreamDefinition im Python SDK

> **Note**
>
> `StreamManagerClient` bietet auch ein Ziel, 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.

Nachdem ein Stream erstellt wurde, können Ihre Greengrass-Komponenten Nachrichten an den Stream anhängen, um Daten für den Export zu senden und Nachrichten aus dem Stream zur lokalen Verarbeitung zu lesen. 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 AWS IoT Analytics oder Kinesis Data Stream zu erstellen, Sie können jedoch mehrere Ziele für einen Stream definieren. Ein Stream hat eine dauerhafte Lebensdauer.

Voraussetzungen

Für diesen Vorgang gelten die folgenden Anforderungen:

- Minimale Stream Manager SDK-Version: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Beispiele

Das folgende Snippet erstellt einen Stream mit dem Namen `StreamName`. Es definiert Stream-Eigenschaften in den untergeordneten Datentypen `MessageStreamDefinition` und .

Python

```
client = StreamManagerClient()

try:
    client.create_message_stream(MessageStreamDefinition(
        name="StreamName",    # Required.
        max_size=268435456,    # Default is 256 MB.
        stream_segment_size=16777216,    # Default is 16 MB.
        time_to_live_millis=None,    # By default, no TTL is enabled.
        strategy_on_full=StrategyOnFull.OverwriteOldestData,    # Required.
        persistence=Persistence.File,    # Default is File.
        flush_on_write=False,    # Default is false.
        export_definition=ExportDefinition(    # Optional. Choose where/how the
 stream is exported to the AWS Cloud.
            kinesis=None,
            iot_analytics=None,
            iot_sitewise=None,
            s3_task_executor=None
        )
    ))
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python-SDK-Referenz: create_message_stream | MessageStreamDefinition

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().build()) {
    client.createMessageStream(
            new MessageStreamDefinition()
                    .withName("StreamName") // Required.
                    .withMaxSize(268435456L)    // Default is 256 MB.
```

```
                    .withStreamSegmentSize(16777216L)    // Default is 16 MB.
                    .withTimeToLiveMillis(null)    // By default, no TTL is enabled.
                    .withStrategyOnFull(StrategyOnFull.OverwriteOldestData)    //
 Required.
                    .withPersistence(Persistence.File)    // Default is File.
                    .withFlushOnWrite(false)    // Default is false.
                    .withExportDefinition(    // Optional. Choose where/how the
 stream is exported to the AWS Cloud.
                            new ExportDefinition()
                                    .withKinesis(null)
                                    .withIotAnalytics(null)
                                    .withIotSitewise(null)
                                    .withS3(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)
            .withS3(null)
        )
```

```
    );
  } catch (e) {
    // Properly handle errors.
  }
});
client.onError((err) => {
  // Properly handle connection errors.
  // This is called only when the connection to the StreamManager server fails.
});
```

Node.js SDK-Referenz: createMessageStream | MessageStreamDefinition

Weitere Informationen zum Konfigurieren von Exportzielen finden Sie unter the section called “Exportkonfigurationen für unterstützte Cloud-Ziele”.

Anhängen einer Nachricht

Um Daten zum Export an den Stream-Manager zu senden, hängen Ihre Greengrass-Komponenten die Daten an den Ziel-Stream an. Das Exportziel bestimmt den Datentyp, der an diese Methode übergeben werden soll.

Voraussetzungen

Für diesen Vorgang gelten die folgenden Anforderungen:

- Minimale Stream Manager SDK-Version: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Beispiele

AWS IoT Analytics oder Kinesis-Data-Streams-Exportziele

Das folgende Snippet fügt eine Nachricht an den Stream namens `StreamName` an. Für Ziele von AWS IoT Analytics oder Kinesis Data Streams hängen Ihre Greengrass-Komponenten einen Datenblob an.

Für diesen Ausschnitt gelten die folgenden Anforderungen:

- Minimale Stream Manager SDK-Version: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Python

```
client = StreamManagerClient()

try:
    sequence_number = client.append_message(stream_name="StreamName",
 data=b'Arbitrary bytes data')
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python-SDK-Referenz: append_message

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().build()) {
    long sequenceNumber = client.appendMessage("StreamName", "Arbitrary byte
 array".getBytes());
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java-SDK-Referenz: appendMessage

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const sequenceNumber = await client.appendMessage("StreamName",
 Buffer.from("Arbitrary byte array"));
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Node.js SDK-Referenz: appendMessage

AWS IoT SiteWise Exportziele

Das folgende Snippet fügt eine Nachricht an den Stream namens `StreamName` an.
Für AWS IoT SiteWise Ziele hängen Ihre Greengrass-Komponenten ein serialisiertes `PutAssetPropertyValueEntry` Objekt an. Weitere Informationen finden Sie unter the section called “Exportieren in AWS IoT SiteWise”.

> **Note**
>
> Wenn Sie Daten an sendenAWS IoT SiteWise, müssen Ihre Daten die Anforderungen der `BatchPutAssetPropertyValue` Aktion erfüllen. Weitere Informationen finden Sie unter BatchPutAssetPropertyValue in der AWS IoT SiteWise-API-Referenz.

Für diesen Ausschnitt gelten die folgenden Anforderungen:

- Minimale Stream Manager SDK-Version: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Python

```
client = StreamManagerClient()

try:
    # SiteWise requires unique timestamps in all messages and also needs timestamps
 not earlier
    # than 10 minutes in the past. Add some randomness to time and offset.

    # Note: To create a new asset property data, you should use the classes defined
 in the
    # greengrasssdk.stream_manager module.

    time_in_nanos = TimeInNanos(
        time_in_seconds=calendar.timegm(time.gmtime()) - random.randint(0, 60),
 offset_in_nanos=random.randint(0, 10000)
    )
    variant = Variant(double_value=random.random())
    asset = [AssetPropertyValue(value=variant, quality=Quality.GOOD,
 timestamp=time_in_nanos)]
```

```
    putAssetPropertyValueEntry =
 PutAssetPropertyValueEntry(entry_id=str(uuid.uuid4()),
 property_alias="PropertyAlias", property_values=asset)
    sequence_number = client.append_message(stream_name="StreamName",
 Util.validate_and_serialize_to_json_bytes(putAssetPropertyValueEntry))
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python-SDK-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 and also needs
 timestamps not earlier
    // than 10 minutes in the past. Add some randomness to time and offset.
    final int maxTimeRandomness = 60;
    final int maxOffsetRandomness = 10000;
    double randomValue = rand.nextDouble();
    TimeInNanos timestamp = new TimeInNanos()
            .withTimeInSeconds(Instant.now().getEpochSecond() -
 rand.nextInt(maxTimeRandomness))
            .withOffsetInNanos((long) (rand.nextInt(maxOffsetRandomness)));
    AssetPropertyValue entry = new AssetPropertyValue()
            .withValue(new Variant().withDoubleValue(randomValue))
            .withQuality(Quality.GOOD)
            .withTimestamp(timestamp);
    entries.add(entry);

    PutAssetPropertyValueEntry putAssetPropertyValueEntry = new
 PutAssetPropertyValueEntry()
            .withEntryId(UUID.randomUUID().toString())
            .withPropertyAlias("PropertyAlias")
```

```
            .withPropertyValues(entries);
    long sequenceNumber = client.appendMessage("StreamName",
 ValidateAndSerialize.validateAndSerializeToJsonBytes(putAssetPropertyValueEntry));
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java-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-Exportziele

Der folgende Codeausschnitt hängt eine Exportaufgabe an den Stream mit dem Namen an`StreamName`. Für Amazon S3-Ziele hängen Ihre Greengrass-Komponenten ein serialisiertes `S3ExportTaskDefinition` Objekt an, das Informationen über die Quelleingabedatei und das Amazon S3-Zielobjekt enthält. Wenn das angegebene Objekt nicht vorhanden ist, erstellt Stream Manager es für Sie. Weitere Informationen finden Sie unter the section called “Exportieren nach Amazon S3”.

Für diesen Ausschnitt gelten die folgenden Anforderungen:

- Minimale Stream Manager SDK-Version: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Python

```
client = StreamManagerClient()

try:
    # Append an Amazon S3 Task definition and print the sequence number.
    s3_export_task_definition = S3ExportTaskDefinition(input_url="URLToFile",
 bucket="BucketName", key="KeyName")
    sequence_number = client.append_message(stream_name="StreamName",
 Util.validate_and_serialize_to_json_bytes(s3_export_task_definition))
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python-SDK-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 von Nachrichten aus einem Stream.

Voraussetzungen

Für diesen Vorgang gelten die folgenden Anforderungen:

- Minimale Stream Manager SDK-Version: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

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.
```

Python-SDK-Referenz: read_messages | ReadMessagesOptions

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().build()) {
    List<Message> messages = client.readMessages("StreamName",
            // By default, if no options are specified, it tries to read one message
 from the beginning of the stream.
            new ReadMessagesOptions()
                    // Try to read from sequence number 100 or greater. By default
 this is 0.
                    .withDesiredStartSequenceNumber(100L)
                    // Try to read 10 messages. If 10 messages are not available,
 then NotEnoughMessagesException is raised. By default, this is 1.
                    .withMinMessageCount(10L)
                    // Accept up to 100 messages. By default this is 1.
                    .withMaxMessageCount(100L)
                    // Try to wait at most 5 seconds for the min_messsage_count to
 be fulfilled. By default, this is 0, which immediately returns the messages or an
 exception.
                    .withReadTimeoutMillis(Duration.ofSeconds(5L).toMillis())
    );
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

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

Für diesen Vorgang gelten die folgenden Anforderungen:

- Minimale Stream Manager SDK-Version: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Beispiele

Das folgende Snippet ruft eine Liste der Streams (nach Namen) im Stream-Manager ab.

Python

```
client = StreamManagerClient()

try:
    stream_names = client.list_streams()
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
```

```
    # Properly handle errors.
```

Python-SDK-Referenz: list_streams

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().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

Abrufen von Metadaten zu einem Stream, einschließlich Stream-Definition, Größe und Exportstatus.

Voraussetzungen

Für diesen Vorgang gelten die folgenden Anforderungen:

- Minimale Stream Manager SDK-Version: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Beispiele

Das folgende Snippet ruft Metadaten über den Stream mit dem Namen `StreamName` ab, einschließlich Definition, Größe und Exporterstatus des Streams.

Python

```
client = StreamManagerClient()

try:
    stream_description = client.describe_message_stream(stream_name="StreamName")
    if stream_description.export_statuses[0].error_message:
        # The last export of export destination 0 failed with some error
        # Here is the last sequence number that was successfully exported
        stream_description.export_statuses[0].last_exported_sequence_number

    if (stream_description.storage_status.newest_sequence_number >
            stream_description.export_statuses[0].last_exported_sequence_number):
        pass
        # The end of the stream is ahead of the last exported sequence number
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python-SDK-Referenz: describe_message_stream

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().build()) {
    MessageStreamInfo description = client.describeMessageStream("StreamName");
    String lastErrorMessage =
 description.getExportStatuses().get(0).getErrorMessage();
    if (lastErrorMessage != null && !lastErrorMessage.equals("")) {
        // The last export of export destination 0 failed with some error.
        // Here is the last sequence number that was successfully exported.
        description.getExportStatuses().get(0).getLastExportedSequenceNumber();
    }

    if (description.getStorageStatus().getNewestSequenceNumber() >
```

```
            description.getExportStatuses().get(0).getLastExportedSequenceNumber())
 {
        // The end of the stream is ahead of the last exported sequence number.
    }
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java-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

Nachrichtenstream aktualisieren

Aktualisieren Sie die Eigenschaften eines vorhandenen Streams. Möglicherweise möchten Sie einen Stream aktualisieren, wenn sich Ihre Anforderungen ändern, nachdem der Stream erstellt wurde. Beispielsweise:

- Fügen Sie eine neue Exportkonfiguration für ein AWS Cloud Ziel hinzu.
- Erhöhen Sie die maximale Größe eines Streams, um zu ändern, wie Daten exportiert oder aufbewahrt werden. Beispielsweise kann die Stream-Größe in Kombination mit Ihrer Strategie für vollständige Einstellungen dazu führen, dass Daten gelöscht oder abgelehnt werden, bevor der Stream-Manager sie verarbeiten kann.
- Pausieren und Fortsetzen von Exporten, z. B. wenn Exportaufgaben lange laufen und Sie Ihre Upload-Daten rationieren möchten.

Ihre Greengrass-Komponenten folgen diesem allgemeinen Prozess, um einen Stream zu aktualisieren:

1. Rufen Sie die Beschreibung des Streams ab.
2. Aktualisieren Sie die Zieleigenschaften für die entsprechenden `MessageStreamDefinition` und untergeordneten Objekte.
3. Übergeben Sie das aktualisierte `MessageStreamDefinition`. Stellen Sie sicher, dass Sie die vollständigen Objektdefinitionen für den aktualisierten Stream angeben. Undefinierte Eigenschaften setzen auf die Standardwerte zurück.

   Sie können die Sequenznummer der Nachricht angeben, die als Startnachricht im Export verwendet werden soll.

Voraussetzungen

Für diesen Vorgang gelten die folgenden Anforderungen:

- Minimale Stream Manager SDK-Version: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Beispiele

Der folgende Codeausschnitt aktualisiert den Stream mit dem Namen `StreamName`. Es aktualisiert mehrere Eigenschaften eines Streams, der nach Kinesis Data Streams exportiert wird.

Python

```
client = StreamManagerClient()

try:
    message_stream_info = client.describe_message_stream(STREAM_NAME)
    message_stream_info.definition.max_size=536870912
    message_stream_info.definition.stream_segment_size=33554432
    message_stream_info.definition.time_to_live_millis=3600000
    message_stream_info.definition.strategy_on_full=StrategyOnFull.RejectNewData
    message_stream_info.definition.persistence=Persistence.Memory
    message_stream_info.definition.flush_on_write=False
    message_stream_info.definition.export_definition.kinesis=
        [KinesisConfig(
            # Updating Export definition to add a Kinesis Stream configuration.
            identifier=str(uuid.uuid4()), kinesis_stream_name=str(uuid.uuid4()))]
    client.update_message_stream(message_stream_info.definition)
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python-SDK-Referenz: updateMessageStream | MessageStreamDefinition

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    MessageStreamInfo messageStreamInfo = client.describeMessageStream(STREAM_NAME);
    // Update the message stream with new values.
    client.updateMessageStream(
        messageStreamInfo.getDefinition()
            .withStrategyOnFull(StrategyOnFull.RejectNewData) // Required. Updating
 Strategy on full to reject new data.
           // Max Size update should be greater than initial Max Size defined in
 Create Message Stream request
            .withMaxSize(536870912L) // Update Max Size to 512 MB.
            .withStreamSegmentSize(33554432L) // Update Segment Size to 32 MB.
            .withFlushOnWrite(true) // Update flush on write to true.
            .withPersistence(Persistence.Memory) // Update the persistence to
 Memory.
            .withTimeToLiveMillis(3600000L)    // Update TTL to 1 hour.
```

```
            .withExportDefinition(
                // Optional. Choose where/how the stream is exported to the AWS
 Cloud.
                messageStreamInfo.getDefinition().getExportDefinition().
                    // Updating Export definition to add a Kinesis Stream
 configuration.
                    .withKinesis(new ArrayList<KinesisConfig>() {{
                        add(new KinesisConfig()
                            .withIdentifier(EXPORT_IDENTIFIER)
                            .withKinesisStreamName("test"));
                        }})
            );
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java-SDK-Referenz: update_message_stream | MessageStreamDefinition

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const messageStreamInfo = await c.describeMessageStream(STREAM_NAME);
        await client.updateMessageStream(
            messageStreamInfo.definition
                // Max Size update should be greater than initial Max Size defined
 in Create Message Stream request
                .withMaxSize(536870912)    // Default is 256 MB. Updating Max Size
 to 512 MB.
                .withStreamSegmentSize(33554432)    // Default is 16 MB. Updating
 Segment Size to 32 MB.
                .withTimeToLiveMillis(3600000)    // By default, no TTL is enabled.
 Update TTL to 1 hour.
                .withStrategyOnFull(StrategyOnFull.RejectNewData)    // Required.
 Updating Strategy on full to reject new data.
                .withPersistence(Persistence.Memory)    // Default is File. Update
 the persistence to Memory
                .withFlushOnWrite(true)    // Default is false. Updating to true.
                .withExportDefinition(
                    // Optional. Choose where/how the stream is exported to the AWS
 Cloud.
                    messageStreamInfo.definition.exportDefinition
```

```
                        // Updating Export definition to add a Kinesis Stream
 configuration.
                        .withKinesis([new
 KinesisConfig().withIdentifier(uuidv4()).withKinesisStreamName(uuidv4())])
                )
        );
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Node.js SDK-Referenz: updateMessageStream | MessageStreamDefinition

Einschränkungen für die Aktualisierung von Streams

Beim Aktualisieren von Streams gelten die folgenden Einschränkungen. Sofern in der folgenden Liste nicht anders angegeben, werden Aktualisierungen sofort wirksam.

- Sie können die Persistenz eines Streams nicht aktualisieren. Um dieses Verhalten zu ändern, löschen Sie den Stream und erstellen Sie einen Stream, der die neue Persistenzrichtlinie definiert.
- Sie können die maximale Größe eines Streams nur unter den folgenden Bedingungen aktualisieren:
  - Die maximale Größe muss größer oder gleich der aktuellen Größe des Streams sein. Um diese Informationen zu finden, beschreiben Sie den Stream und überprüfen Sie dann den Speicherstatus des zurückgegebenen `MessageStreamInfo` Objekts.
  - Die maximale Größe muss größer oder gleich der Segmentgröße des Streams sein.
- Sie können die Stream-Segmentgröße auf einen Wert aktualisieren, der 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 Append-Operationen. Wenn Sie diesen Wert verringern, löscht Stream Manager möglicherweise auch vorhandene Segmente, die die TTL überschreiten.
- Aktualisierungen der Strategie für die vollständige Eigenschaft gelten für neue Append-Operationen. Wenn Sie die Strategie zum Überschreiben der ältesten Daten festlegen,

überschreibt Stream Manager möglicherweise auch vorhandene Segmente basierend auf der neuen Einstellung.

- Aktualisierungen der Eigenschaft flush on write gelten für neue Nachrichten.
- Aktualisierungen der Exportkonfigurationen gelten für neue Exporte. Die Aktualisierungsanforderung muss alle Exportkonfigurationen enthalten, die Sie unterstützen möchten. Andernfalls löscht der Stream-Manager sie.
  - Wenn Sie eine Exportkonfiguration aktualisieren, geben Sie die Kennung der Zielexportkonfiguration an.
  - Um eine Exportkonfiguration hinzuzufügen, geben Sie eine eindeutige Kennung für die neue Exportkonfiguration an.
  - Um eine Exportkonfiguration zu löschen, lassen Sie die Exportkonfiguration weg.
- Um die Startsequenznummer einer Exportkonfiguration in einem Stream zu aktualisieren, müssen Sie einen Wert angeben, der kleiner als die neueste Sequenznummer ist. Um diese Informationen zu finden, beschreiben Sie den Stream und überprüfen Sie dann den Speicherstatus des zurückgegebenen `MessageStreamInfo` Objekts.

Löschen eines Nachrichten-Streams

Löscht einen Stream. Wenn Sie einen Stream löschen, werden alle gespeicherten Daten für den Stream von der Festplatte gelöscht.

Voraussetzungen

Für diesen Vorgang gelten die folgenden Anforderungen:

- Minimale Stream Manager SDK-Version: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Beispiele

Das folgende Snippet löscht den Stream mit dem Namen `StreamName`.

Python

```
client = StreamManagerClient()

try:
    client.delete_message_stream(stream_name="StreamName")
```

```
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python-SDK-Referenz: deleteMessageStream

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().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.
});
```

Referenz zum Node.js SDK: deleteMessageStream

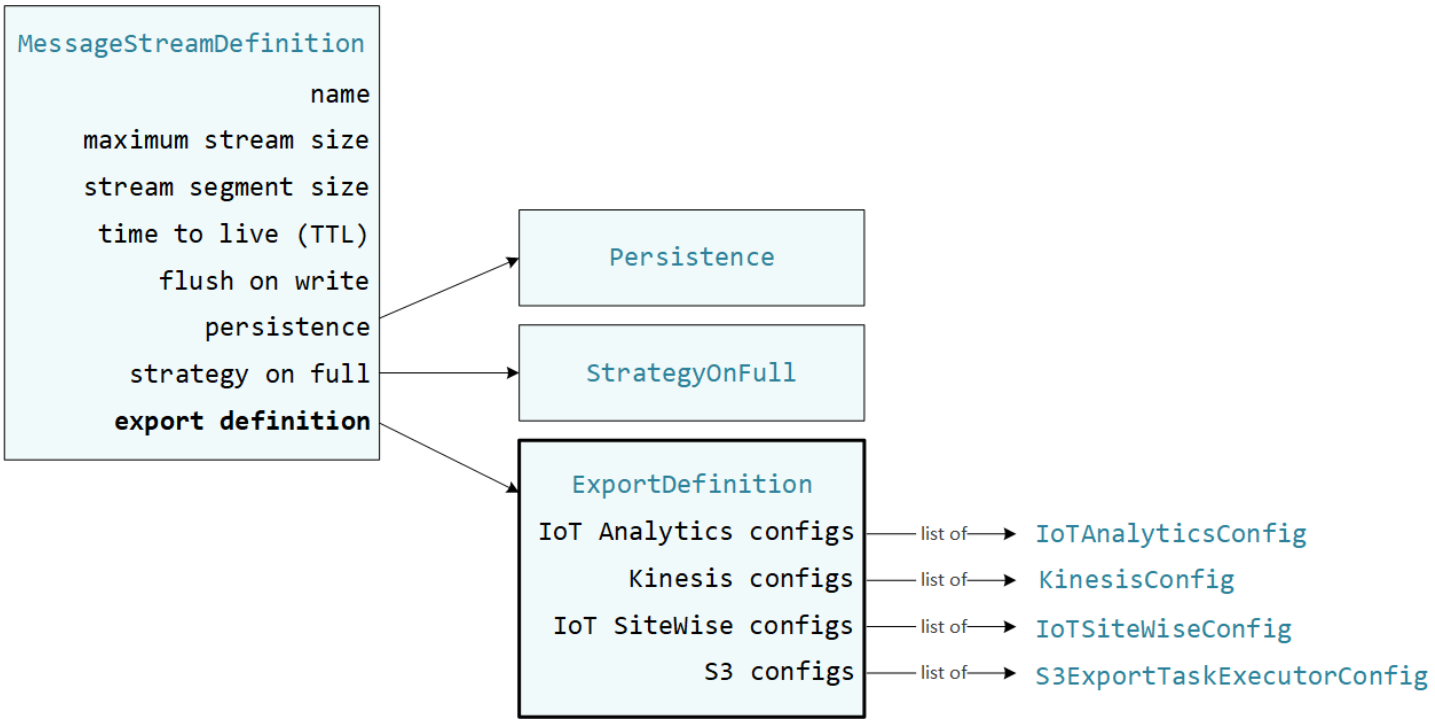
Weitere Informationen finden Sie auch unter

- Verwalten von Datenströmen auf Greengrass-Core-Geräten
- Konfigurieren des AWS IoT Greengrass-Stream-Managers
- Exportkonfigurationen für unterstützte AWS Cloud Ziele

- `StreamManagerClient` in der Stream Manager SDK-Referenz:
  - Python
  - Java
  - Node.js

Exportkonfigurationen für unterstützte AWS Cloud Ziele

Benutzerdefinierte Greengrass-Komponenten verwenden `StreamManagerClient` im Stream Manager SDK, um mit dem Stream Manager zu interagieren. Wenn eine Komponente einen Stream erstellt oder einen Stream aktualisiert, übergibt sie ein `MessageStreamDefinition` Objekt, das Stream-Eigenschaften darstellt, einschließlich der Exportdefinition. Das -`ExportDefinition`Objekt enthält die für den Stream definierten Exportkonfigurationen. Stream Manager verwendet diese Exportkonfigurationen, um zu bestimmen, wo und wie der Stream exportiert werden soll.



Sie können null oder mehr Exportkonfigurationen für einen Stream definieren, einschließlich mehrerer Exportkonfigurationen für einen einzelnen Zieltyp. Sie können beispielsweise einen Stream in zwei AWS IoT Analytics Kanäle und einen Kinesis-Datenstrom exportieren.

Bei fehlgeschlagenen Exportversuchen versucht der Stream-Manager kontinuierlich, Daten AWS Cloud in Intervallen von bis zu fünf Minuten nach zu exportieren. Die Anzahl der Wiederholungsversuche hat kein maximales Limit.

> **Note**
>
> `StreamManagerClient` bietet auch ein Ziel, 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.

Unterstützte AWS Cloud Ziele

- AWS IoT Analytics-Kanäle
- Amazon Kinesis-Datenströme
- AWS IoT SiteWise Komponenteneigenschaften
- Amazon S3-Objekte

Sie sind für die Wartung dieser AWS Cloud Ressourcen verantwortlich.

AWS IoT Analytics-Kanäle

Stream Manager unterstützt automatische Exporte nach AWS IoT Analytics. AWS IoT Analytics ermöglicht Ihnen eine erweiterte Analyse Ihrer Daten, um Geschäftsentscheidungen zu treffen und Machine-Learning-Modelle zu verbessern. Weitere Informationen finden Sie unter Was ist AWS IoT Analytics? im AWS IoT Analytics -Benutzerhandbuch.

Im Stream Manager SDK verwenden Ihre Greengrass-Komponenten die , `IoTAnalyticsConfig` um die Exportkonfiguration für diesen Zieltyp zu definieren. Weitere Informationen finden Sie in der SDK-Referenz für Ihre Zielsprache:

- IoTAnalyticsConfig im Python SDK
- IoTAnalyticsConfig im Java SDK
- IoTAnalyticsConfig im Node.js SDK

Voraussetzungen

Für dieses Exportziel gelten die folgenden Anforderungen:

- Zielkanäle in AWS IoT Analytics müssen sich im selben AWS-Konto und in demselben AWS-Region wie das Greengrass-Kerngerät befinden.

- Der Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS muss der `-iotanalytics:BatchPutMessage`Berechtigung erlauben, Kanäle anzuvisieren. Beispielsweise:

```
{
  "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 auf -Ressourcen gewähren, z. B. durch die Verwendung eines Platzhalter-*Namensschemas. Weitere Informationen finden Sie unter Hinzufügen und Entfernen von IAM-Richtlinien im IAM-Benutzerhandbuch.

Exportieren in AWS IoT Analytics

Um einen Stream zu erstellen, der nach exportiert wirdAWS IoT Analytics, erstellen Ihre Greengrass-Komponenten einen Stream mit einer Exportdefinition, die ein oder mehrere `IoTAnalyticsConfig` Objekte enthält. Dieses Objekt definiert Exporteinstellungen wie Zielkanal, Batch-Größe, Batch-Intervall und Priorität.

Wenn Ihre Greengrass-Komponenten Daten von Geräten empfangen, hängen sie Nachrichten an, die einen Blob von Daten enthalten, an den Ziel-Stream an.

Anschließend exportiert Stream Manager die Daten basierend auf den Batch-Einstellungen und der Priorität, die in den Exportkonfigurationen des Streams definiert sind.

Amazon Kinesis-Datenströme

Stream Manager unterstützt automatische Exporte nach Amazon Kinesis Data Streams. Kinesis Data Streams wird häufig verwendet, um Daten mit hohem Volumen zu aggregieren und in ein Data

Warehouse oder einen MapReduce Cluster zu laden. Weitere Informationen finden Sie unter Was ist Amazon Kinesis Data Streams? im Amazon Kinesis-Entwicklerhandbuch.

Im Stream Manager SDK verwenden Ihre Greengrass-Komponenten die , `KinesisConfig` um die Exportkonfiguration für diesen Zieltyp zu definieren. Weitere Informationen finden Sie in der SDK-Referenz für Ihre Zielsprache:

- KinesisConfig im Python SDK
- KinesisConfig im Java SDK
- KinesisConfig im Node.js SDK

Voraussetzungen

Für dieses Exportziel gelten die folgenden Anforderungen:

- Zielstreams in Kinesis Data Streams müssen sich im selben AWS-Konto und im selben AWS-Region wie das Greengrass-Kerngerät befinden.
- Der Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS muss der -`kinesis:PutRecords`Berechtigung das Ziel von Datenströmen erlauben. Beispielsweise:

```
{
  "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, z. B. durch die Verwendung eines Platzhalter-*Namensschemas. Weitere Informationen finden Sie unter Hinzufügen und Entfernen von IAM-Richtlinien im IAM-Benutzerhandbuch.

Exportieren zu Kinesis Data Streams

Um einen Stream zu erstellen, der nach Kinesis Data Streams exportiert wird, erstellen Ihre Greengrass-Komponenten einen Stream mit einer Exportdefinition, die ein oder mehrere `KinesisConfig` Objekte enthält. Dieses Objekt definiert Exporteinstellungen wie den Zieldatenstrom, die Batchgröße, das Batchintervall und die Priorität.

Wenn Ihre Greengrass-Komponenten Daten von Geräten empfangen, hängen sie Nachrichten an, die einen Blob von Daten enthalten, an den Ziel-Stream an. Anschließend exportiert Stream Manager die Daten basierend auf den Batch-Einstellungen und der Priorität, die in den Exportkonfigurationen des Streams definiert sind.

Stream Manager generiert eine eindeutige, zufällige UUID als Partitionsschlüssel für jeden Datensatz, der in Amazon Kinesis hochgeladen wird.

AWS IoT SiteWise Komponenteneigenschaften

Stream Manager unterstützt automatische Exporte nach AWS IoT SiteWise. AWS IoT SiteWise Mit können Sie Daten von Industrieanlagen erfassen, organisieren und analysieren. Weitere Informationen finden Sie unter Was ist AWS IoT SiteWise? im AWS IoT SiteWise - Benutzerhandbuch.

Im Stream Manager SDK verwenden Ihre Greengrass-Komponenten die , `IoTSiteWiseConfig` um die Exportkonfiguration für diesen Zieltyp zu definieren. Weitere Informationen finden Sie in der SDK-Referenz für Ihre Zielsprache:

- IoTSiteWiseConfig im Python SDK
- IoTSiteWiseConfig im Java SDK
- IoTSiteWiseConfig im Node.js SDK

> **Note**
>
> AWS bietet auch AWS IoT SiteWise Komponenten, die eine vorgefertigte Lösung bieten, mit der Sie Daten aus OPC-UA-Quellen streamen können. Weitere Informationen finden Sie unter IoT SiteWise -OPC-UA-Kollektor.

Voraussetzungen

Für dieses Exportziel gelten die folgenden Anforderungen:

- Die Eigenschaften der Zielkomponente in AWS IoT SiteWise müssen sich in derselben AWS-Konto und demselben AWS-Region wie das Greengrass-Kerngerät befinden.

  > **Note**
  >
  > Eine Liste der AWS-Regionen, die AWS IoT SiteWise unterstützt, finden Sie unter -AWS IoT SiteWiseEndpunkte und -Kontingente in der AWS Allgemeinen Referenz zu .

- Der Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS muss der `-iotsitewise:BatchPutAssetPropertyValue`Berechtigung erlauben, auf Komponenteneigenschaften abzuzielen. Die folgende Beispielrichtlinie verwendet den `-iotsitewise:assetHierarchyPath`Bedingungsschlüssel, um Zugriff auf eine Ziel-Root-Komponente und ihre untergeordneten Elemente zu gewähren. Sie können die `Condition` aus der Richtlinie entfernen, um den Zugriff auf alle Ihre AWS IoT SiteWise Komponenten zu erlauben, oder ARNs einzelner Komponenten angeben.

```
{
  "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 auf -Ressourcen gewähren, z. B. durch die Verwendung eines Platzhalter-*Namensschemas. Weitere Informationen finden Sie unter Hinzufügen und Entfernen von IAM-Richtlinien im IAM-Benutzerhandbuch.

Wichtige Sicherheitsinformationen finden Sie unter - BatchPutAssetPropertyValue Autorisierung im AWS IoT SiteWise -Benutzerhandbuch.

Exportieren in AWS IoT SiteWise

Um einen Stream zu erstellen, der nach exportiert wirdAWS IoT SiteWise, erstellen Ihre Greengrass-Komponenten einen Stream mit einer Exportdefinition, die ein oder mehrere `IoTSiteWiseConfig` Objekte enthält. Dieses Objekt definiert Exporteinstellungen wie Batchgröße, Batchintervall und Priorität.

Wenn Ihre Greengrass-Komponenten Komponenteneigenschaftsdaten von Geräten erhalten, hängen sie Nachrichten, die die Daten enthalten, an den Ziel-Stream an. Nachrichten sind JSON-serialisierte `PutAssetPropertyValueEntry` Objekte, die Eigenschaftswerte für eine oder mehrere Komponenteneigenschaften enthalten. Weitere Informationen finden Sie unter Nachricht für AWS IoT SiteWise Exportziele anhängen.

> **Note**
>
> Wenn Sie Daten an sendenAWS IoT SiteWise, müssen Ihre Daten die Anforderungen der `BatchPutAssetPropertyValue` Aktion erfüllen. Weitere Informationen finden Sie unter BatchPutAssetPropertyValue in der AWS IoT SiteWise-API-Referenz.

Anschließend exportiert Stream Manager die Daten basierend auf den Batch-Einstellungen und der Priorität, die in den Exportkonfigurationen des Streams definiert sind.

Sie können Ihre Stream-Manager-Einstellungen und die Greengrass-Komponentenlogik anpassen, um Ihre Exportstrategie zu entwerfen. Beispielsweise:

- Legen Sie für Exporte nahezu in Echtzeit niedrige Batch-Größen- und Intervalleinstellungen fest und fügen Sie die Daten an den Stream an, wenn sie empfangen werden.
- Um die Stapelverarbeitung zu optimieren, Bandbreitenbeschränkungen zu minimieren oder die Kosten zu minimieren, können Ihre Greengrass-Komponenten die timestamp-quality-value (TQV)-Datenpunkte, die für eine einzelne Komponenteneigenschaft empfangen wurden, bündeln, bevor

die Daten an den Stream angehängt werden. Eine Strategie besteht darin, Einträge für bis zu 10 verschiedene Eigenschafts-Asset-Kombinationen oder Eigenschaftsaliasnamen in einer Nachricht zu sammeln, anstatt mehr als einen Eintrag für dieselbe Eigenschaft zu senden. Dies hilft dem Stream-Manager, die AWS IoT SiteWise Kontingente von einzuhalten.

Amazon S3-Objekte

Stream Manager unterstützt automatische Exporte nach Amazon S3. Sie können Amazon S3 verwenden, um große Datenmengen zu speichern und abzurufen. Weitere Informationen finden Sie unter Was ist Amazon S3? im Amazon Simple Storage Service-Entwicklerhandbuch.

Im Stream Manager SDK verwenden Ihre Greengrass-Komponenten die , `S3ExportTaskExecutorConfig` um die Exportkonfiguration für diesen Zieltyp zu definieren. Weitere Informationen finden Sie in der SDK-Referenz für Ihre Zielsprache:

- S3ExportTaskExecutorConfig im Python SDK
- S3ExportTaskExecutorConfig im Java SDK
- S3ExportTaskExecutorConfig im Node.js SDK

Voraussetzungen

Für dieses Exportziel gelten die folgenden Anforderungen:

- Ziel-Amazon S3-Buckets müssen sich in derselben AWS-Konto wie das Greengrass-Kerngerät befinden.
- Wenn eine Lambda-Funktion, die im Greengrass-Container-Modus ausgeführt wird, Eingabedateien in ein Eingabedateiverzeichnis schreibt, müssen Sie das Verzeichnis als Volume im Container mit Schreibberechtigungen mounten. Dadurch wird sichergestellt, dass die Dateien in das Stammdateisystem geschrieben und für die Stream-Manager-Komponente sichtbar sind, die außerhalb des Containers ausgeführt wird.
- Wenn eine Docker-Containerkomponente Eingabedateien in ein Eingabedateiverzeichnis schreibt, müssen Sie das Verzeichnis als Volume im Container mit Schreibberechtigungen mounten. Dadurch wird sichergestellt, dass die Dateien in das Stammdateisystem geschrieben und für die Stream-Manager-Komponente sichtbar sind, die außerhalb des Containers ausgeführt wird.
- Der Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS muss die folgenden Berechtigungen für die Ziel-Buckets zulassen. Beispielsweise:

```
{
  "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 auf -Ressourcen gewähren, z. B. durch die Verwendung eines Platzhalter-*Namensschemas. Weitere Informationen finden Sie unter Hinzufügen und Entfernen von IAM-Richtlinien im IAM-Benutzerhandbuch.

Exportieren nach Amazon S3

Um einen Stream zu erstellen, der nach Amazon S3 exportiert wird, verwenden Ihre Greengrass-Komponenten das -`S3ExportTaskExecutorConfig`Objekt, um die Exportrichtlinie zu konfigurieren. Die Richtlinie definiert Exporteinstellungen wie den Schwellenwert und die Priorität des mehrteiligen Uploads. Für Amazon S3-Exporte lädt der Stream-Manager Daten hoch, die er aus lokalen Dateien auf dem Core-Gerät liest. Um einen Upload zu initiieren, hängen Ihre Greengrass-Komponenten eine Exportaufgabe an den Ziel-Stream an. Die Exportaufgabe enthält Informationen über die Eingabedatei und das Amazon S3-Zielobjekt. Stream Manager führt Aufgaben in der Reihenfolge aus, in der sie an den Stream angehängt werden.

> **Note**
>
> Der Ziel-Bucket muss bereits in Ihrem vorhanden seinAWS-Konto. Wenn kein Objekt für den angegebenen Schlüssel vorhanden ist, erstellt Stream Manager das Objekt für Sie.

Stream Manager verwendet die Eigenschaft für den Schwellenwert für mehrteilige Uploads, die Einstellung für die minimale Teilegröße und die Größe der Eingabedatei, um zu bestimmen, wie Daten hochgeladen werden sollen. Der Schwellenwert für mehrteilige Uploads 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-Zielobjekte angeben, können gültige Java DateTimeFormatter-Zeichenfolgen in `!{timestamp:value}` Platzhaltern enthalten. 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 aufgelöst`my-key/2020/12/31/data.txt`.

```
my-key/!{timestamp:YYYY}/!{timestamp:MM}/!{timestamp:dd}/data.txt
```

> **Note**
>
> Wenn Sie den Exportstatus für einen Stream überwachen möchten, erstellen Sie zunächst einen Status-Stream und konfigurieren Sie dann den Export-Stream für dessen Verwendung. 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 Beispiel-Workflow zeigt, wie Sie Greengrass-Komponenten verwenden können, um diese Daten zu verwalten.

1. Ein lokaler Prozess empfängt Daten von Geräten oder Telefonie und schreibt die Daten dann in Dateien in einem Verzeichnis auf dem Core-Gerät. Dies sind die Eingabedateien für Stream Manager.
2. Eine Greengrass-Komponente scannt das Verzeichnis und fügt eine Exportaufgabe an den Ziel-Stream an, wenn eine neue Datei erstellt wird. Die Aufgabe ist ein JSON-serialisiertes `S3ExportTaskDefinition` Objekt, das die URL der Eingabedatei, den Amazon S3-Ziel-Bucket und -Schlüssel sowie optionale Benutzermetadaten angibt.
3. Stream Manager liest die Eingabedatei und exportiert die Daten in der Reihenfolge der angehängten Aufgaben nach Amazon S3. Der Ziel-Bucket muss bereits in Ihrem vorhanden seinAWS-Konto. Wenn kein Objekt für den angegebenen Schlüssel vorhanden ist, erstellt Stream Manager das Objekt für Sie.

4. Die Greengrass-Komponente liest Nachrichten aus einem Statusstream, um den Exportstatus zu überwachen. Nach Abschluss der Exportaufgaben kann die Greengrass-Komponente die entsprechenden Eingabedateien löschen. Weitere Informationen finden Sie unter the section called "Überwachen von Exportaufgaben".

Überwachen von Exportaufgaben

Sie können Code erstellen, den IoT-Anwendungen verwenden, um den Status Ihrer Amazon S3-Exporte zu überwachen. Ihre Greengrass-Komponenten müssen einen Statusstream 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.

Erstellen Sie zunächst einen Stream, der als Statusstream verwendet werden soll. Sie können die Größe und die Aufbewahrungsrichtlinien für den Stream konfigurieren, um die Lebensdauer der Statusmeldungen zu steuern. Beispielsweise:

- Setzen Sie `Persistence` auf `Memory`, wenn Sie die Statusmeldungen nicht speichern möchten.
- Setzen Sie den Wert `StrategyOnFull` auf , `OverwriteOldestData` damit keine neuen Statusmeldungen verloren gehen.

Erstellen oder aktualisieren Sie dann den Exportstream, um den Statusstream zu verwenden. Legen Sie insbesondere die Statuskonfigurationseigenschaft der `S3ExportTaskExecutorConfig` Exportkonfiguration des Streams fest. Diese Einstellung weist den Stream-Manager an, Statusmeldungen zu den Exportaufgaben in den Status-Stream zu schreiben. Geben Sie im `StatusConfig` Objekt den Namen des Statusstreams und die Ausführlichkeitsstufe an. Die folgenden unterstützten Werte reichen von am wenigsten ausführlich (`ERROR`) bis am ausführlichsten (`TRACE`). Der Standardwert ist `INFO`.

- `ERROR`
- `WARN`
- `INFO`
- `DEBUG`
- `TRACE`

Der folgende Beispiel-Workflow zeigt, wie Greengrass-Komponenten einen Statusstream verwenden können, um den Exportstatus zu überwachen.

1. Wie im vorherigen Workflow beschrieben, hängt eine Greengrass-Komponente eine Exportaufgabe an einen Stream an, der so konfiguriert ist, dass Statusmeldungen über Exportaufgaben in einen Statusstream geschrieben werden. Die Append-Operation gibt eine Sequenznummer zurück, die die Aufgaben-ID darstellt.
2. Eine Greengrass-Komponente liest Nachrichten sequenziell aus dem Status-Stream und filtert dann die Nachrichten basierend auf dem Stream-Namen und der Aufgaben-ID oder basierend auf einer Exportaufgabeneigenschaft aus dem Nachrichtenkontext. Die Greengrass-Komponente kann beispielsweise nach der URL der Eingabedatei der Exportaufgabe filtern, die durch das `S3ExportTaskDefinition` Objekt im Nachrichtenkontext dargestellt wird.

   Die folgenden Statuscodes zeigen an, dass eine Exportaufgabe den Status „Abgeschlossen“ erreicht hat:

   - `Success`. Der Upload wurde erfolgreich abgeschlossen.
   - `Failure`. Stream Manager ist auf einen Fehler gestoßen, z. B. ist der angegebene Bucket nicht vorhanden. Nachdem Sie das Problem behoben haben, können Sie die Exportaufgabe erneut an den Stream anhängen.
   - `Canceled`. Die Aufgabe wurde gestoppt, weil die Stream- oder Exportdefinition gelöscht wurde oder der time-to-live (TTL)-Zeitraum der Aufgabe abgelaufen ist.

   > **Note**
   >
   > Die Aufgabe hat möglicherweise auch den Status `InProgress` oder `Warning`. Stream Manager gibt Warnungen aus, wenn ein Ereignis einen Fehler zurückgibt, der sich nicht auf die Ausführung der Aufgabe auswirkt. Wenn beispielsweise ein teilweiser Upload nicht bereinigt werden kann, wird eine Warnung zurückgegeben.

3. Nach Abschluss der Exportaufgaben kann die Greengrass-Komponente die entsprechenden Eingabedateien löschen.

Das folgende Beispiel zeigt, wie eine Greengrass-Komponente Statusmeldungen lesen und verarbeiten kann.

Python

```
import time
from stream_manager import (
    ReadMessagesOptions,
    Status,
    StatusConfig,
    StatusLevel,
    StatusMessage,
    StreamManagerClient,
)
from stream_manager.util import Util

client = StreamManagerClient()

try:
    # Read the statuses from the export status stream
    is_file_uploaded_to_s3 = False
    while not is_file_uploaded_to_s3:
        try:
            messages_list = client.read_messages(
                "StatusStreamName", ReadMessagesOptions(min_message_count=1,
 read_timeout_millis=1000)
            )
            for message in messages_list:
                # Deserialize the status message first.
                status_message = Util.deserialize_json_bytes_to_obj(message.payload,
 StatusMessage)

                # Check the status of the status message. If the status is
 "Success",
                # the file was successfully uploaded to S3.
                # If the status was either "Failure" or "Cancelled", the server was
 unable to upload the file to S3.
                # We will print the message for why the upload to S3 failed from the
 status message.
                # If the status was "InProgress", the status indicates that the
 server has started uploading
                # the S3 task.
                if status_message.status == Status.Success:
                    logger.info("Successfully uploaded file at path " + file_url + "
 to S3.")
                    is_file_uploaded_to_s3 = True
```

```
                elif status_message.status == Status.Failure or
 status_message.status == Status.Canceled:
                    logger.info(
                        "Unable to upload file at path " + file_url + " to S3.
 Message: " + status_message.message
                    )
                    is_file_uploaded_to_s3 = True
            time.sleep(5)
        except StreamManagerException:
            logger.exception("Exception while running")
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python-SDK-Referenz: read_messages | StatusMessage

Java

```
import com.amazonaws.greengrass.streammanager.client.StreamManagerClient;
import com.amazonaws.greengrass.streammanager.client.StreamManagerClientFactory;
import com.amazonaws.greengrass.streammanager.client.utils.ValidateAndSerialize;
import com.amazonaws.greengrass.streammanager.model.ReadMessagesOptions;
import com.amazonaws.greengrass.streammanager.model.Status;
import com.amazonaws.greengrass.streammanager.model.StatusConfig;
import com.amazonaws.greengrass.streammanager.model.StatusLevel;
import com.amazonaws.greengrass.streammanager.model.StatusMessage;

 try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().build()) {
    try {
        boolean isS3UploadComplete = false;
        while (!isS3UploadComplete) {
            try {
                // Read the statuses from the export status stream
                List<Message> messages = client.readMessages("StatusStreamName",
                    new
 ReadMessagesOptions().withMinMessageCount(1L).withReadTimeoutMillis(1000L));
                for (Message message : messages) {
                    // Deserialize the status message first.
```

```
                    StatusMessage statusMessage =
 ValidateAndSerialize.deserializeJsonBytesToObj(message.getPayload(),
 StatusMessage.class);
                    // Check the status of the status message. If the status is
 "Success", the file was successfully uploaded to S3.
                    // If the status was either "Failure" or "Canceled", the server
 was unable to upload the file to S3.
                    // We will print the message for why the upload to S3 failed
 from the status message.
                    // If the status was "InProgress", the status indicates that the
 server has started uploading the S3 task.
                    if (Status.Success.equals(statusMessage.getStatus())) {
                        System.out.println("Successfully uploaded file at path " +
 FILE_URL + " to S3.");
                        isS3UploadComplete = true;
                     } else if (Status.Failure.equals(statusMessage.getStatus()) ||
 Status.Canceled.equals(statusMessage.getStatus())) {
                        System.out.println(String.format("Unable to upload file at
 path %s to S3. Message %s",

 statusMessage.getStatusContext().getS3ExportTaskDefinition().getInputUrl(),
                            statusMessage.getMessage()));
                        sS3UploadComplete = true;
                    }
                }
            } catch (StreamManagerException ignored) {
            } finally {
                // Sleep for sometime for the S3 upload task to complete before
 trying to read the status message.
                Thread.sleep(5000);
            }
        } catch (e) {
        // Properly handle errors.
    }
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java-SDK-Referenz: readMessages | StatusMessage

Node.js

```
const {
    StreamManagerClient, ReadMessagesOptions,
```

```
    Status, StatusConfig, StatusLevel, StatusMessage,
    util,
} = require(*'aws-greengrass-stream-manager-sdk'*);

const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        let isS3UploadComplete = false;
        while (!isS3UploadComplete) {
            try {
                // Read the statuses from the export status stream
                const messages = await c.readMessages("StatusStreamName",
                    new ReadMessagesOptions()
                        .withMinMessageCount(1)
                        .withReadTimeoutMillis(1000));

                messages.forEach((message) => {
                    // Deserialize the status message first.
                    const statusMessage =
 util.deserializeJsonBytesToObj(message.payload, StatusMessage);
                    // Check the status of the status message. If the status is
 'Success', the file was successfully uploaded to S3.
                    // If the status was either 'Failure' or 'Cancelled', the server
 was unable to upload the file to S3.
                    // We will print the message for why the upload to S3 failed
 from the status message.
                    // If the status was "InProgress", the status indicates that the
 server has started uploading the S3 task.
                    if (statusMessage.status === Status.Success) {
                        console.log(`Successfully uploaded file at path ${FILE_URL}
 to S3.`);
                        isS3UploadComplete = true;
                    } else if (statusMessage.status === Status.Failure ||
 statusMessage.status === Status.Canceled) {
                        console.log(`Unable to upload file at path ${FILE_URL} to
 S3. Message: ${statusMessage.message}`);
                        isS3UploadComplete = true;
                    }
                });
                // Sleep for sometime for the S3 upload task to complete before
 trying to read the status message.
                await new Promise((r) => setTimeout(r, 5000));
            } catch (e) {
                // Ignored
```

```
            }
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Node.js SDK-Referenz: readMessages | StatusMessage

Konfigurieren des AWS IoT Greengrass-Stream-Managers

Auf Greengrass-Core-Geräten kann Stream Manager IoT-Gerätedaten speichern, verarbeiten und exportieren. Stream Manager stellt Parameter bereit, mit denen Sie Laufzeiteinstellungen konfigurieren. Diese Einstellungen gelten für alle Streams auf dem Greengrass-Core-Gerät. Sie können die AWS IoT Greengrass Konsole oder API verwenden, um Stream-Manager-Einstellungen zu konfigurieren, wenn Sie die Komponente bereitstellen. Änderungen werden nach Abschluss der Bereitstellung wirksam.

Stream-Manager-Parameter

Stream Manager stellt die folgenden Parameter bereit, die Sie konfigurieren können, wenn Sie die Komponente auf Ihren -Core-Geräten bereitstellen. Alle Parameter sind optional.

Speicherverzeichnis

Parametername: `STREAM_MANAGER_STORE_ROOT_DIR`

Der absolute Pfad des lokalen Ordners, der zum Speichern von Streams verwendet wird. Dieser Wert muss mit einem Schrägstrich (z. B. `/data`) beginnen.

Sie müssen einen vorhandenen Ordner angeben, und der Systembenutzer, der die Stream-Manager-Komponente ausführt, muss über Lese- und Schreibberechtigungen für diesen Ordner verfügen. Sie können beispielsweise die folgenden Befehle ausführen, um einen Ordner, , zu erstellen und zu konfigurieren`/var/greengrass/streams`, den Sie als Stammordner des Stream-Managers angeben. Diese Befehle ermöglichen es dem Standardsystembenutzer, `ggc_user`, diesen Ordner zu lesen und in ihn zu schreiben.

```
sudo mkdir /var/greengrass/streams
sudo chown ggc_user /var/greengrass/streams
sudo chmod 700 /var/greengrass/streams
```

Hinweise zum Sichern von Streamdaten finden Sie unter the section called “Lokale Datensicherheit”.

Standard: *`/greengrass/v2`*`/work/aws.greengrass.StreamManager`

Server port

Parametername: `STREAM_MANAGER_SERVER_PORT`

Die lokale Portnummer, die für die Kommunikation mit dem Stream-Manager verwendet wird. Der Standardwert ist `8088`.

Sie können angeben`0`, um einen zufällig verfügbaren Port zu verwenden.

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 dem Stream-Manager wird vom Stream-Manager-SDK gesteuert. Dieser Parameter bestimmt, welche Clients das Stream Manager SDK aufrufen können, um mit Streams zu arbeiten. Weitere Informationen finden Sie unter the section called “Client-Authentifizierung”.

Gültige Werte sind `true` oder `false`. Der Standardwert ist `true` (empfohlen).

- `true`. Erlaubt nur Greengrass-Komponenten als Clients. Komponenten verwenden interne AWS IoT Greengrass Core-Protokolle, um sich beim Stream Manager SDK zu authentifizieren.
- `false`. Ermöglicht jedem Prozess, der auf dem AWS IoT Greengrass Core ausgeführt wird, einen Client zu sein. Setzen Sie den Wert nicht auf , `false` es sei denn, Ihr Geschäftsfall erfordert dies. Verwenden Sie beispielsweise `false` nur, wenn Prozesse, die keine Komponenten sind, auf dem Core-Gerät direkt mit dem Stream-Manager kommunizieren müssen.

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.

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. Versuchen Sie daher, die Anzahl der Exportstreams auf eingeschränkten Geräten zu begrenzen.

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.

Protokollierungsstufe

Parametername: `LOG_LEVEL`

Die Protokollierungsebene für die Komponente. Wählen Sie aus den folgenden Protokollebenen aus, die hier in der Reihenfolge der Ebenen aufgeführt sind:

- `TRACE`
- `DEBUG`

- `INFO`
- `WARN`
- `ERROR`

Standard: `INFO`

Mindestgröße für mehrteilige Uploads

Parametername: `STREAM_MANAGER_EXPORTER_S3_DESTINATION_MULTIPART_UPLOAD_MIN_PART_SIZE_BYTES`

Die Mindestgröße (in Byte) eines Teils eines mehrteiligen Uploads in Amazon S3. Stream Manager verwendet diese Einstellung und die Größe der Eingabedatei, um zu bestimmen, wie Daten in einer mehrteiligen PUT-Anforderung gebündelt werden. Der Standardwert und der Mindestwert sind `5242880` Byte (5 MB).

> **Note**
>
> Stream Manager verwendet die `-sizeThresholdForMultipartUploadBytes`Eigenschaft des Streams, um zu bestimmen, ob als einteiliger oder mehrteiliger Upload nach Amazon S3 exportiert werden soll. Benutzerdefinierte Greengrass-Komponenten legen diesen Schwellenwert fest, wenn sie einen Stream erstellen, der nach Amazon S3 exportiert wird. Der Standardschwellenwert ist 5 MB.

Weitere Informationen finden Sie auch unter

- Verwalten von Datenströmen auf Greengrass-Core-Geräten
- Verwenden von StreamManagerClient für die Arbeit mit Streams
- Exportkonfigurationen für unterstützte AWS Cloud Ziele

Durchführen von Machine Learning-Inferenzen

Mit können Sie mithilfe von Cloud-trainierten AWS IoT GreengrassModellen Machine Learning (ML)-Inferenzen auf Ihren Edge-Geräten für lokal generierte Daten 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.

AWS IoT Greengrass macht die erforderlichen Schritte zur Durchführung von Inferenzen effizienter. Sie können Ihre Inferenzmodelle überall trainieren und lokal als Machine-Learning-Komponenten bereitstellen. Sie können beispielsweise Deep-Learning-Modelle in Amazon SageMaker- oder Computer-Vision-Modellen in Amazon Lookout for Vision erstellen und trainieren. Anschließend können Sie diese Modelle in einem Amazon S3-Bucket speichern, sodass Sie diese Modelle als Artefakte in Ihren Komponenten verwenden können, um Inferenzen auf Ihren Core-Geräten durchzuführen.

Themen

Funktionsweise der AWS IoT Greengrass-ML-Inferenz

AWS bietet Machine-Learning-Komponenten, mit denen Sie in einem Schritt Bereitstellungen erstellen können, um Machine-Learning-Inferenzen auf Ihrem Gerät durchzuführen. Sie können diese Komponenten auch als Vorlagen verwenden, um benutzerdefinierte Komponenten zu erstellen, die Ihren spezifischen Anforderungen entsprechen.

AWS bietet die folgenden Kategorien von Machine Learning-Komponenten:

- Modellkomponente – Enthält Machine-Learning-Modelle als Greengrass-Artefakte.
- Laufzeitkomponente – Enthält das Skript, das das Machine Learning-Framework installiert, und seine Abhängigkeiten auf dem Greengrass-Kerngerät.
- Inferenzkomponente – Enthält den Inferenzcode und enthält Komponentenabhängigkeiten, um das Machine-Learning-Framework zu installieren und vortrainierte Machine-Learning-Modelle herunterzuladen.

Jede Bereitstellung, die Sie erstellen, um Machine Learning-Inferenzen durchzuführen, besteht aus mindestens einer Komponente, die Ihre Inferenzanwendung ausführt, das Machine Learning-Framework installiert und Ihre Machine Learning-Modelle herunterlädt. Um eine Beispielinferenz mit von bereitgestellten Komponenten durchzuführen, stellen Sie eine Inferenzkomponente auf Ihrem CoreAWS-Gerät bereit, die automatisch die entsprechenden Modell- und Laufzeitkomponenten als Abhängigkeiten enthält. Um Ihre Bereitstellungen anzupassen, können Sie die Beispielmodellkomponenten mit benutzerdefinierten Modellkomponenten verbinden oder austauschen, oder Sie können die Komponentenrezepte für die von bereitgestellten Komponenten als Vorlagen verwenden, um Ihre eigenen benutzerdefinierten InferenzAWS-, Modell- und Laufzeitkomponenten zu erstellen.

So führen Sie Machine Learning-Inferenzen mithilfe benutzerdefinierter Komponenten durch:

1. Erstellen Sie eine Modellkomponente. Diese Komponente enthält die Machine-Learning-Modelle, die Sie für die Inferenz verwenden möchten. AWS bietet Beispiele für vortrainierte DLR- und TensorFlow Lite-Modelle. Um ein benutzerdefiniertes Modell zu verwenden, erstellen Sie Ihre eigene Modellkomponente.
2. Erstellen Sie eine Laufzeitkomponente. Diese Komponente enthält die Skripts, die für die Installation der Machine-Learning-Laufzeit für Ihre Modelle erforderlich sind. AWS bietet Beispiellaufzeitkomponenten für Deep Learning Runtime (DLR) und TensorFlow Lite . Um andere Laufzeiten mit Ihren benutzerdefinierten Modellen und Ihrem Inferenzcode zu verwenden, erstellen Sie Ihre eigenen Laufzeitkomponenten.
3. Erstellen Sie eine Inferenzkomponente. Diese Komponente enthält Ihren Inferenzcode und enthält Ihre Modell- und Laufzeitkomponenten als Abhängigkeiten. AWS bietet Beispielinferenzkomponenten für die Bildklassifizierung und Objekterkennung mit DLR und TensorFlow Lite. Um andere Arten von Inferenzen durchzuführen oder benutzerdefinierte Modelle und Laufzeiten zu verwenden, erstellen Sie Ihre eigene Inferenzkomponente.

4. Stellen Sie die Inferenzkomponente bereit. Wenn Sie diese Komponente bereitstellen, stellt AWS IoT Greengrass auch automatisch die Abhängigkeiten der Modell- und Laufzeitkomponenten bereit.

Informationen zu den ersten Schritten mit AWSvon bereitgestellten Komponenten finden Sie unter the section called “Durchführen einer Inferenz der Bildklassifizierung”.

Informationen zum Erstellen von benutzerdefinierten Machine-Learning-Komponenten finden Sie unter Anpassen Ihrer Machine-Learning-Komponenten.

Was ist in AWS IoT Greengrass Version 2 anders?

AWS IoT Greengrass fasst Funktionseinheiten für Machine Learning – wie Modelle, Laufzeiten und Inferenzcode – in Komponenten zusammen, mit denen Sie die Machine-Learning-Laufzeit in einem Schritt installieren, Ihre trainierten Modelle herunterladen und Inferenzen auf Ihrem Gerät durchführen können.

Durch die Verwendung der von bereitgestellten Machine-LearningAWS-Komponenten haben Sie die Flexibilität, mit der Durchführung von Machine-Learning-Inferenzen mit Beispiel-Inferenzcode und vortrainierten Modellen zu beginnen. Sie können benutzerdefinierte Modellkomponenten einbinden, um Ihre eigenen benutzerdefinierten trainierten Modelle mit den von AWSbereitgestellten Inferenz- und Laufzeitkomponenten zu verwenden. Für eine vollständig angepasste Machine-Learning-Lösung können Sie die öffentlichen Komponenten als Vorlagen verwenden, um benutzerdefinierte Komponenten zu erstellen und beliebige Laufzeiten, Modelle oder Inferenztypen zu verwenden.

Voraussetzungen

Um Machine-Learning-Komponenten zu erstellen und zu verwenden, benötigen Sie Folgendes:

- Ein Greengrass-Core-Gerät. Falls Sie noch keines haben, beachten Sie die Informationen unter Tutorial: Erste Schritte mit AWS IoT Greengrass V2.
- Mindestens 500 MB lokaler Speicherplatz für die Verwendung von von bereitgestellten Machine-LearningAWS-Beispielkomponenten.

Unterstützte Modellquellen

AWS IoT Greengrass unterstützt die Verwendung von benutzerdefinierten Machine-Learning-Modellen, die in Amazon S3 gespeichert sind. Sie können auch Amazon SageMaker -Edge-Paketerstellungsaufträge verwenden, um direkt Modellkomponenten für Ihre SageMaker Neo-kompilierten Modelle zu erstellen. Informationen zur Verwendung von SageMaker Edge Manager mit AWS IoT Greengrassfinden Sie unter Verwenden von Amazon SageMaker Edge Manager auf Greengrass-Core-Geräten. Sie können auch Amazon Lookout for Vision-Modellpaketierungsaufträge verwenden, um Modellkomponenten für Ihre Lookout for Vision-Modelle zu erstellen. Weitere Informationen zur Verwendung von Lookout for Vision mit finden Sie AWS IoT Greengrassunter Amazon Lookout for Vision.

Die S3-Buckets, die Ihre Modelle enthalten, müssen die folgenden Anforderungen erfüllen:

- Sie dürfen nicht mit SSE-C verschlüsselt werden. Für Buckets, die serverseitige Verschlüsselung verwenden, unterstützt die Inferenz für AWS IoT Greengrass Machine Learning derzeit nur die Verschlüsselungsoptionen SSE-S3 oder SSE-KMS. Weitere Informationen zu serverseitigen Verschlüsselungsoptionen finden Sie unter Schutz von Daten durch serverseitige Verschlüsselung im Benutzerhandbuch für Amazon Simple Storage Service.
- Ihre Namen dürfen keine Punkte () enthalten`.`. Weitere Informationen finden Sie in der Regel zur Verwendung von Buckets im virtuellen Hosting-Stil mit SSL unter Regeln für die Bucket-Benennung im Benutzerhandbuch für Amazon Simple Storage Service.
- Die S3-Buckets, die Ihre Modellquellen speichern, müssen sich in derselben AWS-Konto und in derselben AWS-Region wie Ihre Machine-Learning-Komponenten befinden.
- AWS IoT Greengrass muss über die `read` Berechtigung für die Modellquelle verfügen. Damit AWS IoT Greengrass auf die S3-Buckets zugreifen kann, muss die Greengrass-Geräterolle die `s3:GetObject` Aktion zulassen. Weitere Informationen zur Geräterolle finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

Unterstützte Machine-Learning-Laufzeiten

AWS IoT Greengrass Mit können Sie benutzerdefinierte Komponenten erstellen, um eine beliebige Machine-Learning-Laufzeitumgebung Ihrer Wahl zu verwenden, um Machine-Learning-Inferenzen mit Ihren benutzerdefinierten trainierten Modellen durchzuführen. Informationen zum Erstellen von benutzerdefinierten Machine-Learning-Komponenten finden Sie unter Anpassen Ihrer Machine-Learning-Komponenten.

Um die ersten Schritte mit Machine Learning effizienter zu gestalten, AWS IoT Greengrass bietet Beispielkomponenten für Inferenz, Modell und Laufzeit, die die folgenden Machine-Learning-Laufzeiten verwenden:

- Deep Learning Runtime (DLR) v1.6.0 und v1.3.0
- TensorFlow Lite v2.5.0

AWSVon bereitgestellte Machine-Learning-Komponenten

In der folgenden Tabelle sind die AWSvon bereitgestellten Komponenten aufgeführt, die für Machine Learning verwendet werden.

> **Note**
>
> Mehrere AWSvon bereitgestellte Komponenten hängen von bestimmten Nebenversionen des Greengrass-Kerns ab. Aufgrund dieser Abhängigkeit müssen Sie diese Komponenten aktualisieren, wenn Sie den Greengrass-Kern auf eine neue Nebenversion aktualisieren. Informationen zu den spezifischen Versionen des Kerns, von denen jede Komponente abhängt, finden Sie im entsprechenden Komponententhema. Weitere Informationen zum Aktualisieren des Kerns finden Sie unter Aktualisieren der AWS IoT Greengrass Core-Software (OTA).

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
Lookout für Vision Edge Agent	Stellt die Laufzeit von Amazon Lookout for Vision auf dem Greengrass-Kerngerät bereit, sodass Sie Computer	Generisch	Linux	Nein

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
	Vision verwenden können, um Fehler in Industrieprodukten zu finden.			
SageMaker Edge-Manager	Stellt den Amazon SageMaker Edge Manager-Agenten auf dem Greengrass-Kerngerät bereit.	Generisch	Linux, Windows	Nein

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
DLR-Bildklassifizierung	Inferenzkomponente, die den DLR-Bildklassifizierungsmodellspeicher und die DLR-Laufzeitkomponente als Abhängigkeiten verwendet, um DLR zu installieren, Beispiel-Bildklassifizierungsmodelle herunterzuladen und Bildklassifizierungsinferenzen auf unterstützten Geräten durchzuführen.	Generisch	Linux, Windows	Nein

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
DLR-Objekterkennung	Inferenzkomponente, die den DLR-Objekterkennungsmodellspeicher und die DLR-Laufzeitkomponente als Abhängigkeiten verwendet, um DLR zu installieren, Beispiel-Objekterkennungsmodelle herunterzuladen und Objekterkennungs-Inferenzen auf unterstützten Geräten durchzuführen.	Generisch	Linux, Windows	Nein

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
Modellspeicher für die DLR-Bildklassifizierung	Modellkomponente, die Beispielmodelle zur Bildklassifizierung von ResNet-50 als Greengrass-Artefakte enthält.	Generisch	Linux, Windows	Nein
Modellspeicher für DLR-Objekterkennung	Modellkomponente, die YOLOv3-Beispielobjekterkennungsmodelle als Greengrass-Artefakte enthält.	Generisch	Linux, Windows	Nein

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
DLR-Laufzeit	Laufzeitkomponente, die ein Installationsskript enthält, das zur Installation von DLR und seinen Abhängigkeiten auf dem Greengrass-Kerngerät verwendet wird.	Generisch	Linux, Windows	Nein

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
TensorFlow Lite-Bildklassifizierung	Inferenzkomponente, die den TensorFlow Lite-Bildklassifizierungsmodellspeicher und die TensorFlow Lite-Laufzeitkomponente als Abhängigkeiten verwendet, um TensorFlow Lite zu installieren, Beispielmodelle zur Bildklassifizierung herunterzuladen und Bildklassifizierungsinferenzen auf unterstützten Geräten durchzuführen.	Generisch	Linux, Windows	Nein

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
TensorFlow Lite-Objekterkennung	Inferenzkomponente, die den TensorFlow Lite-Objekterkennungsmodellspeicher und die TensorFlow Lite-Laufzeitkomponente als Abhängigkeiten verwendet, um TensorFlow Lite zu installieren, Beispiel-Objekterkennungsmodelle herunterzuladen und Objekterkennungs-Inferenzen auf unterstützten Geräten durchzuführen.	Generisch	Linux, Windows	Nein

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
TensorFlow Modellspeicher für Lite-Bildklassifizierung	Modellkomponente, die ein Beispiel MobileNet - v1-Modell als Greengrass-Artefakt enthält.	Generisch	Linux, Windows	Nein
TensorFlow Modellspeicher für Lite-Objekterkennung	Modellkomponente, die ein Single Shot Detection (SSD)-MobileNet Beispielmodell als Greengrass-Artefakt enthält.	Generisch	Linux, Windows	Nein

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
TensorFlow Lite-Laufzeit	Laufzeitkomponente, die ein Installationsskript enthält, das zur Installation von TensorFlow Lite und seinen Abhängigkeiten auf dem Greengrass-Kerngerät verwendet wird.	Generisch	Linux, Windows	Nein

Verwenden von Amazon SageMaker Edge Manager auf Greengrass-Core-Geräten

> **Important**
>
> SageMaker Edge Manager wird am 26. April 2024 eingestellt. Weitere Informationen zum weiteren Bereitstellen Ihrer Modelle auf Edge-Geräten finden Sie unter SageMaker Ende der Lebensdauer von Edge Manager.

Amazon SageMaker Edge Manager ist ein Software-Agent, der auf Edge-Geräten ausgeführt wird. SageMaker Edge Manager bietet Modellverwaltung für Edge-Geräte, sodass Sie von Amazon SageMaker Neo kompilierte Modelle direkt auf Greengrass-Core-Geräten verpacken und verwenden können. Mithilfe von SageMaker Edge Manager können Sie auch Modelleingabe- und Ausgabedaten

von Ihren -Core-Geräten erfassen und diese Daten AWS Cloud zur Überwachung und Analyse an den senden. Da SageMaker Edge Manager SageMaker Neo zur Optimierung Ihrer Modelle für Ihre Zielhardware verwendet, müssen Sie die DLR-Laufzeit nicht direkt auf Ihrem Gerät installieren. Auf Greengrass-Geräten SageMaker lädtEdge Manager keine lokalen AWS IoT Zertifikate und ruft nicht direkt den Endpunkt des AWS IoT Anmeldeinformationsanbieters auf. Stattdessen SageMaker verwendetEdge Manager den Token-Exchange-Service, um temporäre Anmeldeinformationen von einem TES-Endpunkt abzurufen.

In diesem Abschnitt wird beschrieben, wie SageMaker Edge Manager auf Greengrass-Core-Geräten funktioniert.

Funktionsweise von SageMaker Edge Manager auf Greengrass-Geräten

Um den SageMaker Edge-Manager-Agent auf Ihren Core-Geräten bereitzustellen, erstellen Sie eine Bereitstellung, die die -`aws.greengrass.SageMakerEdgeManager`Komponente enthält. AWS IoT Greengrass verwaltet die Installation und den Lebenszyklus des Edge-Manager-Agenten auf Ihren Geräten. Wenn eine neue Version der Agenten-Binärdatei verfügbar ist, stellen Sie die aktualisierte Version der `aws.greengrass.SageMakerEdgeManager` Komponente bereit, um die Version des Agenten zu aktualisieren, die auf Ihrem Gerät installiert ist.

Wenn Sie SageMaker Edge Manager mit verwendenAWS IoT Greengrass, umfasst Ihr Workflow die folgenden allgemeinen Schritte:

1. Kompilieren Sie Modelle mit SageMaker Neo.
2. Verpacken Sie Ihre SageMaker Neo-kompilierten Modelle mithilfe von SageMaker Edge-Paketerstellungsaufträgen. Wenn Sie einen Edge-Paketerstellungsauftrag für Ihr Modell ausführen, können Sie eine Modellkomponente mit dem verpackten Modell als Artefakt erstellen, das auf Ihrem Greengrass-Kerngerät bereitgestellt werden kann.
3. Erstellen Sie eine benutzerdefinierte Inferenzkomponente. Sie verwenden diese Inferenzkomponente, um mit dem Edge Manager-Agenten zu interagieren, um Inferenzen auf dem Core-Gerät durchzuführen. Zu diesen Vorgängen gehören das Laden von Modellen, das Aufrufen von Prognoseanfragen zur Ausführung von Inferenzen und das Entladen von Modellen, wenn die Komponente heruntergefahren wird.
4. Stellen Sie die SageMaker Edge Manager-Komponente, die verpackte Modellkomponente und die Inferenzkomponente bereit, um Ihr Modell auf der SageMaker Inferenz-Engine (Edge Manager-Agent) auf Ihrem Gerät auszuführen.

Weitere Informationen zum Erstellen von Edge-Paketerstellungsaufträgen und Inferenzkomponenten, die mit SageMaker Edge Manager funktionieren, finden Sie unter Modellpaket bereitstellen und Edge-Manager-Agent mit AWS IoT Greengrass im Amazon- SageMaker Entwicklerhandbuch.

Das Tutorial: Erste Schritte mit SageMaker Edge ManagerTutorial zeigt Ihnen, wie Sie den SageMaker Edge Manager-Agenten auf einem vorhandenen GreengrassAWS-Kerngerät einrichten und verwenden, indem Sie den von bereitgestellten Beispielcode verwenden, mit dem Sie Beispielinferenzen und Modellkomponenten erstellen können.

Wenn Sie SageMaker Edge Manager auf Greengrass-Core-Geräten verwenden, können Sie auch die Funktion zur Erfassung von Daten verwenden, um Beispieldaten in die hochzuladenAWS Cloud. Erfassen von Daten ist ein SageMaker Feature, mit dem Sie Inferenzeingaben, Inferenzergebnisse und zusätzliche Inferenzdaten für zukünftige Analysen in einen S3-Bucket oder ein lokales Verzeichnis hochladen. Weitere Informationen zur Verwendung von Erfassungsdaten mit SageMaker Edge Manager finden Sie unter Modell verwalten im Amazon- SageMaker Entwicklerhandbuch.

Voraussetzungen

Sie müssen die folgenden Anforderungen erfüllen, um den SageMaker Edge Manager-Agenten auf Greengrass-Core-Geräten verwenden zu können.

- Ein Greengrass-Core-Gerät, das auf Amazon Linux 2, einer Debian-basierten Linux-Plattform (x86_64 oder Armv8) oder Windows (x86_64) ausgeführt wird. Falls Sie noch keines haben, beachten Sie die Informationen unter Tutorial: Erste Schritte mit AWS IoT Greengrass V2.
- Python 3.6 oder höher, einschließlich `pip` für Ihre Version von Python, die auf Ihrem Core-Gerät installiert ist.
- Die Greengrass-Geräterolle, die wie folgt konfiguriert ist:
  - Eine Vertrauensstellung, die es `credentials.iot.amazonaws.com` und ermöglicht, die Rolle `sagemaker.amazonaws.com` zu übernehmen, wie im folgenden IAM-Richtlinienbeispiel gezeigt.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
       },
```

```
      "Action": "sts:AssumeRole"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sagemaker.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- Die von AmazonSageMakerEdgeDeviceFleetPolicy IAM verwaltete Richtlinie.
- Die `s3:PutObject` Aktion, wie im folgenden Beispiel für eine IAM-Richtlinie gezeigt.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

- Ein Amazon S3-Bucket, der im selben AWS-Konto und AWS-Region wie Ihr Greengrass-Kerngerät erstellt wurde. SageMaker Edge Manager benötigt einen S3-Bucket, um eine Edge-Geräteflotte zu erstellen und Beispieldaten aus der Ausführung von Inferenzen auf Ihrem Gerät zu speichern. Informationen zum Erstellen von S3-Buckets finden Sie unter Erste Schritte mit Amazon S3.
- Eine SageMaker Edge-Geräteflotte, die denselben AWS IoT Rollenalias wie Ihr Greengrass-Kerngerät verwendet. Weitere Informationen finden Sie unter Erstellen Sie eine Flotte von Edge-Geräten.
- Ihr Greengrass-Core-Gerät ist als Edge-Gerät in Ihrer SageMaker Edge-Geräteflotte registriert. Der Name des Edge-Geräts muss mit dem AWS IoT Objektnamen für Ihr Core-Gerät übereinstimmen. Weitere Informationen finden Sie unter Registrieren Sie Ihr Greengrass Core-Gerät.

Erste Schritte mit SageMaker Edge Manager

Sie können ein Tutorial abschließen, um mit der Verwendung von SageMaker Edge Manager zu beginnen. Das Tutorial zeigt Ihnen, wie Sie mit der Verwendung von SageMaker Edge Manager mit von bereitgestellten Beispielkomponenten auf einem vorhandenen CoreAWS-Gerät beginnen. Diese Beispielkomponenten verwenden die SageMaker Edge Manager-Komponente als Abhängigkeit, um den Edge Manager-Agenten bereitzustellen und Inferenzen mit vortrainierten Modellen durchzuführen, die mit SageMaker Neo kompiliert wurden. Weitere Informationen finden Sie unter Tutorial: Erste Schritte mit SageMaker Edge Manager.

Amazon Lookout for Vision

> **Note**
>
> AWS IoT Greengrassunterstützt diese Funktion derzeit nicht auf Windows-Core-Geräten.

Amazon Lookout for Vision ist einAWS-Service Tool, mit dem Sie visuelle Fehler in Industrieprodukten finden können. Es nutzt Computer Vision, um fehlende Komponenten in Industrieprodukten, ebenso wie Schäden an Fahrzeugen oder Strukturen, Unregelmäßigkeiten in Produktionslinien, fehlende Kondensatoren auf Leiterplatten und Defekte in Silizium-Wafern oder anderen physischen Artikeln, bei denen höchste Qualität wichtig ist. Lookout for Vision im Amazon Lookout for Vision Developer Guide.

Sie können Greengrass-Anwendungen erstellen, die die Lookout for Vision Vision-Inferenz verwenden, um visuelle Fehler auf Greengrass-Core-Geräten zu finden. Nachdem Sie einen Lookout for Vision Vision-Workflow auf einem Greengrass-Core-Gerät bereitgestellt haben, können Sie Computer Vision ohne Verbindung zum Lookout for Vision Vision-Dienst in der ausführenAWS Cloud. Um eine Greengrass-Anwendung zu erstellen, die Lookout for Vision verwendet, richten Sie die folgenden Greengrass-Komponenten ein und implementieren sie:

- Lookout for Vision Vision-Modellkomponenten — Enthält Lookout for Vision Vision-Modelle für maschinelles Lernen als Greengrass-Artefakte. Sie können die Lookout for Vision Vision-Konsole und die API verwenden, um Modellkomponenten zu generieren, die Ihre vortrainierten Modelle für maschinelles Lernen bündeln. Diese Komponenten sind private Greengrass-Komponenten in IhremAWS-Konto. Weitere Informationen finden Sie unter Erstellen eines Lookout for Vision Vision-Modells und Verpacken eines Lookout for Vision Vision-Modells im Amazon Lookout for Vision Vision-Entwicklerhandbuch.

- Komponente Lookout for Vision Edge Agent — Stellt einen lokalen Lookout for Vision Vision-Laufzeitserver bereit, der Computer Vision verwendet, um Anomalien mithilfe von von Ihnen bereitgestellten Machine-Learning-Modellen zu erkennen. Diese Komponente ist eine vonAWS -bereitgestellte Komponente. ???
- Lookout for Vision Client-Anwendungskomponente — Interagiert mit der Lookout for Vision Edge Agent-Komponente, um Bilder auf Anomalien hin zu verarbeiten. Sie können benutzerdefinierte Client-Anwendungskomponenten entwickeln, die Bilder und Videostreams an den lokalen Lookout for Vision Edge Agent senden und alle Anomalien melden, die die Machine-Learning-Modelle erkennen. Weitere Informationen finden Sie unter Writing a client application component und Lookout for Vision Edge Agent API reference im Amazon Lookout for Vision Developer Guide.

Weitere Informationen zur Erstellung, Konfiguration und Verwendung dieser Komponenten finden Sie unter Verwenden eines Lookout for Vision-Modells auf einem Edge-Gerät im Amazon Lookout for Vision Developer Guide.

Anpassen Ihrer Machine-Learning-Komponenten

In können Sie Machine-Learning-Beispielkomponenten konfigurierenAWS IoT Greengrass, um anzupassen, wie Sie Machine-Learning-Inferenzen auf Ihren Geräten mit den Inferenz-, Modell- und Laufzeitkomponenten als Bausteine durchführen. bietet Ihnen AWS IoT Greengrass auch die Flexibilität, die Beispielkomponenten als Vorlagen zu verwenden und bei Bedarf eigene benutzerdefinierte Komponenten zu erstellen. Sie können diesen modularen Ansatz kombinieren und anpassen, um Ihre Machine-Learning-Inferenzkomponenten auf folgende Weise anzupassen:

Verwenden von Inferenzkomponenten

- Ändern Sie die Konfiguration von Inferenzkomponenten, wenn Sie sie bereitstellen.
- Verwenden Sie ein benutzerdefiniertes Modell mit der Beispiel-Inferenzkomponente, indem Sie die Beispiel-Modellspeicherkomponente durch eine benutzerdefinierte Modellkomponente ersetzen. Ihr benutzerdefiniertes Modell muss mit derselben Laufzeit wie das Beispielmodell trainiert werden.

Verwenden von benutzerdefinierten Inferenzkomponenten

- Verwenden Sie benutzerdefinierten Inferenzcode mit den Beispielmodellen und Laufzeiten, indem Sie öffentliche Modellkomponenten und Laufzeitkomponenten als Abhängigkeiten von benutzerdefinierten Inferenzkomponenten hinzufügen.

- Erstellen und fügen Sie benutzerdefinierte Modellkomponenten oder Laufzeitkomponenten als Abhängigkeiten von benutzerdefinierten Inferenzkomponenten hinzu. Sie müssen benutzerdefinierte Komponenten verwenden, wenn Sie benutzerdefinierten Inferenzcode oder eine Laufzeit verwenden möchten, für die AWS IoT Greengrass keine Beispielkomponente bereitstellt.

Themen

Ändern der Konfiguration einer öffentlichen Inferenzkomponente

In der AWS IoT Greengrass Konsole zeigt die Komponentenseite die Standardkonfiguration dieser Komponente an. Die Standardkonfiguration der TensorFlow Lite-Bildklassifizierungskomponente sieht beispielsweise wie folgt aus:

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.TensorFlowLiteImageClassification:mqttproxy:1": {
        "policyDescription": "Allows access to publish via topic ml/tflite/image-
classification.",
        "operations": [
          "aws.greengrass#PublishToIoTCore"
        ],
        "resources": [
          "ml/tflite/image-classification"
        ]
      }
    }
  },
  "PublishResultsOnTopic": "ml/tflite/image-classification",
  "ImageName": "cat.jpeg",
  "InferenceInterval": 3600,
  "ModelResourceKey": {
    "model": "TensorFlowLite-Mobilenet"
  }
```

```
}
```

Wenn Sie eine öffentliche Inferenzkomponente bereitstellen, können Sie die Standardkonfiguration ändern, um Ihre Bereitstellung anzupassen. Informationen zu den verfügbaren Konfigurationsparametern für jede öffentliche Inferenzkomponente finden Sie im Komponententhema unter AWSVon bereitgestellte Machine-Learning-Komponenten.

In diesem Abschnitt wird beschrieben, wie Sie eine geänderte Komponente über die AWS IoT Greengrass Konsole bereitstellen. Informationen zum Bereitstellen von Komponenten mit der finden Sie AWS CLIunter Erstellen von Bereitstellungen.

So stellen Sie eine geänderte öffentliche Inferenzkomponente bereit (Konsole)

1. Melden Sie sich an der AWS IoT Greengrass-Konsole an.
2. Wählen Sie im Navigationsmenü Komponenten aus.
3. Wählen Sie auf der Seite Komponenten auf der Registerkarte Öffentliche Komponenten die Komponente aus, die Sie bereitstellen möchten.
4. Wählen Sie auf der Komponentenseite Bereitstellen aus.
5. Wählen Sie unter Zur Bereitstellung hinzufügen eine der folgenden Optionen aus:

   a. Um diese Komponente mit einer auf Ihrem Zielgerät vorhandenen Bereitstellung zusammenzuführen, wählen Sie Zu vorhandener Bereitstellung hinzufügen und wählen Sie dann die Bereitstellung aus, die Sie überarbeiten möchten.

   b. Um auf Ihrem Zielgerät eine neue Bereitstellung zu erstellen, wählen Sie Neue Bereitstellung erstellen aus. Wenn auf Ihrem Gerät bereits eine Bereitstellung vorhanden ist, ersetzt die Auswahl in diesem Schritt die vorhandene Bereitstellung.
6. Gehen Sie auf der Seite Ziel angeben wie folgt vor:

   a. Geben Sie unter Bereitstellungsinformationen den Anzeigenamen für Ihre Bereitstellung ein oder ändern Sie ihn.

   b. Wählen Sie unter Bereitstellungsziele ein Ziel für Ihre Bereitstellung aus und klicken Sie auf Weiter. Wenn Sie eine vorhandene Bereitstellung überarbeiten, können Sie das Bereitstellungsziel nicht ändern.
7. Überprüfen Sie auf der Seite Komponenten auswählen unter Öffentliche Komponenten, ob die Inferenzkomponente mit Ihrer geänderten Konfiguration ausgewählt ist, und wählen Sie Weiter aus.
8. Gehen Sie auf der Seite Komponenten konfigurieren wie folgt vor:

a. Wählen Sie die Inferenzkomponente und dann Komponente konfigurieren aus.

b. Geben Sie unter Konfigurationsaktualisierung die Konfigurationswerte ein, die Sie aktualisieren möchten. Geben Sie beispielsweise das folgende Konfigurationsupdate in das Feld Konfiguration zum Zusammenführen ein, um das Inferenzintervall auf 15 Sekunden zu ändern, und weisen Sie die Komponente an, `custom.jpg` nach dem Image im `/custom-ml-inference/images/` Ordner zu suchen.

```
{
  "InferenceInterval": "15",
  "ImageName": "custom.jpg",
  "ImageDirectory": "/custom-ml-inference/images/"
}
```

Um die gesamte Konfiguration einer Komponente auf ihre Standardwerte zurückzusetzen, geben Sie eine einzelne leere Zeichenfolge `""` im Feld Pfade zurücksetzen an.

c. Wählen Sie Bestätigen aus, und wählen Sie dann Weiter.

9. Behalten Sie auf der Seite Konfigurieren erweiterter Einstellungen die Standardkonfigurationseinstellungen bei und wählen Sie Weiter aus.
10. Wählen Sie auf der Seite Review die Option Deploy aus.

Verwenden eines benutzerdefinierten Modells mit der Beispiel-Inferenzkomponente

Wenn Sie die Beispiel-Inferenzkomponente mit Ihren eigenen Machine-Learning-Modellen für eine Laufzeit verwenden möchten, für die eine Beispiel-Laufzeitkomponente AWS IoT Greengrass bereitstellt, müssen Sie die öffentlichen Modellkomponenten mit Komponenten überschreiben, die diese Modelle als Artefakte verwenden. Führen Sie auf hoher Ebene die folgenden Schritte aus, um ein benutzerdefiniertes Modell mit der Beispiel-Inferenzkomponente zu verwenden:

1. Erstellen Sie eine Modellkomponente, die ein benutzerdefiniertes Modell in einem S3-Bucket als Artefakt verwendet. Ihr benutzerdefiniertes Modell muss mit derselben Laufzeit trainiert werden wie das Modell, das Sie ersetzen möchten.
2. Ändern Sie den `ModelResourceKey` Konfigurationsparameter in der Inferenzkomponente, um das benutzerdefinierte Modell zu verwenden. Informationen zum Aktualisieren der Konfiguration

der Inferenzkomponente finden Sie unter . Ändern der Konfiguration einer öffentlichen Inferenzkomponente

Wenn Sie die Inferenzkomponente bereitstellen, AWS IoT Greengrasssucht nach der neuesten Version ihrer Komponentenabhängigkeiten. Sie überschreibt die abhängige öffentliche Modellkomponente, wenn eine spätere benutzerdefinierte Version der Komponente im selben AWS-Konto und in derselben vorhanden istAWS-Region.

Erstellen einer benutzerdefinierten Modellkomponente (Konsole)

1. Laden Sie Ihr Modell in einen S3-Bucket hoch. Informationen zum Hochladen Ihrer Modelle in einen S3-Bucket finden Sie unter Arbeiten mit Amazon S3-Buckets im Benutzerhandbuch für Amazon Simple Storage Service.

   > **Note**
   >
   > Sie müssen Ihre Artefakte in S3-Buckets speichern, die sich im selben AWS-Konto und in derselben AWS-Region wie die Komponenten befinden. Damit AWS IoT Greengrass auf diese Artefakte zugreifen kann, muss die Greengrass-Geräterolle die `s3:GetObject` Aktion zulassen. Weitere Informationen zur Geräterolle finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

2. Wählen Sie im Navigationsmenü der AWS IoT GreengrassKonsole Komponenten aus.
3. Rufen Sie das Komponentenrezept für die Komponente des öffentlichen Modellspeichers ab.

   a. Suchen Sie auf der Seite Komponenten auf der Registerkarte Öffentliche Komponenten nach der öffentlichen Modellkomponente, für die Sie eine neue Version erstellen möchten, und wählen Sie sie aus. Beispiel: `variant.DLR.ImageClassification.ModelStore`

   b. Wählen Sie auf der Komponentenseite Rezept anzeigen und kopieren Sie das angezeigte JSON-Rezept.
4. Wählen Sie auf der Seite Komponenten auf der Registerkarte Meine Komponenten die Option Komponente erstellen aus.
5. Wählen Sie auf der Seite Komponente erstellen unter Komponenteninformationen die Option Rezept als JSON als Komponentenquelle eingeben aus.
6. Fügen Sie im Feld Rezept das zuvor kopierte Komponentenrezept ein.
7. Aktualisieren Sie im Rezept die folgenden Werte:

- `ComponentVersion`: Erhöhen Sie die Nebenversion der Komponente.

  Wenn Sie eine benutzerdefinierte Komponente erstellen, um eine öffentliche Modellkomponente zu überschreiben, dürfen Sie nur die Nebenversion der vorhandenen Komponentenversion aktualisieren. Wenn die öffentliche Komponentenversion beispielsweise ist`2.1.0`, können Sie eine benutzerdefinierte Komponente mit Version erstellen`2.1.1`.
- `Manifests.Artifacts.Uri`: Aktualisieren Sie jeden URI-Wert auf den Amazon S3-URI des Modells, das Sie verwenden möchten.

> **Note**
>
> Ändern Sie nicht den Namen der Komponente.

8. Wählen Sie Komponente erstellen aus.

Erstellen einer benutzerdefinierten Modellkomponente (AWS CLI)

1. Laden Sie Ihr Modell in einen S3-Bucket hoch. Informationen zum Hochladen Ihrer Modelle in einen S3-Bucket finden Sie unter Arbeiten mit Amazon S3-Buckets im Benutzerhandbuch für Amazon Simple Storage Service.

   > **Note**
   >
   > Sie müssen Ihre Artefakte in S3-Buckets speichern, die sich im selben AWS-Konto und in derselben AWS-Region wie die Komponenten befinden. Damit AWS IoT Greengrass auf diese Artefakte zugreifen kann, muss die Greengrass-Geräterolle die `s3:GetObject` Aktion zulassen. Weitere Informationen zur Geräterolle finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

2. Führen Sie den folgenden Befehl aus, um das Komponentenrezept der öffentlichen Komponente abzurufen. Dieser Befehl schreibt das Komponentenrezept in die Ausgabedatei, die Sie in Ihrem Befehl angeben. Konvertieren Sie die abgerufene base64-kodierte Zeichenfolge nach Bedarf in JSON oder YAML.

   Linux, macOS, or Unix

   ```
   aws greengrassv2 get-component \
   ```

```
    --arn <arn> \
    --recipe-output-format <recipe-format> \
    --query recipe \
    --output text | base64 --decode > <recipe-file>
```

Windows Command Prompt (CMD)

```
aws greengrassv2 get-component ^
    --arn <arn> ^
    --recipe-output-format <recipe-format> ^
    --query recipe ^
    --output text > <recipe-file>.base64

certutil -decode <recipe-file>.base64 <recipe-file>
```

PowerShell

```
aws greengrassv2 get-component `
    --arn <arn> `
    --recipe-output-format <recipe-format> `
    --query recipe `
    --output text > <recipe-file>.base64

certutil -decode <recipe-file>.base64 <recipe-file>
```

3. Aktualisieren Sie den Namen der Rezeptdatei auf *<component-name>*-*<component-version>*, wobei die Komponentenversion die Zielversion der neuen Komponente ist. Beispiel: `variant.DLR.ImageClassification.ModelStore-2.1.1.yaml`
4. Aktualisieren Sie im Rezept die folgenden Werte:

   - `ComponentVersion`: Erhöhen Sie die Nebenversion der Komponente.

     Wenn Sie eine benutzerdefinierte Komponente erstellen, um eine öffentliche Modellkomponente zu überschreiben, dürfen Sie nur die Nebenversion der vorhandenen Komponentenversion aktualisieren. Wenn die öffentliche Komponentenversion beispielsweise ist`2.1.0`, können Sie eine benutzerdefinierte Komponente mit Version erstellen`2.1.1`.

   - `Manifests.Artifacts.Uri`: Aktualisieren Sie jeden URI-Wert auf den Amazon S3-URI des Modells, das Sie verwenden möchten.

> **Note**
>
> Ändern Sie nicht den Namen der Komponente.

5. Führen Sie den folgenden Befehl aus, um eine neue Komponente mit dem Rezept zu erstellen, das Sie abgerufen und geändert haben.

```
aws greengrassv2 create-component-version \
    --inline-recipe fileb://path/to/component/recipe
```

> **Note**
>
> In diesem Schritt wird die Komponente im AWS IoT Greengrass Service in der erstelltAWS Cloud. Sie können die Greengrass-CLI verwenden, um Ihre Komponente lokal zu entwickeln, zu testen und bereitzustellen, bevor Sie sie in die Cloud hochladen. Weitere Informationen finden Sie unter Entwickeln von AWS IoT Greengrass Komponenten.

Weitere Informationen zum Erstellen von Komponenten finden Sie unter Entwickeln von AWS IoT Greengrass Komponenten.

Erstellen von benutzerdefinierten Machine-Learning-Komponenten

Sie müssen benutzerdefinierte Komponenten erstellen, wenn Sie benutzerdefinierten Inferenzcode oder eine Laufzeit verwenden möchten, für die AWS IoT Greengrass keine Beispielkomponente bereitstellt. Sie können Ihren benutzerdefinierten Inferenzcode mit den von bereitgestellten Machine-LearningAWS-Beispielmodellen und Laufzeiten verwenden oder eine vollständig angepasste Machine-Learning-Inferenzlösung mit Ihren eigenen Modellen und Laufzeit entwickeln. Wenn Ihre Modelle eine Laufzeit verwenden, für die eine Beispiel-Laufzeitkomponente AWS IoT Greengrass bereitstellt, können Sie diese Laufzeitkomponente verwenden, und Sie müssen benutzerdefinierte Komponenten nur für Ihren Inferenzcode und die Modelle erstellen, die Sie verwenden möchten.

Themen

- Abrufen des Rezepts für eine öffentliche Komponente
- Abrufen von Beispielkomponentenartefakten

- Hochladen von Komponentenartefakten in einen S3-Bucket
- Erstellen von benutzerdefinierten Komponenten

Abrufen des Rezepts für eine öffentliche Komponente

Sie können das Rezept einer vorhandenen Komponente für öffentliches Machine Learning als Vorlage verwenden, um eine benutzerdefinierte Komponente zu erstellen. Um das Komponentenrezept für die neueste Version einer öffentlichen Komponente anzuzeigen, verwenden Sie die -Konsole oder die AWS CLI wie folgt:

- Verwenden der Konsole

  1. Suchen Sie auf der Seite Komponenten auf der Registerkarte Öffentliche Komponenten nach der öffentlichen Komponente und wählen Sie sie aus.
  2. Wählen Sie auf der Komponentenseite Rezept anzeigen aus.

- Verwenden von AWS CLI

  Führen Sie den folgenden Befehl aus, um das Komponentenrezept der öffentlichen Variantenkomponente abzurufen. Dieser Befehl schreibt das Komponentenrezept in die JSON- oder YAML-Rezeptdatei, die Sie in Ihrem Befehl angeben.

  Linux, macOS, or Unix

  ```
  aws greengrassv2 get-component \
      --arn <arn> \
      --recipe-output-format <recipe-format> \
      --query recipe \
      --output text | base64 --decode > <recipe-file>
  ```

  Windows Command Prompt (CMD)

  ```
  aws greengrassv2 get-component ^
      --arn <arn> ^
      --recipe-output-format <recipe-format> ^
      --query recipe ^
      --output text > <recipe-file>.base64

  certutil -decode <recipe-file>.base64 <recipe-file>
  ```

PowerShell

```
aws greengrassv2 get-component `
    --arn <arn> `
    --recipe-output-format <recipe-format> `
    --query recipe `
    --output text > <recipe-file>.base64

certutil -decode <recipe-file>.base64 <recipe-file>
```

Ersetzen Sie die Werte in Ihrem Befehl wie folgt:

- *<arn>*. Der Amazon-Ressourcenname (ARN) der öffentlichen Komponente.
- *<recipe-format>*. Das Format, in dem Sie die Rezeptdatei erstellen möchten. Unterstützte Werte sind `JSON` und `YAML`.
- *<recipe-file>*. Der Name des Rezepts im Format *<component-name>*-*<component-version>*.

Abrufen von Beispielkomponentenartefakten

Sie können die Artefakte, die von den öffentlichen Machine-Learning-Komponenten verwendet werden, als Vorlagen verwenden, um Ihre benutzerdefinierten Komponentenartefakte zu erstellen, z. B. Inferenzcode oder Laufzeitinstallationsskripts.

Um die Beispielartefakte anzuzeigen, die in den öffentlichen Machine-Learning-Komponenten enthalten sind, stellen Sie die öffentliche Inferenzkomponente bereit und zeigen Sie dann die Artefakte auf Ihrem Gerät im */greengrass/v2*`/packages/artifacts-unarchived/`*component-name*/*component-version*/ Ordner an.

Hochladen von Komponentenartefakten in einen S3-Bucket

Bevor Sie eine benutzerdefinierte Komponente erstellen können, müssen Sie die Komponentenartefakte in einen S3-Bucket hochladen und die S3-URIs in Ihrem Komponentenrezept verwenden. Um beispielsweise benutzerdefinierten Inferenzcode in Ihrer Inferenzkomponente zu verwenden, laden Sie den Code in einen S3-Bucket hoch. Anschließend können Sie den Amazon S3-URI Ihres Inferenzcodes als Artefakt in Ihrer Komponente verwenden.

Informationen zum Hochladen von Inhalten in einen S3-Bucket finden Sie unter Arbeiten mit Amazon S3-Buckets im Benutzerhandbuch für Amazon Simple Storage Service.

> **Note**
>
> Sie müssen Ihre Artefakte in S3-Buckets speichern, die sich im selben AWS-Konto und in derselben AWS-Region wie die Komponenten befinden. Damit AWS IoT Greengrass auf diese Artefakte zugreifen kann, muss die Greengrass-Geräterolle die `s3:GetObject` Aktion zulassen. Weitere Informationen zur Geräterolle finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

Erstellen von benutzerdefinierten Komponenten

Sie können die Artefakte und Rezepte verwenden, die Sie abgerufen haben, um Ihre benutzerdefinierten Machine-Learning-Komponenten zu erstellen. Ein Beispiel finden Sie unter Erstellen einer benutzerdefinierten Inferenzkomponente.

Ausführliche Informationen zum Erstellen und Bereitstellen von Komponenten auf Greengrass-Geräten finden Sie unter Entwickeln von AWS IoT Greengrass Komponenten und Bereitstellen von AWS IoT Greengrass Komponenten auf Geräten.

Erstellen einer benutzerdefinierten Inferenzkomponente

In diesem Abschnitt erfahren Sie, wie Sie eine benutzerdefinierte Inferenzkomponente mit der DLR-Bildklassifizierungskomponente als Vorlage erstellen.

Themen

- Laden Sie Ihren Inferenzcode in einen Amazon S3-Bucket hoch
- Erstellen eines Rezepts für Ihre Inferenzkomponente
- Erstellen der Inferenzkomponente

Laden Sie Ihren Inferenzcode in einen Amazon S3-Bucket hoch

Erstellen Sie Ihren Inferenzcode und laden Sie ihn dann in einen S3-Bucket hoch. Informationen zum Hochladen von Inhalten in einen S3-Bucket finden Sie unter Arbeiten mit Amazon S3-Buckets im Benutzerhandbuch für Amazon Simple Storage Service.

> **Note**
>
> Sie müssen Ihre Artefakte in S3-Buckets speichern, die sich im selben AWS-Konto und in derselben AWS-Region wie die Komponenten befinden. Damit AWS IoT Greengrass auf diese Artefakte zugreifen kann, muss die Greengrass-Geräterolle die `s3:GetObject` Aktion zulassen. Weitere Informationen zur Geräterolle finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

Erstellen eines Rezepts für Ihre Inferenzkomponente

1. Führen Sie den folgenden Befehl aus, um das Komponentenrezept der DLR-Bildklassifizierungskomponente abzurufen. Dieser Befehl schreibt das Komponentenrezept in die JSON- oder YAML-Rezeptdatei, die Sie in Ihrem Befehl angeben.

   Linux, macOS, or Unix

   ```
   aws greengrassv2 get-component \
       --arn
    arn:aws:greengrass:region:aws:components:aws.greengrass.DLRImageClassification:versions
    \
       --recipe-output-format JSON | YAML \
       --query recipe \
       --output text | base64 --decode > <recipe-file>
   ```

   Windows Command Prompt (CMD)

   ```
   aws greengrassv2 get-component ^
       --arn
    arn:aws:greengrass:region:aws:components:aws.greengrass.DLRImageClassification:versions
    ^
       --recipe-output-format JSON | YAML ^
       --query recipe ^
       --output text > <recipe-file>.base64

   certutil -decode <recipe-file>.base64 <recipe-file>
   ```

   PowerShell

   ```
   aws greengrassv2 get-component `
   ```

```
    --arn
 arn:aws:greengrass:region:aws:components:aws.greengrass.DLRImageClassification:versions
 `
    --recipe-output-format JSON | YAML `
    --query recipe `
    --output text > <recipe-file>.base64

certutil -decode <recipe-file>.base64 <recipe-file>
```

Ersetzen Sie *<recipe-file>* durch den Namen des Rezepts im Format *<component-name>*-*<component-version>*.

2. Führen Sie im -`ComponentDependencies`Objekt in Ihrem Rezept je nach Modell und Laufzeitkomponenten, die Sie verwenden möchten, einen oder mehrere der folgenden Schritte aus:

   - Behalten Sie die Abhängigkeit der DLR-Komponente bei, wenn Sie DLR-kompilierte Modelle verwenden möchten. Sie können sie auch durch eine Abhängigkeit von einer benutzerdefinierten Laufzeitkomponente ersetzen, wie im folgenden Beispiel gezeigt.

     Laufzeitkomponente

     JSON

     ```
     {
         "<runtime-component>": {
             "VersionRequirement": "<version>",
             "DependencyType": "HARD"
         }
     }
     ```

     YAML

     ```
     <runtime-component>:
         VersionRequirement: "<version>"
         DependencyType: HARD
     ```

   - Behalten Sie die Abhängigkeit des DLR-Bildklassifizierungsmodellspeichers bei, um die von AWSbereitgestellten vortrainierten ResNet-50-Modelle zu verwenden, oder ändern Sie sie, um eine benutzerdefinierte Modellkomponente zu verwenden. Wenn Sie eine Abhängigkeit für eine öffentliche Modellkomponente einschließen und eine spätere benutzerdefinierte Version

der Komponente in demselben AWS-Konto und vorhanden istAWS-Region, verwendet die Inferenzkomponente diese benutzerdefinierte Komponente. Geben Sie die Abhängigkeit der Modellkomponente an, wie in den folgenden Beispielen gezeigt.

Öffentliche Modellkomponente

JSON

```
{
    "variant.DLR.ImageClassification.ModelStore": {
        "VersionRequirement": "<version>",
        "DependencyType": "HARD"
    }
}
```

YAML

```
variant.DLR.ImageClassification.ModelStore:
    VersionRequirement: "<version>"
    DependencyType: HARD
```

Benutzerdefinierte Modellkomponente

JSON

```
{
    "<custom-model-component>": {
        "VersionRequirement": "<version>",
        "DependencyType": "HARD"
    }
}
```

YAML

```
<custom-model-component>:
    VersionRequirement: "<version>"
    DependencyType: HARD
```

3. Fügen Sie dem -`ComponentConfiguration`Objekt die Standardkonfiguration für diese Komponente hinzu. Sie können diese Konfiguration später ändern, wenn Sie die Komponente bereitstellen. Der folgende Auszug zeigt die Komponentenkonfiguration für die DLR-Bildklassifizierungskomponente.

Wenn Sie beispielsweise eine benutzerdefinierte Modellkomponente als Abhängigkeit für Ihre benutzerdefinierte Inferenzkomponente verwenden, ändern Sie , `ModelResourceKey` um die Namen der Modelle anzugeben, die Sie verwenden.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.ImageClassification:mqttproxy:1": {
        "policyDescription": "Allows access to publish via topic ml/dlr/image-
classification.",
        "operations": [
          "aws.greengrass#PublishToIoTCore"
        ],
        "resources": [
          "ml/dlr/image-classification"
        ]
      }
    }
  },
  "PublishResultsOnTopic": "ml/dlr/image-classification",
  "ImageName": "cat.jpeg",
  "InferenceInterval": 3600,
  "ModelResourceKey": {
    "armv7l": "DLR-resnet50-armv7l-cpu-ImageClassification",
    "x86_64": "DLR-resnet50-x86_64-cpu-ImageClassification",
    "aarch64": "DLR-resnet50-aarch64-cpu-ImageClassification"
  }
}
```

YAML

```
accessControl:
    aws.greengrass.ipc.mqttproxy:
        'aws.greengrass.ImageClassification:mqttproxy:1':
            policyDescription: 'Allows access to publish via topic ml/dlr/image-
classification.'
            operations:
                - 'aws.greengrass#PublishToIoTCore'
            resources:
                - ml/dlr/image-classification
```

```
PublishResultsOnTopic: ml/dlr/image-classification
ImageName: cat.jpeg
InferenceInterval: 3600
ModelResourceKey:
    armv7l: "DLR-resnet50-armv7l-cpu-ImageClassification"
    x86_64: "DLR-resnet50-x86_64-cpu-ImageClassification"
    aarch64: "DLR-resnet50-aarch64-cpu-ImageClassification"
```

4. Geben Sie im -`Manifests`Objekt Informationen über die Artefakte und die Konfiguration dieser Komponente an, die verwendet werden, wenn die Komponente auf verschiedenen Plattformen bereitgestellt wird, sowie alle anderen Informationen, die zum erfolgreichen Ausführen der Komponente erforderlich sind. Der folgende Auszug zeigt die Konfiguration des -`Manifests`Objekts für die Linux-Plattform in der DLR-Image-Klassifizierungskomponente.

   JSON

```
{
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "architecture": "arm"
      },
      "Name": "32-bit armv7l - Linux (raspberry pi)",
      "Artifacts": [
        {
          "URI": "s3://SAMPLE-BUCKET/sample-artifacts-directory/
image_classification.zip",
          "Unarchive": "ZIP"
        }
      ],
      "Lifecycle": {
        "Setenv": {
          "DLR_IC_MODEL_DIR":
 "{variant.DLR.ImageClassification.ModelStore:artifacts:decompressedPath}/
{configuration:/ModelResourceKey/armv7l}",
          "DEFAULT_DLR_IC_IMAGE_DIR": "{artifacts:decompressedPath}/
image_classification/sample_images/"
        },
        "run": {
          "RequiresPrivilege": true,
```

```
          "script": ". {variant.DLR:configuration:/MLRootPath}/
greengrass_ml_dlr_venv/bin/activate\npython3 {artifacts:decompressedPath}/
image_classification/inference.py"
        }
      }
    }
  ]
}
```

YAML

```
Manifests:
  - Platform:
      os: linux
      architecture: arm
    Name: 32-bit armv7l - Linux (raspberry pi)
    Artifacts:
      - URI: s3://SAMPLE-BUCKET/sample-artifacts-directory/
image_classification.zip
        Unarchive: ZIP
    Lifecycle:
      Setenv:
        DLR_IC_MODEL_DIR:
 "{variant.DLR.ImageClassification.ModelStore:artifacts:decompressedPath}/
{configuration:/ModelResourceKey/armv7l}"
        DEFAULT_DLR_IC_IMAGE_DIR: "{artifacts:decompressedPath}/
image_classification/sample_images/"
      run:
        RequiresPrivilege: true
        script: |-
          . {variant.DLR:configuration:/MLRootPath}/greengrass_ml_dlr_venv/bin/
activate
          python3 {artifacts:decompressedPath}/image_classification/inference.py
```

Ausführliche Informationen zum Erstellen von Komponentenrezepten finden Sie unter AWS IoT Greengrass Referenz zum Komponenten-Rezept.

Erstellen der Inferenzkomponente

Verwenden Sie die -AWS IoT GreengrassKonsole oder die AWS CLI, um eine Komponente mit dem soeben definierten Rezept zu erstellen. Nachdem Sie die Komponente erstellt haben, können Sie sie

bereitstellen, um Inferenzen auf Ihrem Gerät durchzuführen. Ein Beispiel für die Bereitstellung einer Inferenzkomponente finden Sie unter Tutorial: Durchführen einer Inferenz bei der Bildklassifizierung mit TensorFlow Lite.

Erstellen einer benutzerdefinierten Inferenzkomponente (Konsole)

1. Melden Sie sich an der AWS IoT Greengrass-Konsole an.
2. Wählen Sie im Navigationsmenü Komponenten aus.
3. Wählen Sie auf der Seite Komponenten auf der Registerkarte Meine Komponenten die Option Komponente erstellen aus.
4. Wählen Sie auf der Seite Komponente erstellen unter Komponenteninformationen entweder Rezept als JSON eingeben oder Rezept als YAML als Komponentenquelle eingeben aus.
5. Geben Sie im Feld Rezept das benutzerdefinierte Rezept ein, das Sie erstellt haben.
6. Klicken Sie auf Komponente erstellen.

Erstellen einer benutzerdefinierten Inferenzkomponente (AWS CLI)

Führen Sie den folgenden Befehl aus, um eine neue benutzerdefinierte Komponente mit dem von Ihnen erstellten Rezept zu erstellen.

```
aws greengrassv2 create-component-version \
    --inline-recipe fileb://path/to/recipe/file
```

> **Note**
>
> In diesem Schritt wird die Komponente im AWS IoT Greengrass Service in der erstelltAWS Cloud. Sie können die Greengrass-CLI verwenden, um Ihre Komponente lokal zu entwickeln, zu testen und bereitzustellen, bevor Sie sie in die Cloud hochladen. Weitere Informationen finden Sie unter Entwickeln von AWS IoT Greengrass Komponenten.

Fehlerbehebung bei Machine Learning-Inferenzen

Verwenden Sie die Informationen und Lösungen zur Fehlerbehebung in diesem Abschnitt, um Probleme mit Ihren Machine-Learning-Komponenten zu beheben. Informationen zu den öffentlichen Inferenzkomponenten für Machine Learning finden Sie in den Fehlermeldungen in den folgenden Komponentenprotokollen:

Linux or Unix

- */greengrass/v2*/logs/aws.greengrass.DLRImageClassification.log
- */greengrass/v2*/logs/aws.greengrass.DLRObjectDetection.log
- */greengrass/v2*/logs/aws.greengrass.TensorFlowLiteImageClassification.log
- */greengrass/v2*/logs/aws.greengrass.TensorFlowLiteObjectDetection.log

Windows

- *C:\greengrass\v2*\logs\aws.greengrass.DLRImageClassification.log
- *C:\greengrass\v2*\logs\aws.greengrass.DLRObjectDetection.log
- *C:\greengrass\v2*\logs\aws.greengrass.TensorFlowLiteImageClassification.log
- *C:\greengrass\v2*\logs\aws.greengrass.TensorFlowLiteObjectDetection.log

Wenn eine Komponente korrekt installiert ist, enthält das Komponentenprotokoll den Speicherort der Bibliothek, die es für die Inferenz verwendet.

Problembereiche

- Bibliothek konnte nicht abgerufen werden
- Cannot open shared object file
- Error: ModuleNotFoundError: No module named '<library>'
- Es wird kein CUDA-fähiges Gerät erkannt
- Keine solche Datei oder kein solches Verzeichnis
- RuntimeError: module compiled against API version 0xf but this version of NumPy is <version>
- picamera.exc.PiCameraError: Camera is not enabled
- Speicherfehler
- Festplattenspeicherfehler
- Timeout-Fehler

Bibliothek konnte nicht abgerufen werden

Der folgende Fehler tritt auf, wenn das Installationsskript während der Bereitstellung auf einem Raspberry Pi-Gerät keine erforderliche Bibliothek herunterladen kann.

```
Err:2 http://raspbian.raspberrypi.org/raspbian buster/main armhf python3.7-dev armhf
 3.7.3-2+deb10u1
404 Not Found [IP: 93.93.128.193 80]
E: Failed to fetch http://raspbian.raspberrypi.org/raspbian/pool/main/p/python3.7/
libpython3.7-dev_3.7.3-2+deb10u1_armhf.deb 404 Not Found [IP: 93.93.128.193 80]
```

Führen Sie Ihre Komponente erneut aus `sudo apt-get update` und stellen Sie sie erneut bereit.

Cannot open shared object file

Möglicherweise werden Fehler ähnlich den folgenden angezeigt, wenn das Installationsskript `opencv-python` während der Bereitstellung auf einem Raspberry Pi-Gerät keine erforderliche Abhängigkeit für herunterladen kann.

```
ImportError: libopenjp2.so.7: cannot open shared object file: No such file or directory
```

Führen Sie den folgenden Befehl aus, um die Abhängigkeiten für manuell zu installierenopencv-python:

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

Error: ModuleNotFoundError: No module named '<library>'

Dieser Fehler wird möglicherweise in den ML-Laufzeitkomponentenprotokollen (`variant.DLR.log` oder `variant.TensorFlowLite.log`) angezeigt, wenn die ML-Laufzeitbibliothek oder ihre Abhängigkeiten nicht korrekt installiert sind. Dieser Fehler kann in den folgenden Fällen auftreten:

- Wenn Sie die Option verwendenUseInstaller, die standardmäßig aktiviert ist, weist dieser Fehler darauf hin, dass die ML-Laufzeitkomponente die Laufzeit oder ihre Abhängigkeiten nicht installieren konnte. Gehen Sie wie folgt vor:

  1. Konfigurieren Sie die ML-Laufzeitkomponente, um die `UseInstaller` Option zu deaktivieren.

2. Installieren Sie die ML-Laufzeit und ihre Abhängigkeiten und stellen Sie sie dem Systembenutzer zur Verfügung, der die ML-Komponenten ausführt. Weitere Informationen finden Sie hier:
   - DLR- UseInstaller Laufzeitoption
   - TensorFlow Lite- UseInstaller Laufzeitoption

- Wenn Sie die `UseInstaller` Option nicht verwenden, weist dieser Fehler darauf hin, dass die ML-Laufzeit oder ihre Abhängigkeiten nicht für den Systembenutzer installiert sind, der die ML-Komponenten ausführt. Gehen Sie wie folgt vor:

  1. Überprüfen Sie, ob die Bibliothek für den Systembenutzer installiert ist, der die ML-Komponenten ausführt. Ersetzen Sie *`ggc_user`* durch den Namen des Systembenutzers und *`tf_runtime`* durch den Namen der zu überprüfenden Bibliothek.

     Linux or Unix

     ```
     sudo -H -u ggc_user bash -c "python3 -c 'import tflite_runtime'"
     ```

     Windows

     ```
     runas /user:ggc_user "py -3 -c \"import tflite_runtime\""
     ```

  2. Wenn die Bibliothek nicht installiert ist, installieren Sie sie für diesen Benutzer. Ersetzen Sie *`ggc_user`* durch den Namen des Systembenutzers und *`tf_runtime`* durch den Namen der Bibliothek.

     Linux or Unix

     ```
     sudo -H -u ggc_user bash -c "python3 -m pip install --user tflite_runtime"
     ```

     Windows

     ```
     runas /user:ggc_user "py -3 -m pip install --user tflite_runtime"
     ```

     Weitere Informationen zu den Abhängigkeiten für jede ML-Laufzeit finden Sie im Folgenden:
     - DLR- UseInstaller Laufzeitoption
     - TensorFlow Lite- UseInstaller Laufzeitoption

  3. Wenn das Problem weiterhin besteht, installieren Sie die Bibliothek für einen anderen Benutzer, um zu bestätigen, ob dieses Gerät die Bibliothek installieren kann. Der Benutzer könnte

beispielsweise Ihr Benutzer, der Root-Benutzer oder ein Administratorbenutzer sein. Wenn Sie die Bibliothek für keinen Benutzer erfolgreich installieren können, unterstützt Ihr Gerät die Bibliothek möglicherweise nicht. Lesen Sie die Dokumentation der Bibliothek, um die Anforderungen zu überprüfen und Installationsprobleme zu beheben.

Es wird kein CUDA-fähiges Gerät erkannt

Möglicherweise wird der folgende Fehler angezeigt, wenn Sie die GPU-Beschleunigung verwenden. Führen Sie den folgenden Befehl aus, um den GPU-Zugriff für den Greengrass-Benutzer zu aktivieren.

```
sudo usermod -a -G video ggc_user
```

Keine solche Datei oder kein solches Verzeichnis

Die folgenden Fehler weisen darauf hin, dass die Laufzeitkomponente die virtuelle Umgebung nicht korrekt einrichten konnte:

- *MLRootPath*`/greengrass_ml_dlr_conda/bin/conda: No such file or directory`
- *MLRootPath*`/greengrass_ml_dlr_venv/bin/activate: No such file or directory`
- *MLRootPath*`/greengrass_ml_tflite_conda/bin/conda: No such file or directory`
- *MLRootPath*`/greengrass_ml_tflite_venv/bin/activate: No such file or directory`

Überprüfen Sie die Protokolle, um sicherzustellen, dass alle Laufzeitabhängigkeiten korrekt installiert wurden. Weitere Informationen zu den vom Installationsskript installierten Bibliotheken finden Sie in den folgenden Themen:

- DLR-Laufzeit
- TensorFlow Lite-Laufzeit

Standardmäßig ist *MLRootPath* auf festgelegt*/greengrass/v2*`/work/`*component-name*`/greengrass_ml`. Um diesen Speicherort zu ändern, schließen Sie die - DLR-Laufzeit oder -TensorFlow Lite-LaufzeitLaufzeitkomponente direkt in Ihre Bereitstellung ein und geben Sie einen geänderten Wert für den -`MLRootPath`Parameter in einer

Konfigurationszusammenführungsaktualisierung an. Weitere Informationen zum Konfigurieren von Komponenten finden Sie unter Komponentenkonfigurationen aktualisieren.

> **Note**
>
> Für die DLR-Komponente v1.3.x legen Sie den `MLRootPath` Parameter in der Konfiguration der Inferenzkomponente fest und der Standardwert ist `$HOME/greengrass_ml`.

RuntimeError: module compiled against API version 0xf but this version of NumPy is <version>

Möglicherweise werden die folgenden Fehler angezeigt, wenn Sie Machine Learning-Inferenzen auf einem Raspberry Pi ausführen, auf dem Raspberry Pi OS Bullseye ausgeführt wird.

```
RuntimeError: module compiled against API version 0xf but this version of numpy is 0xd
ImportError: numpy.core.multiarray failed to import
```

Dieser Fehler tritt auf, weil Raspberry Pi OS Bullseye eine frühere Version von NumPy als die Version enthält, die OpenCV benötigt. Um dieses Problem zu beheben, führen Sie den folgenden Befehl aus, um NumPy auf die neueste Version zu aktualisieren.

```
pip3 install --upgrade numpy
```

picamera.exc.PiCameraError: Camera is not enabled

Möglicherweise wird der folgende Fehler angezeigt, wenn Sie Machine Learning-Inferenz auf einem Raspberry Pi ausführen, auf dem Raspberry Pi OS Bullseye ausgeführt wird.

```
picamera.exc.PiCameraError: Camera is not enabled. Try running 'sudo raspi-config' and
 ensure that the camera has been enabled.
```

Dieser Fehler tritt auf, weil Raspberry Pi OS Bullseye einen neuen Kamera-Stack enthält, der nicht mit den ML-Komponenten kompatibel ist. Um dieses Problem zu beheben, aktivieren Sie den Legacy-Kamera-Stack.

So aktivieren Sie den Legacy-Kamera-Stack

1. Führen Sie den folgenden Befehl aus, um das Raspberry Pi-Konfigurationstool zu öffnen.

```
sudo raspi-config
```

2. Wählen Sie Schnittstellenoptionen aus.
3. Wählen Sie Legacy-Kamera aus, um den Legacy-Kamera-Stack zu aktivieren.
4. Starten Sie den Raspberry Pi neu.

Speicherfehler

Die folgenden Fehler treten in der Regel auf, wenn das Gerät nicht über genügend Speicher verfügt und der Komponentenprozess unterbrochen wird.

- `stderr. Killed.`
- `exitCode=137`

Wir empfehlen mindestens 500 MB Arbeitsspeicher für die Bereitstellung einer öffentlichen Inferenzkomponente für Machine Learning.

Festplattenspeicherfehler

Der `no space left on device` Fehler tritt in der Regel auf, wenn ein Gerät nicht über genügend Speicherplatz verfügt. Stellen Sie sicher, dass auf Ihrem Gerät genügend Speicherplatz verfügbar ist, bevor Sie die Komponente erneut bereitstellen. Wir empfehlen mindestens 500 MB freien Speicherplatz für die Bereitstellung einer öffentlichen Inferenzkomponente für Machine Learning.

Timeout-Fehler

Die öffentlichen Machine-Learning-Komponenten laden große Machine-Learning-Modelldateien herunter, die größer als 200 MB sind. Wenn der Download während der Bereitstellung abläuft, überprüfen Sie die Geschwindigkeit Ihrer Internetverbindung und versuchen Sie die Bereitstellung erneut.

Verwalten von Greengrass-Core-Geräten mit AWS Systems Manager

> **Note**
>
> AWS IoT Greengrass unterstützt diese Funktion derzeit nicht auf Windows-Core-Geräten.

Systems Manager ist ein -AWSService, mit dem Sie Ihre Infrastruktur auf anzeigen und steuern könnenAWS, einschließlich Amazon EC2-Instances, On-Premises-Servern und virtuellen Maschinen (VMs) sowie Edge-Geräten. Systems Manager ermöglicht es Ihnen, Betriebsdaten anzuzeigen, Betriebsaufgaben zu automatisieren und Sicherheit und Compliance aufrechtzuerhalten. Wenn Sie eine Maschine bei Systems Manager registrieren, wird sie als verwalteter Knoten bezeichnet. Weitere Informationen finden Sie unter Was ist AWS Systems Manager? im AWS Systems Manager-Benutzerhandbuch.

Der AWS Systems Manager-Agent (Systems Manager Agent) ist Software, die Sie auf Geräten installieren können, damit Systems Manager sie aktualisieren, verwalten und konfigurieren kann. Um den Systems Manager Agent auf Greengrass-Core-Geräten zu installieren, stellen Sie die Systems Manager Agent-Komponente bereit. Wenn Sie den Systems Manager Agent zum ersten Mal bereitstellen, wird das Core-Gerät als verwalteter Systems Manager-Knoten registriert. Der Systems Manager Agent wird auf dem Gerät ausgeführt, um die Kommunikation mit dem Systems Manager-Service in der zu ermöglichenAWS Cloud. Weitere Informationen zum Installieren und Konfigurieren der Systems Manager Agent-Komponente finden Sie unter Installieren des AWS Systems Manager-Agenten.

Tools und Funktionen von Systems Manager werden als Funktionen bezeichnet. Greengrass-Core-Geräte unterstützen alle Systems Manager-Funktionen. Weitere Informationen zu diesen Funktionen und zur Verwendung von Systems Manager zur Verwaltung von -Core-Geräten finden Sie unter Systems Manager-Funktionen im AWS Systems Manager -Benutzerhandbuch.

AWS Systems Manager bietet ein Standard-Instances-Kontingent und ein Advanced-Instances-Kontingent für von Systems Manager verwaltete Knoten. Wenn Sie Systems Manager zum ersten Mal verwenden, beginnen Sie mit dem Kontingent für Standard-Instances. Auf dem Standard-Instances-Kontingent können Sie bis zu 1 000 verwaltete Knoten pro AWS-Region in Ihrem registrierenAWS-Konto. Wenn Sie mehr als 1 000 verwaltete Knoten in einem einzigen Konto und einer Region registrieren müssen oder wenn Sie die Session Manager-Funktion verwenden

müssen, verwenden Sie das Advanced-Instances-Kontingent. Weitere Informationen finden Sie unter Konfigurieren von Instance-Stufen im AWS Systems Manager -Benutzerhandbuch.

Themen

Installieren des AWS Systems Manager-Agenten

Der AWS Systems Manager -Agent (Systems Manager Agent) ist eine Amazon-Software, die Sie installieren, damit Systems Manager Greengrass-Core-Geräte, Amazon EC2-Instances und andere -Ressourcen aktualisieren, verwalten und konfigurieren kann. Der Agent verarbeitet und führt Anfragen vom Systems Manager-Service in der ausAWS Cloud. Anschließend sendet der Agent Status- und Laufzeitinformationen zurück an den Systems Manager-Service. Weitere Informationen finden Sie unter Informationen zu Systems Manager Agent im AWS Systems Manager -Benutzerhandbuch.

AWS stellt den Systems Manager Agent als Greengrass-Komponente bereit, die Sie auf Ihren Greengrass-Core-Geräten bereitstellen können, um sie mit Systems Manager zu verwalten. Die Systems Manager Agent-Komponente installiert die Systems Manager Agent-Software und registriert das Core-Gerät als verwalteten Knoten in Systems Manager. Führen Sie die Schritte auf dieser Seite aus, um die Voraussetzungen zu erfüllen und die Systems Manager Agent-Komponente auf einem Core-Gerät oder einer Gruppe von Core-Geräten bereitzustellen.

Themen

Schritt 1: Ausführen allgemeiner Systems Manager-Einrichtungsschritte

Wenn Sie dies noch nicht getan haben, führen Sie die allgemeinen Einrichtungsschritte für ausAWS Systems Manager. Weitere Informationen finden Sie unter Ausführen allgemeiner Systems Manager-Einrichtungsschritte im AWS Systems Manager -Benutzerhandbuch.

Schritt 2: Erstellen einer IAM-Servicerolle für Systems Manager

Der Systems Manager Agent verwendet eine AWS Identity and Access Management (IAM)-Servicerolle, um mit zu kommunizierenAWS Systems Manager. Systems Manager übernimmt diese Rolle, um Systems Manager-Funktionen auf jedem Core-Gerät zu aktivieren. Die Systems Manager Agent-Komponente verwendet diese Rolle auch, um das Core-Gerät als von Systems Manager verwalteten Knoten zu registrieren, wenn Sie die Komponente bereitstellen. Falls noch nicht geschehen, erstellen Sie eine Systems Manager-Servicerolle für die zu verwendende Systems Manager Agent-Komponente. Weitere Informationen finden Sie unter Erstellen einer IAM-Servicerolle für Edge-Geräte im AWS Systems Manager -Benutzerhandbuch.

Schritt 3: Hinzufügen von Berechtigungen zur Token-Exchange-Rolle

Greengrass-Core-Geräte verwenden eine IAM-Servicerolle, die als Token-Exchange-Rolle bezeichnet wird, um mit -AWSServices zu interagieren. Jedes Core-Gerät verfügt über eine Token-Austauschrolle, die Sie bei der Installation der AWS IoT Greengrass Core-Software erstellen. Viele Greengrass-Komponenten, wie z. B. der Systems Manager Agent, erfordern zusätzliche Berechtigungen für diese Rolle. Die Systems Manager-Agentenkomponente erfordert die folgenden Berechtigungen, die die Berechtigung zur Verwendung der Rolle enthalten, die Sie in erstellt habenSchritt 2: Erstellen einer IAM-Servicerolle für Systems Manager.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "iam:PassRole"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:iam::account-id:role/SSMServiceRole"
      ]
    },
    {
      "Action": [
        "ssm:AddTagsToResource",
        "ssm:RegisterManagedInstance"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
```

```
  ]
}
```

Wenn Sie dies noch nicht getan haben, fügen Sie diese Berechtigungen der Token-Austauschrolle des Core-Geräts hinzu, damit der Systems Manager Agent ausgeführt werden kann. Sie können der Token-Exchange-Rolle eine neue Richtlinie hinzufügen, um diese Berechtigung zu erteilen.

So fügen Sie der Token-Exchange-Rolle (Konsole) Berechtigungen hinzu

1. Wählen Sie im Navigationsmenü der IAM-Konsole Rollen aus.
2. Wählen Sie die IAM-Rolle aus, die Sie als Token-Austauschrolle bei der Installation der AWS IoT Greengrass Core-Software eingerichtet haben. Wenn Sie bei der Installation der AWS IoT Greengrass Core-Software keinen Namen für die Token-Exchange-Rolle angegeben haben, wurde eine Rolle mit dem Namen erstellt`GreengrassV2TokenExchangeRole`.
3. Wählen Sie unter Berechtigungen die Option Berechtigungen hinzufügen und dann Richtlinien anfügen aus.
4. Wählen Sie Richtlinie erstellen aus. Die Seite Richtlinie erstellen wird in einer neuen Browser-Registerkarte geöffnet.
5. Führen Sie auf der Seite Create policy (Richtlinie erstellen) die folgenden Schritte aus:

   a. Wählen Sie JSON, um den JSON-Editor zu öffnen.

   b. Fügen Sie die folgende -Richtlinie in den JSON-Editor ein. Ersetzen Sie *`SSMServiceRole`* durch den Namen der Servicerolle, die Sie in erstellt habenSchritt 2: Erstellen einer IAM-Servicerolle für Systems Manager.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "iam:PassRole"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:iam::account-id:role/SSMServiceRole"
      ]
    },
    {
      "Action": [
```

```
        "ssm:AddTagsToResource",
        "ssm:RegisterManagedInstance"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

c. Wählen Sie Next: Tags (Weiter: Tags) aus.

d. Klicken Sie auf Weiter: Prüfen.

e. Geben Sie unter Name einen Namen für die Richtlinie ein, z. B. **`GreengrassSSMAgentComponentPolicy`**.

f. Wählen Sie Richtlinie erstellen aus.

g. Wechseln Sie zur vorherigen Browser-Registerkarte, auf der Sie die Token-Exchange-Rolle geöffnet haben.

6. Wählen Sie auf der Seite Berechtigungen hinzufügen die Schaltfläche Aktualisieren und dann die Greengrass Systems Manager-Agentenrichtlinie aus, die Sie im vorherigen Schritt erstellt haben.
7. Wählen Sie Richtlinien anfügen.

   Die Core-Geräte, die diese Token-Exchange-Rolle verwenden, verfügen jetzt über die Berechtigung, mit dem Systems Manager-Service zu interagieren.

So fügen Sie der Token-Exchange-Rolle (AWS CLI) Berechtigungen hinzu

So fügen Sie eine Richtlinie hinzu, die die Berechtigung zur Verwendung von Systems Manager gewährt

1. Erstellen Sie eine Datei mit dem Namen `ssm-agent-component-policy.json` und kopieren Sie den folgenden JSON-Code in die Datei . Ersetzen Sie *`SSMServiceRole`* durch den Namen der Servicerolle, die Sie in erstellt habenSchritt 2: Erstellen einer IAM-Servicerolle für Systems Manager.

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Action": [
           "iam:PassRole"
   ```

```
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:iam::account-id:role/SSMServiceRole"
      ]
    },
    {
      "Action": [
        "ssm:AddTagsToResource",
        "ssm:RegisterManagedInstance"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

2. Führen Sie den folgenden Befehl aus, um die Richtlinie aus dem Richtliniendokument in zu erstellen`ssm-agent-component-policy.json`.

   Linux or Unix

   ```
   aws iam create-policy \
     --policy-name GreengrassSSMAgentComponentPolicy \
     --policy-document file://ssm-agent-component-policy.json
   ```

   Windows Command Prompt (CMD)

   ```
   aws iam create-policy ^
     --policy-name GreengrassSSMAgentComponentPolicy ^
     --policy-document file://ssm-agent-component-policy.json
   ```

   PowerShell

   ```
   aws iam create-policy `
     --policy-name GreengrassSSMAgentComponentPolicy `
     --policy-document file://ssm-agent-component-policy.json
   ```

Kopieren Sie den Amazon-Ressourcennamen (ARN) der Richtlinie aus den Richtlinienmetadaten in der Ausgabe. Sie verwenden diesen ARN, um diese Richtlinie im nächsten Schritt an die Rolle des Core-Geräts anzuhängen.

3. Führen Sie den folgenden Befehl aus, um die Richtlinie an die Token-Exchange-Rolle anzuhängen.

   - Ersetzen Sie *GreengrassV2TokenExchangeRole* durch den Namen der Token-Exchange-Rolle, die Sie bei der Installation der AWS IoT Greengrass Core-Software angegeben haben. Wenn Sie bei der Installation der AWS IoT Greengrass Core-Software keinen Namen für die Token-Exchange-Rolle angegeben haben, wurde eine Rolle mit dem Namen erstelltGreengrassV2TokenExchangeRole.
   - Ersetzen Sie den Richtlinien-ARN durch den ARN aus dem vorherigen Schritt.

   Linux or Unix

   ```
   aws iam attach-role-policy \
     --role-name GreengrassV2TokenExchangeRole \
     --policy-arn
    arn:aws:iam::123456789012:policy/GreengrassSSMAgentComponentPolicy
   ```

   Windows Command Prompt (CMD)

   ```
   aws iam attach-role-policy ^
     --role-name GreengrassV2TokenExchangeRole ^
     --policy-arn
    arn:aws:iam::123456789012:policy/GreengrassSSMAgentComponentPolicy
   ```

   PowerShell

   ```
   aws iam attach-role-policy `
     --role-name GreengrassV2TokenExchangeRole `
     --policy-arn
    arn:aws:iam::123456789012:policy/GreengrassSSMAgentComponentPolicy
   ```

Wenn der Befehl keine Ausgabe hat, war er erfolgreich. Die Core-Geräte, die diese Token-Exchange-Rolle verwenden, verfügen jetzt über die Berechtigung, mit dem Systems Manager-Service zu interagieren.

Schritt 4: Bereitstellen der Systems Manager Agent-Komponente

Führen Sie die folgenden Schritte aus, um die Systems Manager Agent-Komponente bereitzustellen und zu konfigurieren. Sie können die Komponente auf einem einzelnen Core-Gerät oder auf einer Gruppe von Core-Geräten bereitstellen.

So stellen Sie die Systems Manager Agent-Komponente bereit (Konsole)

1. Wählen Sie im Navigationsmenü der AWS IoT GreengrassKonsole Komponenten aus.
2. Wählen Sie auf der Seite Komponenten die Registerkarte Öffentliche Komponenten und dann ausaws.greengrass.SystemsManagerAgent.
3. Wählen Sie auf der aws.greengrass.SystemsManagerAgent Seite Bereitstellen aus.
4. Wählen Sie unter Zur Bereitstellung hinzufügen eine vorhandene Bereitstellung aus, die Sie überarbeiten möchten, oder wählen Sie aus, eine neue Bereitstellung zu erstellen, und wählen Sie dann Weiter aus.
5. Wenn Sie eine neue Bereitstellung erstellen möchten, wählen Sie das Ziel-Core-Gerät oder die Objektgruppe für die Bereitstellung aus. Wählen Sie auf der Seite Ziel angeben unter Bereitstellungsziel ein Core-Gerät oder eine Objektgruppe und dann Weiter aus.
6. Überprüfen Sie auf der Seite Komponenten auswählen, ob die aws.greengrass.SystemsManagerAgent Komponente ausgewählt ist, und wählen Sie Weiter aus.
7. Wählen Sie auf der Seite Komponenten konfigurieren die Option aus aws.greengrass.SystemsManagerAgentund gehen Sie dann wie folgt vor:

   a. Wählen Sie Komponente konfigurieren aus.

   b. Geben Sie im Modal Konfigurieren aws.greengrass.SystemsManagerAgent unter Konfigurationsaktualisierung unter Konfiguration zum Zusammenführen von das folgende Konfigurationsupdate ein. Ersetzen Sie *SSMServiceRole* durch den Namen der Servicerolle, die Sie in erstellt habenSchritt 2: Erstellen einer IAM-Servicerolle für Systems Manager.

      ```
      {
      ```

```
  "SSMRegistrationRole": "SSMServiceRole",
  "SSMOverrideExistingRegistration": false
}
```

> **Note**
>
> Wenn das Core-Gerät bereits den Systems Manager Agent ausführt, der bei einer Hybrid-Aktivierung registriert ist, ändern Sie `SSMOverrideExistingRegistration` zu `true`. Dieser Parameter gibt an, ob die Systems Manager Agent-Komponente das Core-Gerät registriert, wenn der Systems Manager Agent bereits mit einer Hybrid-Aktivierung auf dem Gerät ausgeführt wird.
> Sie können auch Tags (`SSMResourceTags`) angeben, die dem von Systems Manager verwalteten Knoten hinzugefügt werden sollen, den die Systems Manager Agent-Komponente für das Core-Gerät erstellt. Weitere Informationen finden Sie unter Systems Manager Agent-Komponentenkonfiguration.

c. Wählen Sie Bestätigen, um das Modal zu schließen, und wählen Sie dann Weiter aus.

8. Behalten Sie auf der Seite Erweiterte Einstellungen konfigurieren die Standardkonfigurationseinstellungen bei und wählen Sie Weiter.
9. Wählen Sie auf der Seite Review (Prüfen) die Option Deploy (Bereitstellen) aus.

   Die Bereitstellung kann bis zu einer Minute dauern.

So stellen Sie die Systems Manager Agent-Komponente bereit (AWS CLI)

Um die Systems Manager Agent-Komponente bereitzustellen, erstellen Sie ein Bereitstellungsdokument, das `aws.greengrass.SystemsManagerAgent` im `-componentsObjekt` enthält, und geben Sie das Konfigurationsupdate für die Komponente an. Folgen Sie den Anweisungen unter Erstellen von Bereitstellungen, um eine neue Bereitstellung zu erstellen oder eine vorhandene Bereitstellung zu überarbeiten.

Das folgende Beispiel für ein teilweises Bereitstellungsdokument gibt an, eine Servicerolle mit dem Namen zu verwenden`SSMServiceRole`. Ersetzen Sie *`SSMServiceRole`* durch den Namen der Servicerolle, die Sie in erstellt habenSchritt 2: Erstellen einer IAM-Servicerolle für Systems Manager.

```
{
  ...,
```

```
  "components": {
    ...,
    "aws.greengrass.SystemsManagerAgent": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"SSMRegistrationRole\":\"SSMServiceRole\",
\"SSMOverrideExistingRegistration\":false}"
      }
    }
  }
}
```

> **Note**
>
> Wenn das Core-Gerät bereits den Systems Manager Agent ausführt, der bei einer Hybrid-Aktivierung registriert ist, ändern Sie `SSMOverrideExistingRegistration` zu `true`. Dieser Parameter gibt an, ob die Systems Manager Agent-Komponente das Core-Gerät registriert, wenn der Systems Manager Agent bereits mit einer Hybrid-Aktivierung auf dem Gerät ausgeführt wird.
>
> Sie können auch Tags (`SSMResourceTags`) angeben, die dem von Systems Manager verwalteten Knoten hinzugefügt werden sollen, den die Systems Manager Agent-Komponente für das Core-Gerät erstellt. Weitere Informationen finden Sie unter Systems Manager Agent-Komponentenkonfiguration.

Es kann einige Minuten dauern, bis die Bereitstellung abgeschlossen ist. Sie können den AWS IoT Greengrass Service verwenden, um den Status der Bereitstellung zu überprüfen, und Sie können die AWS IoT Greengrass Core-Softwareprotokolle und Systems Manager Agent-Komponentenprotokolle überprüfen, um sicherzustellen, dass der Systems Manager Agent erfolgreich ausgeführt wird. Weitere Informationen finden Sie hier:

- Prüfen des Bereitstellungsstatus
- Überwachen von AWS IoT Greengrass Protokollen
- Anzeigen von Systems Manager Agent-Protokollen im AWS Systems Manager -Benutzerhandbuch

Wenn die Bereitstellung fehlschlägt oder der Systems Manager Agent nicht ausgeführt wird, können Sie die Bereitstellung auf jedem Core-Gerät beheben. Weitere Informationen finden Sie hier:

- Problembehebung AWS IoT Greengrass V2
- Fehlerbehebung bei Systems Manager Agent im AWS Systems Manager -Benutzerhandbuch

Schritt 5: Überprüfen der Registrierung des Core-Geräts mit Systems Manager

Wenn die Systems Manager Agent-Komponente ausgeführt wird, registriert sie das Core-Gerät als verwalteten Knoten in Systems Manager. Sie können die AWS IoT Greengrass Konsole, die Systems Manager-Konsole und die Systems Manager-API verwenden, um zu überprüfen, ob ein Core-Gerät als verwalteter Knoten registriert ist. Verwaltete Knoten werden auch als Instances in Teilen der AWS Konsole und der API bezeichnet.

So überprüfen Sie die Registrierung des Core-Geräts (AWS IoT Greengrass-Konsole)

1. Wählen Sie im Navigationsmenü der AWS IoT GreengrassKonsole die Option Core-Geräte aus.
2. Wählen Sie das zu überprüfende Core-Gerät aus.
3. Suchen Sie auf der Detailseite des Core-Geräts die AWS Systems Manager Instance-Eigenschaft. Wenn diese Eigenschaft vorhanden ist und einen Link zur Systems Manager-Konsole anzeigt, wird das Core-Gerät als verwalteter Knoten registriert.

   Sie finden auch die Eigenschaft AWS Systems Manager Ping-Status, um den Status des Systems Manager Agent auf dem Core-Gerät zu überprüfen. Wenn der Status Online lautet, können Sie das Core-Gerät mit Systems Manager verwalten.

So überprüfen Sie die Registrierung des Core-Geräts (Systems Manager-Konsole)

1. Wählen Sie im Navigationsmenü der Systems Manager-Konsole Fleet Manager aus.
2. Gehen Sie unter Verwaltete Knoten wie folgt vor:

   a. Fügen Sie einen Filter hinzu, bei dem der Quelltyp istAWS::IoT::Thing.

   b. Fügen Sie einen Filter hinzu, wobei Source ID der Name des zu überprüfenden Core-Geräts ist.
3. Suchen Sie das Core-Gerät in der Tabelle Verwaltete Knoten. Wenn sich das Core-Gerät in der Tabelle befindet, wird es als verwalteter Knoten registriert.

Sie finden auch die Eigenschaft Systems Manager Agent Ping-Status, um den Status des Systems Manager Agent auf dem Core-Gerät zu überprüfen. Wenn der Status Online lautet, können Sie das Core-Gerät mit Systems Manager verwalten.

So überprüfen Sie die Registrierung des Core-Geräts (AWS CLI)

- Verwenden Sie die -DescribeInstanceInformationOperation, um die Liste der verwalteten Knoten abzurufen, die mit einem von Ihnen angegebenen Filter übereinstimmen. Führen Sie den folgenden Befehl aus, um zu überprüfen, ob ein Core-Gerät als verwalteter Knoten registriert ist. Ersetzen Sie durch *MyGreengrassCore* den Namen des zu überprüfenden Core-Geräts.

  ```
  aws ssm describe-instance-information --filter
   Key=SourceIds,Values=MyGreengrassCore Key=SourceTypes,Values=AWS::IoT::Thing
  ```

  Die Antwort enthält die Liste der verwalteten Knoten, die dem Filter entsprechen. Wenn die Liste einen verwalteten Knoten enthält, wird das Core-Gerät als verwalteter Knoten registriert. Sie finden in der Antwort auch weitere Informationen über den verwalteten Knoten des Core-Geräts. Wenn die Eigenschaft `PingStatus` lautet`Online`, können Sie das Core-Gerät mit Systems Manager verwalten.

Nachdem Sie überprüft haben, ob ein Core-Gerät als verwalteter Knoten in Systems Manager registriert ist, können Sie die Systems Manager-Konsole und die API verwenden, um dieses Core-Gerät zu verwalten. Weitere Informationen zu den Systems Manager-Funktionen, die Sie zur Verwaltung von Greengrass-Core-Geräten verwenden können, finden Sie unter Systems Manager-Funktionen im AWS Systems Manager -Benutzerhandbuch.

Deinstallieren des AWS Systems Manager-Agenten

Wenn Sie ein Greengrass-Core-Gerät nicht mehr damit verwalten möchtenAWS Systems Manager, können Sie das Kerngerät von Systems Manager abmelden und denAWS Systems Manager Agenten (Systems Manager Agent) vom Gerät deinstallieren.

Sie können ein Core-Gerät jederzeit neu registrieren. Stellen Sie dazu erneut die Systems Manager Agent-Komponente bereit, die das Kerngerät bei der Installation bei Systems Manager registriert. Systems Manager speichert den Befehlsverlauf für ein deregistriertes Kerngerät 30 Tage lang.

Themen

Schritt 1: Das Kerngerät von Systems Manager abmelden

Sie können die Systems-Manager-Konsole oder die API verwenden, um die Registrierung des Kerngeräts aufzuheben. Weitere Informationen finden Sie im AWS Systems ManagerBenutzerhandbuch unter Deregistrierung verwalteter Knoten.

Schritt 2: Deinstallieren Sie die Systems Manager Agent-Komponente

Nachdem Sie die Registrierung des Kerngeräts aufgehoben haben, deinstallieren Sie die Systems Manager Agent-Komponente vom Gerät. Um eine Komponente von einem Greengrass-Core-Gerät zu entfernen, überarbeiten Sie das Deployment, in dem die Komponente installiert wurde, und entfernen Sie die Komponente aus dem Deployment. DieAWS IoT Greengrass Core-Software deinstalliert eine Komponente, wenn keine der Bereitstellungen eines Kerngeräts diese Komponente spezifiziert. Weitere Informationen finden Sie unter Bereitstellen von AWS IoT Greengrass Komponenten auf Geräten.

So deinstallieren Sie die Systems Manager Agent-Komponente (Konsole)

1. Wählen Sie im Navigationsmenü der AWS IoT GreengrassKonsole die Option Core-Geräte aus.
2. Wählen Sie das Kerngerät aus, auf dem Sie die Systems Manager Agent-Komponente deinstallieren möchten.
3. Wählen Sie auf der Seite mit den Details zu den Kerngeräten die Registerkarte Bereitstellungen aus.
4. Wählen Sie das Deployment aus, das die Systems Manager Agent-Komponente auf dem Kerngerät bereitstellt.
5. Wählen Sie auf der Seite mit den Deployment-Details die Option Revise aus.
6. Wählen Sie im Bereitstellungsmodal Revision erstellen die Option Bereitstellung überarbeiten aus.
7. Wählen Sie in Schritt 1: Ziel angeben die Option Weiter.
8. In Schritt 2: Wählen Sie Komponenten aus, löschen Sie die Auswahl für die aws.greengrass.SystemsManagerAgentKomponente und wählen Sie dann Weiter.

9. Wählen Sie in Schritt 3: Komponenten konfigurieren die Option Weiter.
10. Wählen Sie in Schritt 4: Erweiterte Einstellungen konfigurieren die Option Weiter.
11. Wählen Sie in Schritt 5: Überprüfung die Option Deploy aus.

So deinstallieren Sie die Systems Manager Agent-Komponente (CLI)

Um die Systems Manager Agent-Komponente zu deinstallieren, überarbeiten Sie die Bereitstellung, in der sie bereitgestellt wird, und entfernen Sie sie aus der Bereitstellung. Weitere Informationen finden Sie unter Überarbeiten von Bereitstellungen.

Die Bereitstellung kann einige Minuten dauern. Sie können denAWS IoT Greengrass Service verwenden, um den Status der Bereitstellung zu überprüfen. Weitere Informationen finden Sie unter Prüfen des Bereitstellungsstatus.

Schritt 3: Systems-Manager-Agent-Software deinstallieren

Die Systems Manager Agent-Software wird weiterhin auf dem Kerngerät ausgeführt, nachdem Sie die Systems Manager Agent-Komponente entfernt haben. Um die Systems Manager Agent-Software zu entfernen, können Sie Befehle auf dem Kerngerät ausführen. Weitere Informationen finden Sie im AWS Systems ManagerBenutzerhandbuch unter Systems Manager Agent von Linux-Instanzen deinstallieren.

Sicherheit in AWS IoT Greengrass

Die Sicherheit in der Cloud hat für 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
- Sicherheit der Infrastruktur in AWS IoT Greengrass
- Konfigurations- und Schwachstellenanalyse in AWS IoT Greengrass
- Code-Integrität inAWS IoT Greengrass V2
- AWS IoT Greengrass und Schnittstellen-VPC-Endpunkte (AWS PrivateLink)
- Bewährte Methoden für die Sicherheit für AWS IoT Greengrass

Datenschutz in AWS IoT Greengrass

Das Modell der geteilten Verantwortung von AWS gilt für den Datenschutz in AWS IoT Greengrass. Wie in diesem Modell beschrieben, ist AWS verantwortlich für den Schutz der globalen Infrastruktur, in der die gesamte AWS Cloud ausgeführt wird. Sie sind dafür verantwortlich, die Kontrolle über Ihre in dieser Infrastruktur gehosteten Inhalte zu behalten. Sie sind auch für die Sicherheitskonfiguration und die Verwaltungsaufgaben für die von Ihnen verwendeten AWS-Services verantwortlich. Weitere Informationen zum Datenschutz finden Sie unter Häufig gestellte Fragen zum Datenschutz. Informationen zum Datenschutz in Europa finden Sie im Blog-Beitrag AWS-Modell der geteilten Verantwortung und in der DSGVO im AWS-Sicherheitsblog.

Aus Datenschutzgründen empfehlen wir, AWS-Konto-Anmeldeinformationen zu schützen und einzelne Benutzer mit AWS IAM Identity Center oder AWS Identity and Access Management (IAM) einzurichten. So erhält jeder Benutzer nur die Berechtigungen, die zum Durchführen seiner Aufgaben erforderlich sind. Außerdem empfehlen wir, die Daten mit folgenden Methoden schützen:

- Verwenden Sie für jedes Konto die Multi-Faktor Authentifizierung (MFA).
- Verwenden Sie SSL/TLS für die Kommunikation mit AWS-Ressourcen. Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Richten Sie die API und die Protokollierung von Benutzeraktivitäten mit AWS CloudTrail ein.
- Verwenden Sie AWS-Verschlüsselungslösungen zusammen mit allen Standardsicherheitskontrollen in AWS-Services.
- Verwenden Sie erweiterte verwaltete Sicherheitsservices wie Amazon Macie, die dabei helfen, in Amazon S3 gespeicherte persönliche Daten zu erkennen und zu schützen.
- Wenn Sie für den Zugriff auf AWS über eine Befehlszeilenschnittstelle oder über eine API FIPS 140-2-validierte kryptografische Module benötigen, verwenden Sie einen FIPS-Endpunkt. Weitere Informationen über verfügbare FIPS-Endpunkte finden Sie unter Federal Information Processing Standard (FIPS) 140-2.

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 AWS IoT Greengrass oder anderen AWS-Services über die Konsole, API, AWS CLI oder AWS-SDKs arbeiten. Alle Daten, die Sie in Tags oder Freitextfelder eingeben, die für Namen verwendet werden, können für Abrechnungs- oder Diagnoseprotokolle verwendet werden. Wenn Sie eine URL für einen externen Server bereitstellen, empfehlen wir dringend, keine Anmeldeinformationen zur Validierung Ihrer Anforderung an den betreffenden Server in die URL einzuschließen.

Weitere Informationen zum Schutz vertraulicher Informationen in finden Sie AWS IoT Greengrassunter the section called “Keine Protokollierung sensibler Informationen”.

Weitere Informationen zum Datenschutz enthält der Blog-Beitrag AWS Shared Responsibility Model and GDPR im AWS-Sicherheitsblog.

Themen

Datenverschlüsselung

AWS IoT Greengrass verwendet Verschlüsselung, um Daten während der Übertragung (über das Internet oder lokale Netzwerk) und im Ruhezustand (gespeichert in der) zu schützenAWS Cloud.

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

AWS IoT Greengrassbesitzt zwei 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 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 an dieAWS Cloudwird über eine TLS-Verbindung mit dem MQTT- oder HTTPS-Protokoll gesendet, sodass sie standardmäßig sicher sind.AWS IoT Greengrassverwendet dasAWS IoT-Transportsicherheitsmodell. Weitere Informationen finden Sie unter Transportsicherheit im AWS IoT Core-Entwicklerhandbuch.

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. Dies beinhaltet die Kommunikation zwischen benutzerdefinierten Komponenten, dieAWS IoTDevice SDK und öffentliche Komponenten wie 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 im .Secrets ManagerKomponente.

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 Greengrassverwendet öffentliche und private Schlüssel für das folgende Szenario:

- 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.

Ein Greengrass-Kerngerät unterstützt den privaten Schlüsselspeicher mithilfe von Dateisystemberechtigungen oder einemHardwaresicherheitsmodulaus. Wenn Sie auf dem Dateisystem basierende private Schlüssel verwenden, sind Sie für die sichere Speicherung auf dem Kerngerät verantwortlich.

Integration von Hardware-Sicherheit

> **Note**
>
> Diese Funktion ist für v2.5.3 und höher der Greengrass-Kernkomponente verfügbar. unterstützt diese Funktion derzeit AWS IoT Greengrass nicht auf Windows-Kerngeräten.

Sie können die AWS IoT Greengrass -Core-Software so konfigurieren, dass sie ein Hardware-Sicherheitsmodul (HSM) über die PKCS#11-Schnittstelle verwendet. Mit dieser Funktion können Sie den privaten Schlüssel und das Zertifikat des Geräts sicher speichern, sodass sie nicht in der Software verfügbar gemacht oder dupliziert werden. Sie können den privaten Schlüssel und das Zertifikat in einem Hardwaremodul wie einem HSM oder einem Trusted Platform Module (TPM) speichern.

Die -AWS IoT GreengrassCore-Software verwendet einen privaten Schlüssel und ein X.509-Zertifikat, um Verbindungen zu den - AWS IoT und -AWS IoT GreengrassServices zu authentifizieren. Die

Secret-Manager-Komponente verwendet diesen privaten Schlüssel, um die Secrets, die Sie auf einem Greengrass-Kerngerät bereitstellen, sicher zu verschlüsseln und zu entschlüsseln. Wenn Sie ein Core-Gerät für die Verwendung eines HSM konfigurieren, verwenden diese Komponenten den privaten Schlüssel und das Zertifikat, die Sie im HSM speichern.

Die Moquette MQTT-Brokerkomponente speichert auch einen privaten Schlüssel für ihr lokales MQTT-Serverzertifikat. Diese Komponente speichert den privaten Schlüssel auf dem Dateisystem des Geräts im Arbeitsordner der Komponente. Derzeit unterstützt AWS IoT Greengrass nicht das Speichern dieses privaten Schlüssels oder Zertifikats in einem HSM.

> **Tip**
>
> Suchen Sie im AWS Partner Device Catalog nach Geräten, die diese Funktion unterstützen.

Themen

Voraussetzungen

Sie müssen die folgenden Anforderungen erfüllen, um ein HSM auf einem Greengrass-Core-Gerät verwenden zu können:

- Greengrass-Kern v2.5.3 oder höher auf dem Core-Gerät installiert. Sie können eine kompatible Version auswählen, wenn Sie die AWS IoT Greengrass Core-Software auf einem Core-Gerät installieren.
- Die auf dem Core-Gerät installierte PKCS#11-Anbieterkomponente. Sie können diese Komponente herunterladen und installieren, wenn Sie die AWS IoT Greengrass Core-Software auf einem Core-Gerät installieren.
- Ein Hardware-Sicherheitsmodul, das das PKCS#1 v1.5-Signaturschema und RSA-Schlüssel mit einer RSA-2048-Schlüsselgröße (oder größer) oder ECC-Schlüsseln unterstützt.

> **Note**
>
> Um ein Hardware-Sicherheitsmodul mit ECC-Schlüsseln zu verwenden, müssen Sie Greengrass-Kern v2.5.6 oder höher verwenden.
> Um ein Hardware-Sicherheitsmodul und einen Secret Manager zu verwenden, müssen Sie ein Hardware-Sicherheitsmodul mit RSA-Schlüsseln verwenden.

- Eine PKCS#11-Anbieterbibliothek, die die AWS IoT Greengrass -Core-Software zur Laufzeit (mit libdl) laden kann, um PKCS#11-Funktionen aufzurufen. Die PKCS#11-Anbieterbibliothek muss die folgenden PKCS#11-API-Operationen implementieren:
  - `C_Initialize`
  - `C_Finalize`
  - `C_GetSlotList`
  - `C_GetSlotInfo`
  - `C_GetTokenInfo`
  - `C_OpenSession`
  - `C_GetSessionInfo`
  - `C_CloseSession`
  - `C_Login`
  - `C_Logout`
  - `C_GetAttributeValue`
  - `C_FindObjectsInit`
  - `C_FindObjects`
  - `C_FindObjectsFinal`
  - `C_DecryptInit`
  - `C_Decrypt`
  - `C_DecryptUpdate`
  - `C_DecryptFinal`
  - `C_SignInit`
  - `C_Sign`
  - `C_SignUpdate`

  - `C_SignFinal`
  - `C_GetMechanismList`
  - `C_GetMechanismInfo`
  - `C_GetInfo`
  - `C_GetFunctionList`
- Das Hardwaremodul muss nach Slot-Label auflösbar sein, wie in der PKCS#11-Spezifikation definiert.
- Sie müssen den privaten Schlüssel und das Zertifikat im HSM im selben Slot speichern und dieselbe Objektbezeichnung und Objekt-ID verwenden, wenn das HSM Objekt-IDs unterstützt.
- Das Zertifikat und der private Schlüssel müssen durch Objektbezeichnungen auflösbar sein.
- Der private Schlüssel muss über die folgenden Berechtigungen verfügen:
  - `sign`
  - `decrypt`
- (Optional) Um die Secret-Manager-Komponente zu verwenden, müssen Sie Version 2.1.0 oder höher verwenden und der private Schlüssel muss über die folgenden Berechtigungen verfügen:
  - `unwrap`
  - `wrap`

Bewährte Methoden für die Hardwaresicherheit

Beachten Sie die folgenden bewährten Methoden, wenn Sie die Hardwaresicherheit auf Greengrass-Core-Geräten konfigurieren.

- Generieren Sie private Schlüssel direkt auf dem HSM mit Hilfe des internen Hardware-Zufallszahlengenerators. Dieser Ansatz ist sicherer als der Import eines privaten Schlüssels, den Sie an anderer Stelle generieren, da der private Schlüssel innerhalb des HSM verbleibt.
- Konfigurieren Sie private Schlüssel so, dass sie unveränderlich sind und den Export verbieten.
- Verwenden Sie das Bereitstellungstool, das der HSM-Hardwareanbieter empfiehlt, eine Zertifikatsignierungsanforderung (CSR) mit dem hardwaregeschützten privaten Schlüssel zu generieren, und verwenden Sie dann die AWS IoTKonsole oder API, um ein Clientzertifikat zu generieren.

> **Note**
>
> Die bewährte Sicherheitsmethode zum Rotieren von Schlüsseln gilt nicht, wenn Sie private Schlüssel auf einem HSM generieren.

Installieren der -AWS IoT GreengrassCore-Software mit Hardwaresicherheit

Wenn Sie die AWS IoT Greengrass Core-Software installieren, können Sie sie so konfigurieren, dass sie einen privaten Schlüssel verwendet, den Sie in einem HSM generieren. Dieser Ansatz folgt der bewährten Sicherheitsmethode, um den privaten Schlüssel im HSM zu generieren, sodass der private Schlüssel innerhalb des HSM verbleibt.

Gehen Sie wie folgt vor, um die -AWS IoT GreengrassCore-Software mit Hardwaresicherheit zu installieren:

1. Generieren Sie einen privaten Schlüssel im HSM.
2. Erstellen Sie eine Zertifikatsignierungsanforderung (CSR) aus dem privaten Schlüssel.
3. Erstellen Sie ein Zertifikat aus der CSR. Sie können ein Zertifikat erstellen, das von AWS IoT oder einer anderen Stammzertifizierungsstelle (CA) signiert wurde. Weitere Informationen zur Verwendung einer anderen Stammzertifizierungsstelle finden Sie unter Erstellen eigener Clientzertifikate im AWS IoT Core -Entwicklerhandbuch.
4. Laden Sie das AWS IoT Zertifikat herunter und importieren Sie es in das HSM.
5. Installieren Sie die AWS IoT Greengrass Core-Software aus einer Konfigurationsdatei, die angibt, dass die PKCS#11-Anbieterkomponente sowie der private Schlüssel und das Zertifikat im HSM verwendet werden sollen.

Sie können eine der folgenden Installationsoptionen wählen, um die AWS IoT Greengrass Core-Software mit Hardwaresicherheit zu installieren:

- Manuelle Installation

  Wählen Sie diese Option, um die erforderlichen AWS Ressourcen manuell zu erstellen und die Hardwaresicherheit zu konfigurieren. Weitere Informationen finden Sie unter Installieren Sie die AWS IoT Greengrass Core-Software mit manueller Ressourcenbereitstellung.
- Installation mit benutzerdefinierter Bereitstellung

Wählen Sie diese Option, um eine benutzerdefinierte Java-Anwendung zu entwickeln, die automatisch die erforderlichen AWS Ressourcen erstellt und die Hardwaresicherheit konfiguriert. Weitere Informationen finden Sie unter Installieren Sie die AWS IoT Greengrass Core-Software mit benutzerdefinierter Ressourcenbereitstellung.

Derzeit unterstützt AWS IoT Greengrass nicht die Installation der -AWS IoT GreengrassCore-Software mit Hardwaresicherheit, wenn Sie mit automatischer Ressourcenbereitstellung oder Flottenbereitstellung installieren. AWS IoT

Konfigurieren der Hardwaresicherheit auf einem vorhandenen Core-Gerät

Sie können den privaten Schlüssel und das Zertifikat eines Core-Geräts in ein HSM importieren, um die Hardwaresicherheit zu konfigurieren.

Überlegungen

- Sie müssen Root-Zugriff auf das Dateisystem des Core-Geräts haben.
- In diesem Verfahren fahren Sie die AWS IoT Greengrass Core-Software herunter, sodass das Core-Gerät offline und nicht verfügbar ist, während Sie die Hardwaresicherheit konfigurieren.

Gehen Sie wie folgt vor, um die Hardwaresicherheit auf einem vorhandenen Core-Gerät zu konfigurieren:

1. Initialisieren Sie das HSM.
2. Stellen Sie die PKCS#11-Anbieterkomponente auf dem Core-Gerät bereit.
3. Halten Sie die AWS IoT Greengrass Core-Software an.
4. Importieren Sie den privaten Schlüssel und das Zertifikat des Core-Geräts in das HSM.
5. Aktualisieren Sie die Konfigurationsdatei der AWS IoT Greengrass Core-Software, um den privaten Schlüssel und das Zertifikat im HSM zu verwenden.
6. Starten Sie die -AWS IoT GreengrassCore-Software.

Schritt 1: Initialisieren des Hardware-Sicherheitsmoduls

Führen Sie den folgenden Schritt aus, um das HSM auf Ihrem Core-Gerät zu initialisieren.

So initialisieren Sie das Hardware-Sicherheitsmodul

- Initialisieren Sie ein PKCS#11-Token im HSM und speichern Sie die Slot-ID und die Benutzer-PIN für das Token. In der Dokumentation zu Ihrem HSM erfahren Sie, wie Sie ein Token initialisieren. Sie verwenden die Slot-ID und die Benutzer-PIN später, wenn Sie die PKCS#11-Anbieterkomponente bereitstellen und konfigurieren.

Schritt 2: Bereitstellen der PKCS#11-Anbieterkomponente

Führen Sie die folgenden Schritte aus, um die PKCS#11-Anbieterkomponente bereitzustellen und zu konfigurieren. Sie können die Komponente auf einem oder mehreren -Core-Geräten bereitstellen.

So stellen Sie die PKCS#11-Anbieterkomponente bereit (Konsole)

1. Wählen Sie im Navigationsmenü der AWS IoT GreengrassKonsole Komponenten aus.
2. Wählen Sie auf der Seite Komponenten die Registerkarte Öffentliche Komponenten und dann ausaws.greengrass.crypto.Pkcs11Provider.
3. Wählen Sie auf der aws.greengrass.crypto.Pkcs11Provider Seite Bereitstellen aus.
4. Wählen Sie unter Zu Bereitstellung hinzufügen eine vorhandene Bereitstellung aus, die Sie überarbeiten möchten, oder wählen Sie aus, um eine neue Bereitstellung zu erstellen, und wählen Sie dann Weiter aus.
5. Wenn Sie eine neue Bereitstellung erstellen möchten, wählen Sie das Ziel-Core-Gerät oder die Objektgruppe für die Bereitstellung aus. Wählen Sie auf der Seite Ziel angeben unter Bereitstellungsziel ein Core-Gerät oder eine Objektgruppe und dann Weiter aus.
6. Wählen Sie auf der Seite Komponenten auswählen unter Öffentliche Komponenten die Option aws.greengrass.crypto.Pkcs11Providerund dann Weiter aus.
7. Wählen Sie auf der Seite Komponenten konfigurieren die Option aus aws.greengrass.crypto.Pkcs11Providerund gehen Sie dann wie folgt vor:

    a. Wählen Sie Komponente konfigurieren aus.

    b. Geben Sie im Modal Konfigurieren aws.greengrass.crypto.Pkcs11Provider unter Konfigurationsaktualisierung unter Konfiguration zum Zusammenführen von das folgende Konfigurationsupdate ein. Aktualisieren Sie die folgenden Konfigurationsparameter mit Werten für die Ziel-Core-Geräte. Geben Sie die Slot-ID und die Benutzer-PIN an, in der Sie das PKCS#11-Token zuvor initialisiert haben. Sie importieren den privaten Schlüssel und das Zertifikat später in diesen Slot im HSM.

`name`

Ein Name für die PKCS#11-Konfiguration.

`library`

Der absolute Dateipfad zur Bibliothek der PKCS#11-Implementierung, die die AWS IoT Greengrass Core-Software mit libdl laden kann.

`slot`

Die ID des Slots, der den privaten Schlüssel und das Gerätezertifikat enthält. Dieser Wert unterscheidet sich vom Slot-Index oder der Slot-Bezeichnung.

`userPin`

Die Benutzer-PIN, die für den Zugriff auf den Slot verwendet werden soll.

```
{
  "name": "softhsm_pkcs11",
  "library": "/usr/lib/softhsm/libsofthsm2.so",
  "slot": 1,
  "userPin": "1234"
}
```

c. Wählen Sie Bestätigen, um das Modal zu schließen, und wählen Sie dann Weiter aus.

8. Behalten Sie auf der Seite Erweiterte Einstellungen konfigurieren die Standardkonfigurationseinstellungen bei und wählen Sie Weiter.
9. Wählen Sie auf der Seite Review (Prüfen) die Option Deploy (Bereitstellen) aus.

   Die Bereitstellung kann bis zu einer Minute dauern.

So stellen Sie die PKCS#11-Anbieterkomponente bereit (AWS CLI)

Um die PKCS#11-Anbieterkomponente bereitzustellen, erstellen Sie ein Bereitstellungsdokument, das `aws.greengrass.crypto.Pkcs11Provider` im `-componentsObjekt` enthält, und geben Sie das Konfigurationsupdate für die Komponente an. Folgen Sie den Anweisungen unter Erstellen von Bereitstellungen, um eine neue Bereitstellung zu erstellen oder eine vorhandene Bereitstellung zu überarbeiten.

Das folgende Beispiel für ein partielles Bereitstellungsdokument gibt an, die PKCS#11-Anbieterkomponente bereitzustellen und zu konfigurieren. Aktualisieren Sie die folgenden Konfigurationsparameter mit Werten für die Ziel-Core-Geräte. Speichern Sie die Slot-ID und die Benutzer-PIN, die Sie später beim Importieren des privaten Schlüssels und Zertifikats in das HSM verwenden können.

`name`

Ein Name für die PKCS#11-Konfiguration.

`library`

Der absolute Dateipfad zur Bibliothek der PKCS#11-Implementierung, die die AWS IoT Greengrass Core-Software mit libdl laden kann.

`slot`

Die ID des Slots, der den privaten Schlüssel und das Gerätezertifikat enthält. Dieser Wert unterscheidet sich vom Slot-Index oder der Slot-Bezeichnung.

`userPin`

Die Benutzer-PIN, die für den Zugriff auf den Slot verwendet werden soll.

```
{
  "name": "softhsm_pkcs11",
  "library": "/usr/lib/softhsm/libsofthsm2.so",
  "slot": 1,
  "userPin": "1234"
}
```

```
{
  ...,
  "components": {
    ...,
    "aws.greengrass.crypto.Pkcs11Provider": {
      "componentVersion": "2.0.0",
      "configurationUpdate": {
        "merge": "{\"name\":\"softhsm_pkcs11\",\"library\":\"/usr/lib/softhsm/
libsofthsm2.so\",\"slot\":1,\"userPin\":\"1234\"}"
      }
    }
  }
```

```
}
```

Es kann einige Minuten dauern, bis die Bereitstellung abgeschlossen ist. Sie können den AWS IoT Greengrass Service verwenden, um den Status der Bereitstellung zu überprüfen. Sie können die AWS IoT Greengrass -Core-Softwareprotokolle überprüfen, um sicherzustellen, dass die PKCS#11-Anbieterkomponente erfolgreich bereitgestellt wurde. Weitere Informationen finden Sie hier:

- Prüfen des Bereitstellungsstatus
- Überwachen von AWS IoT Greengrass Protokollen

Wenn die Bereitstellung fehlschlägt, können Sie die Bereitstellung auf jedem Core-Gerät beheben. Weitere Informationen finden Sie unter Problembehebung AWS IoT Greengrass V2.

Schritt 3: Aktualisieren der Konfiguration auf dem Core-Gerät

Die -AWS IoT GreengrassCore-Software verwendet eine Konfigurationsdatei, die angibt, wie das Gerät funktioniert. Diese Konfigurationsdatei enthält, wo Sie den privaten Schlüssel und das Zertifikat finden, die das Gerät zum Herstellen einer Verbindung mit dem verwendetAWS Cloud. Führen Sie die folgenden Schritte aus, um den privaten Schlüssel und das Zertifikat des Core-Geräts in das HSM zu importieren und die Konfigurationsdatei für die Verwendung des HSM zu aktualisieren.

So aktualisieren Sie die Konfiguration auf dem Core-Gerät, um die Hardwaresicherheit zu verwenden

1. Halten Sie die AWS IoT Greengrass Core-Software an. Wenn Sie die AWS IoT Greengrass Core-Software als Systemservice mit systemd konfiguriert haben, können Sie den folgenden Befehl ausführen, um die Software zu beenden.

   ```
   sudo systemctl stop greengrass.service
   ```

2. Suchen Sie den privaten Schlüssel und die Zertifikatsdateien des Core-Geräts.

   - Wenn Sie die -AWS IoT GreengrassCore-Software mit automatischer Bereitstellung oder Flottenbereitstellung installiert haben, existiert der private Schlüssel unter */greengrass/v2/*`privKey.key`und das Zertifikat existiert unter */greengrass/v2*`/thingCert.crt`.
   - Wenn Sie die -AWS IoT GreengrassCore-Software mit manueller Bereitstellung installiert haben, ist der private Schlüssel */greengrass/v2*`/private.pem.key` standardmäßig bei vorhanden und das Zertifikat ist */greengrass/v2*`/device.pem.crt` standardmäßig bei vorhanden.

Sie können auch die `system.certificateFilePath` Eigenschaften `system.privateKeyPath` und in überprüfen*/greengrass/v2*`/config/effectiveConfig.yaml`, um den Speicherort dieser Dateien zu finden.

3. Importieren Sie den privaten Schlüssel und das Zertifikat in das HSM. In der Dokumentation für Ihr HSM erfahren Sie, wie Sie private Schlüssel und Zertifikate in dieses importieren. Importieren Sie den privaten Schlüssel und das Zertifikat mit der Slot-ID und der Benutzer-PIN, in der Sie das PKCS#11-Token zuvor initialisiert haben. Sie müssen dieselbe Objektbezeichnung und Objekt-ID für den privaten Schlüssel und das Zertifikat verwenden. Speichern Sie die Objektbezeichnung, die Sie beim Importieren jeder Datei angeben. Sie verwenden diese Bezeichnung später, wenn Sie die AWS IoT Greengrass Core-Softwarekonfiguration aktualisieren, um den privaten Schlüssel und das Zertifikat im HSM zu verwenden.
4. Aktualisieren Sie die AWS IoT Greengrass Core-Konfiguration, um den privaten Schlüssel und das Zertifikat im HSM zu verwenden. Um die Konfiguration zu aktualisieren, ändern Sie die AWS IoT Greengrass Core-Konfigurationsdatei und führen die AWS IoT Greengrass Core-Software mit der aktualisierten Konfigurationsdatei aus, um die neue Konfiguration anzuwenden.

   Gehen Sie wie folgt vor:

   a. Erstellen Sie eine Sicherung der AWS IoT Greengrass Core-Konfigurationsdatei. Sie können diese Sicherung verwenden, um das Core-Gerät wiederherzustellen, wenn bei der Konfiguration der Hardwaresicherheit Probleme auftreten.

      ```
      sudo cp /greengrass/v2/config/effectiveConfig.yaml ~/ggc-config-backup.yaml
      ```

   b. Öffnen Sie die AWS IoT Greengrass Core-Konfigurationsdatei in einem Texteditor. Sie können beispielsweise den folgenden Befehl ausführen, um die Datei mit GNU Nano zu bearbeiten. Ersetzen Sie */greengrass/v2* durch den Pfad zum Greengrass-Stammordner.

      ```
      sudo nano /greengrass/v2/config/effectiveConfig.yaml
      ```

   c. Ersetzen Sie den Wert von `system.privateKeyPath` durch den PKCS#11-URI für den privaten Schlüssel im HSM. Ersetzen Sie *iotdevicekey* durch die Objektbezeichnung, in die Sie den privaten Schlüssel und das Zertifikat zuvor importiert haben.

      ```
      pkcs11:object=iotdevicekey;type=private
      ```

d. Ersetzen Sie den Wert von `system.certificateFilePath` durch den PKCS#11-URI für das Zertifikat im HSM. Ersetzen Sie *`iotdevicekey`* durch die Objektbezeichnung, in die Sie den privaten Schlüssel und das Zertifikat zuvor importiert haben.

```
pkcs11:object=iotdevicekey;type=cert
```

Nachdem Sie diese Schritte abgeschlossen haben, sollte die -`system`Eigenschaft in der AWS IoT Greengrass Core-Konfigurationsdatei dem folgenden Beispiel ähneln.

```
system:
  certificateFilePath: "pkcs11:object=iotdevicekey;type=cert"
  privateKeyPath: "pkcs11:object=iotdevicekey;type=private"
  rootCaPath: "/greengrass/v2/rootCA.pem"
  rootpath: "/greengrass/v2"
  thingName: "MyGreengrassCore"
```

5. Wenden Sie die Konfiguration in der aktualisierten `effectiveConfig.yaml` Datei an. Führen Sie `Greengrass.jar` mit dem `--init-config` Parameter aus, um die Konfiguration in anzuwendeneffectiveConfig.yaml. Ersetzen Sie */greengrass/v2* durch den Pfad zum Greengrass-Stammordner.

```
sudo java -Droot="/greengrass/v2" \
  -jar /greengrass/v2/alts/current/distro/lib/Greengrass.jar \
  --start false \
  --init-config /greengrass/v2/config/effectiveConfig.yaml
```

6. Starten Sie die -AWS IoT GreengrassCore-Software. Wenn Sie die AWS IoT Greengrass Core-Software als Systemservice mit systemd konfiguriert haben, können Sie den folgenden Befehl ausführen, um die Software zu starten.

```
sudo systemctl start greengrass.service
```

Weitere Informationen finden Sie unter Ausführen der AWS IoT Greengrass -Core-Software.

7. Überprüfen Sie die AWS IoT Greengrass-Core-Softwareprotokolle, um zu überprüfen, ob die Software gestartet wird und eine Verbindung mit der herstelltAWS Cloud. Die AWS IoT Greengrass -Core-Software verwendet den privaten Schlüssel und das Zertifikat, um eine Verbindung zu den - AWS IoT und -AWS IoT GreengrassServices herzustellen.

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

Die folgenden Protokollmeldungen auf INFO-Ebene zeigen an, dass die -AWS IoT GreengrassCore-Software erfolgreich eine Verbindung zu den - AWS IoT und -AWS IoT GreengrassServices herstellt.

```
2021-12-06T22:47:53.702Z [INFO] (Thread-3)
 com.aws.greengrass.mqttclient.AwsIotMqttClient: Successfully connected to AWS IoT
 Core. {clientId=MyGreengrassCore5, sessionPresent=false}
```

8. (Optional) Nachdem Sie überprüft haben, ob die AWS IoT Greengrass -Core-Software mit dem privaten Schlüssel und dem Zertifikat im HSM funktioniert, löschen Sie den privaten Schlüssel und die Zertifikatsdateien aus dem Dateisystem des Geräts. Führen Sie den folgenden Befehl aus und ersetzen Sie die Dateipfade durch die Pfade zum privaten Schlüssel und zu den Zertifikatsdateien.

```
sudo rm /greengrass/v2/privKey.key
sudo rm /greengrass/v2/thingCert.crt
```

Hardware ohne PKCS#11-Unterstützung verwenden

Die PKCS#11-Bibliothek wird typischerweise vom Hardwarehersteller bereitgestellt oder ist Open Source. So ist es beispielsweise bei standardkonformer Hardware (z. B. TPM1.2) möglich, vorhandene Open-Source-Software zu nutzen. Wenn Ihre Hardware jedoch nicht über eine entsprechende PKCS#11-Bibliotheksimplementierung verfügt oder Sie einen benutzerdefinierten PKCS#11-Anbieter schreiben möchten, wenden Sie sich bei Fragen zur Integration an Ihren Amazon Web Services Enterprise Support-Mitarbeiter.

Weitere Informationen finden Sie auch unter

- PKCS #11 – Nutzungshandbuch für kryptografische Token-Schnittstellen Version 2.4.0
- RFC 7512
- PKCS #1: RSA Encryption Version 1.5

Geräteauthentifizierung und -autorisierung für AWS IoT Greengrass

Geräte in AWS IoT Greengrass-Umgebungen verwenden X.509-Zertifikate für die Authentifizierung und AWS IoT-Richtlinien für die Autorisierung. Zertifikate und Richtlinien ermöglichen es Geräten, sich sicher miteinander 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-Core-Geräte

  Greengrass-Core-Geräte verwenden Zertifikate und AWS IoT Richtlinien, um eine Verbindung zu AWS IoT Core und herzustellenAWS IoT Greengrass. Die Zertifikate und Richtlinien ermöglichen es auchAWS IoT Greengrass, Komponenten und Konfigurationen auf -Core-Geräten bereitzustellen.
- Client-Geräte

  MQTT-Clientgeräte verwenden Zertifikate und Richtlinien, um eine Verbindung zu AWS IoT Core und dem AWS IoT Greengrass Service herzustellen. Auf diese Weise können Client-Geräte die AWS IoT Greengrass Cloud-Erkennung verwenden, um ein Greengrass-Kerngerät zu finden und eine Verbindung zu ihm herzustellen. Ein Client-Gerät verwendet dasselbe Zertifikat, um eine Verbindung mit dem AWS IoT Core Cloud-Service und den Core-Geräten herzustellen. Client-Geräte verwenden auch Erkennungsinformationen für die gegenseitige Authentifizierung mit dem Core-Gerät. Weitere Informationen finden Sie unter Interagieren mit lokalen IoT-Geräten.

X.509-Zertifikate

Die Kommunikation zwischen -Core-Geräten 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 Clientkomponente auf dem Greengrass-Kerngerät, die eine Verbindung zum AWS IoT Core und AWS IoT Greengrass über das Internet herstellt.
- Client-Geräte, die AWS IoT Greengrass über das Internet eine Verbindung zu herstellen, um -Core-Geräte zu erkennen.
- Die MQTT-Brokerkomponente auf dem Greengrass-Kern, die über das lokale Netzwerk eine Verbindung zu Greengrass-Geräten in der Gruppe herstellt.

AWS IoT Greengrass -Core-Geräte speichern Zertifikate im Greengrass-Stammordner.

CA-Zertifikate

Greengrass-Core-Geräte und Client-Geräte laden ein Stammzertifizierungsstellenzertifikat herunter, das für die Authentifizierung mit den - 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.

Client-Geräte laden auch ein Greengrass-Core-Geräte-CA-Zertifikat herunter. Sie verwenden dieses Zertifikat, um das MQTT-Serverzertifikat auf dem Core-Gerät während der gegenseitigen Authentifizierung zu validieren.

Zertifikatrotation auf dem lokalen MQTT-Broker

Wenn Sie die Client-Geräteunterstützung aktivieren, generieren Greengrass-Core-Geräte ein lokales MQTT-Serverzertifikat, das Client-Geräte für die gegenseitige Authentifizierung verwenden. Dieses Zertifikat wird durch das CA-Zertifikat des Core-Geräts signiert, das das Core-Gerät in der AWS IoT Greengrass Cloud speichert. Client-Geräte rufen das CA-Zertifikat des Core-Geräts ab, wenn sie das Core-Gerät erkennen. Sie verwenden das CA-Zertifikat des Core-Geräts, um das MQTT-Serverzertifikat des Core-Geräts zu überprüfen, wenn sie eine Verbindung zum Core-Gerät herstellen. Das CA-Zertifikat des Core-Geräts läuft nach 5 Jahren ab.

Das MQTT-Serverzertifikat läuft standardmäßig alle 7 Tage ab und Sie können diese Dauer auf zwischen 2 und 10 Tage konfigurieren. Dieser begrenzte Zeitraum basiert auf bewährten Sicherheitsmethoden. Diese Rotation trägt dazu bei, die Bedrohung zu minimieren, dass ein Angreifer das MQTT-Serverzertifikat und den privaten Schlüssel stiehlt, um sich als das Greengrass-Kerngerät auszugeben.

Das Greengrass-Core-Gerät rotiert das MQTT-Serverzertifikat 24 Stunden vor Ablauf. Das Greengrass-Core-Gerät generiert ein neues Zertifikat und startet den lokalen MQTT-Broker neu. In diesem Fall werden alle Client-Geräte, die mit dem Greengrass-Core-Gerät verbunden sind, getrennt. Client-Geräte können nach kurzer Zeit wieder eine Verbindung zum Greengrass-Core-Gerät herstellen.

AWS IoT-Richtlinien für Operationen auf Datenebene

Verwenden Sie AWS IoT Richtlinien, um den Zugriff auf die AWS IoT Greengrass Datenebenen AWS IoT Core und zu autorisieren. Die AWS IoT Core Datenebene bietet Operationen für Geräte, Benutzer und Anwendungen. Zu diesen Vorgängen gehört die Möglichkeit, eine Verbindung zu Themen herzustellen AWS IoT Core und diese zu abonnieren. Die AWS IoT Greengrass Datenebene bietet Operationen für Greengrass-Geräte. Weitere Informationen finden Sie unter AWS IoT Greengrass V2-Richtlinienaktionen. Zu diesen Vorgängen gehört die Möglichkeit, Komponentenabhängigkeiten aufzulösen und Artefakte öffentlicher Komponenten herunterzuladen.

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.

> **Important**
>
> Objektrichtlinienvariablen (`iot:Connection.Thing.*`) werden für in AWS IoT Richtlinien für -Core-Geräte oder Greengrass-Operationen auf Datenebene nicht unterstützt. Stattdessen können Sie einen Platzhalter verwenden, der mehreren Geräten entspricht, die ähnliche Namen haben. Sie können beispielsweise angeben, `MyGreengrassDevice*` dass mit `MyGreengrassDevice1`, `MyGreengrassDevice2`usw. übereinstimmt.

> **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 V2-Richtlinienaktionen

AWS IoT Greengrass V2 definiert die folgenden Richtlinienaktionen, die Greengrass-Core-Geräte und Client-Geräte in -AWS IoTRichtlinien verwenden können. Um eine Ressource für eine -Richtlinienaktion anzugeben, verwenden Sie den Amazon-Ressourcennamen (ARN) der Ressource.

Core-Geräteaktionen

`greengrass:GetComponentVersionArtifact`

Gewährt die Berechtigung zum Abrufen einer vorsignierten URL zum Herunterladen eines öffentlichen Komponentenartefakts oder eines Lambda-Komponentenartefakts.

Diese Berechtigung wird ausgewertet, wenn ein Core-Gerät eine Bereitstellung erhält, die eine öffentliche Komponente oder ein Lambda mit Artefakten angibt. Wenn das Core-Gerät bereits über das Artefakt verfügt, wird das Artefakt nicht erneut heruntergeladen.

Ressourcentyp: `componentVersion`

Format des Ressourcen-ARN: `arn:aws:greengrass:`*`region`*`:`*`account-id`*`:components:`*`component-name`*`:versions:`*`component-version`*

`greengrass:ResolveComponentCandidates`

Gewährt die Berechtigung zum Identifizieren einer Liste von Komponenten, die die Anforderungen an Komponente, Version und Plattform für eine Bereitstellung erfüllen. Wenn die Anforderungen in Konflikt geraten oder keine Komponenten vorhanden sind, die die Anforderungen erfüllen, gibt dieser Vorgang einen Fehler zurück und die Bereitstellung auf dem Gerät schlägt fehl.

Diese Berechtigung wird ausgewertet, wenn ein Core-Gerät eine Bereitstellung erhält, die Komponenten angibt.

Ressourcentyp: Keine

Format des Ressourcen-ARN: *

`greengrass:GetDeploymentConfiguration`

Gewährt die Berechtigung zum Abrufen einer vorsignierten URL zum Herunterladen eines großen Bereitstellungsdokuments.

Diese Berechtigung wird ausgewertet, wenn ein Core-Gerät eine Bereitstellung erhält, die ein Bereitstellungsdokument größer als 7 KB (wenn die Bereitstellung auf ein Objekt abzielt) oder 31 KB (wenn die Bereitstellung auf eine Objektgruppe abzielt) angibt. Das Bereitstellungsdokument enthält Komponentenkonfigurationen, Bereitstellungsrichtlinien und Bereitstellungsmetadaten. Weitere Informationen finden Sie unter Bereitstellen von AWS IoT Greengrass Komponenten auf Geräten.

Diese Funktion ist für v2.3.0 und höher der Greengrass-Kernkomponente verfügbar.

Ressourcentyp: Keine

Format des Ressourcen-ARN: *

`greengrass:ListThingGroupsForCoreDevice`

Gewährt die Berechtigung zum Abrufen der Objektgruppenhierarchie eines Core-Geräts.

Diese Berechtigung wird überprüft, wenn ein Core-Gerät eine Bereitstellung von erhältAWS IoT Greengrass. Das Core-Gerät verwendet diese Aktion, um festzustellen, ob es seit der letzten Bereitstellung aus einer Objektgruppe entfernt wurde. Wenn das Core-Gerät aus einer Objektgruppe entfernt wurde und diese Objektgruppe das Ziel einer Bereitstellung auf dem Core-Gerät ist, entfernt das Core-Gerät die von dieser Bereitstellung installierten Komponenten.

Diese Funktion wird von v2.5.0 und höher der Greengrass-Kernkomponente verwendet.

Ressourcentyp: `thing` (Core-Gerät)

Format des Ressourcen-ARN: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`core-device-thing-name`*

`greengrass:VerifyClientDeviceIdentity`

Gewährt die Berechtigung zum Überprüfen der Identität eines Client-Geräts, das eine Verbindung zu einem Core-Gerät herstellt.

Diese Berechtigung wird ausgewertet, wenn ein Core-Gerät die Authentifizierungskomponente des Client-Geräts ausführt und eine MQTT-Verbindung von einem Client-Gerät erhält. Das Client-Gerät präsentiert sein AWS IoT Gerätezertifikat. Anschließend sendet das Core-Gerät das Gerätezertifikat an den AWS IoT Greengrass Cloud-Service, um die Identität des Client-Geräts zu überprüfen. Weitere Informationen finden Sie unter Interagieren mit lokalen IoT-Geräten.

Ressourcentyp: Keine

Format des Ressourcen-ARN: *

`greengrass:VerifyClientDeviceIoTCertificateAssociation`

Gewährt die Berechtigung zum Überprüfen, ob ein Client-Gerät einem -AWS IoTZertifikat zugeordnet ist.

Diese Berechtigung wird ausgewertet, wenn ein Core-Gerät die Authentifizierungskomponente des Client-Geräts ausführt und ein Client-Gerät autorisiert, eine Verbindung über MQTT herzustellen. Weitere Informationen finden Sie unter Interagieren mit lokalen IoT-Geräten.

> **Note**
>
> Damit ein Core-Gerät diesen Vorgang verwenden kann, muss die Greengrass-Servicerolle Ihrem zugeordnet sein AWS-Konto und die `-iot:DescribeCertificate`Berechtigung zulassen.

Ressourcentyp: `thing` (Client-Gerät)

Format des Ressourcen-ARN: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`client-device-thing-name`*

`greengrass:PutCertificateAuthorities`

Gewährt die Berechtigung zum Hochladen von Zertifizierungsstellenzertifikaten (CA), die Client-Geräte herunterladen können, um das Core-Gerät zu überprüfen.

Diese Berechtigung wird ausgewertet, wenn ein Core-Gerät die Authentifizierungskomponente des Client-Geräts installiert und ausführt. Diese Komponente erstellt eine lokale Zertifizierungsstelle und verwendet diese Operation, um ihre CA-Zertifikate hochzuladen. Client-Geräte laden diese CA-Zertifikate herunter, wenn sie den Vorgang Discover verwenden, um Kerngeräte zu finden, mit denen sie eine Verbindung herstellen können. Wenn Client-Geräte eine Verbindung zu einem MQTT-Broker auf einem Core-Gerät herstellen, verwenden sie diese CA-Zertifikate, um die Identität des Core-Geräts zu überprüfen. Weitere Informationen finden Sie unter Interagieren mit lokalen IoT-Geräten.

Ressourcentyp: Keine

ARN-Format: *

`greengrass:GetConnectivityInfo`

Gewährt die Berechtigung zum Abrufen von Verbindungsinformationen für ein Core-Gerät. Diese Informationen beschreiben, wie Client-Geräte eine Verbindung zum Core-Gerät herstellen können.

Diese Berechtigung wird ausgewertet, wenn ein Core-Gerät die Authentifizierungskomponente des Client-Geräts installiert und ausführt. Diese Komponente verwendet die Konnektivitätsinformationen, um gültige CA-Zertifikate zum Hochladen in den AWS IoT Greengrass Cloud-Service mit der -PutCertificateAuthoriesOperation zu generieren. Client-Geräte verwenden diese CA-Zertifikate, um die Identität des Core-Geräts zu überprüfen. Weitere Informationen finden Sie unter Interagieren mit lokalen IoT-Geräten.

Sie können diesen Vorgang auch auf der AWS IoT Greengrass Steuerebene verwenden, um Konnektivitätsinformationen für ein Core-Gerät anzuzeigen. Weitere Informationen finden Sie unter GetConnectivityInfo in der AWS IoT Greengrass V1-API-Referenz.

Ressourcentyp: `thing` (Core-Gerät)

Format des Ressourcen-ARN: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`core-device-thing-name`*

`greengrass:UpdateConnectivityInfo`

Gewährt die Berechtigung zum Aktualisieren von Verbindungsinformationen für ein Core-Gerät. Diese Informationen beschreiben, wie Client-Geräte eine Verbindung zum Core-Gerät herstellen können.

Diese Berechtigung wird ausgewertet, wenn ein Core-Gerät die IP-Detektorkomponente ausführt. Diese Komponente identifiziert die Informationen, die Client-Geräte benötigen, um eine Verbindung zum Core-Gerät im lokalen Netzwerk herzustellen. Anschließend verwendet diese Komponente diese Operation, um die Konnektivitätsinformationen in den AWS IoT Greengrass Cloud-Service hochzuladen, sodass Client-Geräte diese Informationen mit der Discover-Operation abrufen können. Weitere Informationen finden Sie unter Interagieren mit lokalen IoT-Geräten.

Sie können diesen Vorgang auch auf der AWS IoT Greengrass Steuerebene verwenden, um die Konnektivitätsinformationen für ein Core-Gerät manuell zu aktualisieren. Weitere Informationen finden Sie unter UpdateConnectivityInfo in der AWS IoT Greengrass V1-API-Referenz.

Ressourcentyp: `thing` (Core-Gerät)

Format des Ressourcen-ARN: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`core-device-thing-name`*

Client-Geräteaktionen

`greengrass:Discover`

Gewährt die Berechtigung zum Erkennen von Verbindungsinformationen für -Core-Geräte, bei denen ein Client-Gerät eine Verbindung herstellen kann. Diese Informationen beschreiben, wie das Client-Gerät eine Verbindung zu den Core-Geräten herstellen kann. Ein Client-Gerät kann nur die Core-Geräte erkennen, denen Sie es mithilfe der -BatchAssociateClientDeviceWithCoreDeviceOperation zugeordnet haben. Weitere Informationen finden Sie unter Interagieren mit lokalen IoT-Geräten.

Ressourcentyp: `thing` (Client-Gerät)

Format des Ressourcen-ARN: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`client-device-thing-name`*

Aktualisieren der AWS IoT Richtlinie eines Core-Geräts

Sie können die - AWS IoT Greengrass und -AWS IoTKonsolen oder die AWS IoT-API verwenden, um die AWS IoT Richtlinie eines Core-Geräts anzuzeigen und zu aktualisieren.

> **Note**
>
> Wenn Sie das AWS IoT Greengrass -Core-Software-Installationsprogramm zur Bereitstellung von Ressourcen verwendet haben, verfügt Ihr -Core-Gerät über eine -AWS IoTRichtlinie, die den Zugriff auf alle -AWS IoT GreengrassAktionen ermöglicht (`greengrass:*`). Sie können diese Schritte ausführen, um den Zugriff nur auf die Aktionen zu beschränken, die ein Core-Gerät verwendet.

Überprüfen und Aktualisieren der AWS IoT Richtlinie eines Core-Geräts (Konsole)

1. Wählen Sie im Navigationsmenü der AWS IoT Greengrass Konsole die Option Core-Geräte aus.
2. Wählen Sie auf der Seite Core-Geräte das zu aktualisierende Core-Gerät aus.
3. Wählen Sie auf der Detailseite des Core-Geräts den Link zum Objekt des Core-Geräts aus. Dieser Link öffnet die Seite mit den Objektdetails in der -AWS IoTKonsole.
4. Wählen Sie auf der Seite mit den Objektdetails die Option Zertifikate aus.
5. Wählen Sie auf der Registerkarte Zertifikate das aktive Zertifikat des Objekts aus.
6. Wählen Sie auf der Seite mit den Zertifikatsdetails Richtlinien aus.
7. Wählen Sie auf der Registerkarte Richtlinien die zu überprüfende und zu aktualisierende AWS IoT Richtlinie aus. Sie können die erforderlichen Berechtigungen zu jeder Richtlinie hinzufügen, die an das aktive Zertifikat des Core-Geräts angehängt ist.

   > **Note**
   >
   > Wenn Sie das AWS IoT Greengrass-Core-Softwareinstallationsprogramm zur Bereitstellung von Ressourcen verwendet haben, haben Sie zwei AWS IoT Richtlinien. Wir empfehlen Ihnen, die Richtlinie mit dem Namen auszuwählenGreengrassV2IoTThingPolicy, falls sie vorhanden ist. Core-Geräte, die Sie mit dem Schnellinstallationsprogramm erstellen, verwenden diesen Richtliniennamen standardmäßig. Wenn Sie dieser Richtlinie Berechtigungen hinzufügen, erteilen Sie diese Berechtigungen auch anderen -Core-Geräten, die diese Richtlinie verwenden.

8. Wählen Sie in der Richtlinienübersicht die Option Aktive Version bearbeiten aus.
9. Überprüfen Sie die Richtlinie und fügen Sie Berechtigungen nach Bedarf hinzu, entfernen oder bearbeiten Sie sie.
10. 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.
11. Wählen Sie Als neue Version speichern aus.

Überprüfen und Aktualisieren der AWS IoT Richtlinie eines Core-Geräts (AWS CLI)

1. Listen Sie die Prinzipale für das AWS IoT Objekt des Core-Geräts auf. Objektprinzipale können X.509-Gerätezertifikate oder andere Identifikatoren sein. Führen Sie den folgenden Befehl aus und ersetzen Sie *MyGreengrassCore* durch den Namen des Core-Geräts.

   ```
   aws iot list-thing-principals --thing-name MyGreengrassCore
   ```

   Der Vorgang gibt eine Antwort zurück, die die Objektprinzipale des Core-Geräts auflistet.

   ```
   {
       "principals": [
           "arn:aws:iot:us-west-2:123456789012:cert/certificateId"
       ]
   }
   ```

2. Identifizieren Sie das aktive Zertifikat des Core-Geräts. Führen Sie den folgenden Befehl aus und ersetzen Sie *certificateId* durch die ID jedes Zertifikats aus dem vorherigen Schritt, bis Sie das aktive Zertifikat finden. Die Zertifikat-ID ist die hexadezimale Zeichenfolge am Ende des Zertifikat-ARN. Das `--query` Argument gibt an, dass nur der Status des Zertifikats ausgegeben wird.

   ```
   aws iot describe-certificate --certificate-id certificateId --query
    'certificateDescription.status'
   ```

   Der Vorgang gibt den Zertifikatsstatus als Zeichenfolge zurück. Wenn das Zertifikat beispielsweise aktiv ist, gibt dieser Vorgang aus`"ACTIVE"`.

3. Listen Sie die AWS IoT Richtlinien auf, die dem Zertifikat angefügt sind. Führen Sie den folgenden Befehl aus und ersetzen Sie den Zertifikat-ARN durch den ARN des Zertifikats.

```
aws iot list-principal-policies --principal arn:aws:iot:us-
west-2:123456789012:cert/certificateId
```

Der Vorgang gibt eine Antwort zurück, in der die AWS IoT Richtlinien aufgeführt sind, die dem Zertifikat angefügt sind.

```
{
    "policies": [
        {
            "policyName":
 "GreengrassTESCertificatePolicyMyGreengrassCoreTokenExchangeRoleAlias",
            "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassTESCertificatePolicyMyGreengrassCoreTokenExchangeRoleAlias"
        },
        {
            "policyName": "GreengrassV2IoTThingPolicy",
            "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy"
        }
    ]
}
```

4. Wählen Sie die anzuzeigende und zu aktualisierende Richtlinie aus.

> **Note**
>
> Wenn Sie das AWS IoT Greengrass-Core-Softwareinstallationsprogramm zur Bereitstellung von Ressourcen verwendet haben, haben Sie zwei AWS IoT Richtlinien. Wir empfehlen Ihnen, die Richtlinie mit dem Namen auszuwählenGreengrassV2IoTThingPolicy, falls vorhanden. Core-Geräte, die Sie mit dem Schnellinstallationsprogramm erstellen, verwenden diesen Richtliniennamen standardmäßig. Wenn Sie dieser Richtlinie Berechtigungen hinzufügen, erteilen Sie diese Berechtigungen auch anderen -Core-Geräten, die diese Richtlinie verwenden.

5. Rufen Sie das Dokument der Richtlinie ab. Führen Sie den folgenden Befehl aus und ersetzen Sie *GreengrassV2IoTThingPolicy* durch den Namen der Richtlinie.

```
aws iot get-policy --policy-name GreengrassV2IoTThingPolicy
```

Die Operation gibt eine Antwort zurück, die das Dokument der Richtlinie und andere Informationen über die Richtlinie enthält. Das Richtliniendokument ist ein JSON-Objekt, das als Zeichenfolge serialisiert wird.

```
{
    "policyName": "GreengrassV2IoTThingPolicy",
    "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
    "policyDocument": "{\
  \\"Version\\": \\"2012-10-17\\",\
  \\"Statement\\": [\
    {\
      \\"Effect\\": \\"Allow\\",\
      \\"Action\\": [\
                \\"iot:Connect\\",\
                \\"iot:Publish\\",\
                \\"iot:Subscribe\\",\
                \\"iot:Receive\\",\
                \\"greengrass:*\\"\
],\
      \\"Resource\\": \\"*\\"\
    }\
  ]\
}",
    "defaultVersionId": "1",
    "creationDate": "2021-02-05T16:03:14.098000-08:00",
    "lastModifiedDate": "2021-02-05T16:03:14.098000-08:00",
    "generationId":
 "f19144b798534f52c619d44f771a354f1b957dfa2b850625d9f1d0fde530e75f"
}
```

6. Verwenden Sie einen Online-Konverter oder ein anderes Tool, um die Richtliniendokumentzeichenfolge in ein JSON-Objekt zu konvertieren, und speichern Sie sie dann in einer Datei mit dem Namen `iot-policy.json`.

   Wenn Sie beispielsweise das Tool jq installiert haben, können Sie den folgenden Befehl ausführen, um das Richtliniendokument abzurufen, es in ein JSON-Objekt zu konvertieren und das Richtliniendokument als JSON-Objekt zu speichern.

   ```
   aws iot get-policy --policy-name GreengrassV2IoTThingPolicy --query
    'policyDocument' | jq fromjson >> iot-policy.json
   ```

7. Überprüfen Sie das Richtliniendokument und fügen Sie Berechtigungen nach Bedarf hinzu, entfernen oder bearbeiten Sie sie.

   Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um GNU Nano zum Öffnen der Datei zu verwenden.

   ```
   nano iot-policy.json
   ```

   Wenn Sie fertig sind, könnte das Richtliniendokument der minimalen AWS IoT Richtlinie für -Core-Geräte ähneln.

8. Speichern Sie die Änderungen als neue Version der Richtlinie. Führen Sie den folgenden Befehl aus und ersetzen Sie *GreengrassV2IoTThingPolicy* durch den Namen der Richtlinie.

   ```
   aws iot create-policy-version --policy-name GreengrassV2IoTThingPolicy --policy-
   document file://iot-policy.json --set-as-default
   ```

   Die Operation gibt eine Antwort ähnlich dem folgenden Beispiel zurück, wenn sie erfolgreich ist.

   ```
   {
       "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
   GreengrassV2IoTThingPolicy",
       "policyDocument": "{\
     \\"Version\\": \\"2012-10-17\\",\
     \\"Statement\\": [\
       {\
         \\"Effect\\": \\"Allow\\",\
         \\"Action\\": [\
   \\t\\t\\"iot:Connect\\",\
   \\t\\t\\"iot:Publish\\",\
   \\t\\t\\"iot:Subscribe\\",\
   \\t\\t\\"iot:Receive\\",\
   \\t\\t\\"greengrass:*\\"\
         ],\
         \\"Resource\\": \\"*\\"\
       }\
     ]\
   }",
       "policyVersionId": "2",
       "isDefaultVersion": true
   }
   ```

Minimale AWS IoT Richtlinie für -AWS IoT Greengrass V2Core-Geräte

Important

Spätere Versionen der Greengrass-Kernkomponente erfordern zusätzliche Berechtigungen für die minimale AWS IoT Richtlinie. Möglicherweise müssen Sie die AWS IoT Richtlinien Ihrer Core-Geräte aktualisieren, um zusätzliche Berechtigungen zu erteilen.

- Core-Geräte, auf denen Greengrass-Kern v2.5.0 und höher ausgeführt wird, verwenden die `-greengrass:ListThingGroupsForCoreDevice`Berechtigung, um Komponenten zu deinstallieren, wenn Sie ein Core-Gerät aus einer Objektgruppe entfernen.
- Core-Geräte, auf denen Greengrass-Kern v2.3.0 und höher ausgeführt wird, verwenden die `-greengrass:GetDeploymentConfiguration`Berechtigung, um große Bereitstellungskonfigurationsdokumente zu unterstützen.

Die folgende Beispielrichtlinie enthält den Mindestsatz von Aktionen, die erforderlich sind, um eine grundlegende Greengrass-Funktionalität für Ihr Core-Gerät zu unterstützen.

- Die `Connect` Richtlinie enthält den `*` Platzhalter nach dem Objektnamen des Core-Geräts (z. B. *`core-device-thing-name`*`*`). Das Core-Gerät verwendet dasselbe Gerätezertifikat, um mehrere gleichzeitige Abonnements für zu erstellenAWS IoT Core, aber die Client-ID in einer Verbindung stimmt möglicherweise nicht genau mit dem Objektnamen des Core-Geräts überein. Nach den ersten 50 Abonnements verwendet das Core-Gerät *`core-device-thing-name`*`#`*`number`* als Client-ID, wobei für jede weitere 50 Abonnements *`number`* erhöht wird. Wenn beispielsweise ein Core-Gerät mit dem Namen 150 gleichzeitige Abonnements `MyCoreDevice` erstellt, verwendet es die folgenden Client-IDs:
  - Abonnements 1 bis 50: `MyCoreDevice`
  - Abonnements 51 bis 100: `MyCoreDevice#2`
  - Abonnements 101 bis 150: `MyCoreDevice#3`

  Der Platzhalter ermöglicht es dem Core-Gerät, eine Verbindung herzustellen, wenn es diese Client-IDs mit einem Suffix verwendet.
- 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, Greengrass-Komponenten und Client-Geräten 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 gewährt die Berechtigung zum Veröffentlichen im folgenden Thema für Telemetriedaten.

  ```
  $aws/things/core-device-thing-name/greengrass/health/json
  ```

  Sie können diese Berechtigung für -Core-Geräte entfernen, bei denen Sie Telemetrie deaktivieren. Weitere Informationen finden Sie unter Erfassen von Telemetriedaten zum Systemstatus von -AWS IoT GreengrassCore-Geräten.

- Die Richtlinie gewährt die Berechtigung, eine IAM-Rolle über einen -AWS IoTRollenalias anzunehmen. Das Core-Gerät verwendet diese Rolle, die als Token-Exchange-Rolle bezeichnet wird, um AWS Anmeldeinformationen zu erhalten, die es zur Authentifizierung von AWS Anforderungen verwenden kann. Weitere Informationen finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

  Wenn Sie die -AWS IoT GreengrassCore-Software installieren, erstellen Sie eine zweite AWS IoT Richtlinie, die nur diese Berechtigung enthält, und fügen sie an. Wenn Sie diese Berechtigung in die primäre AWS IoT Richtlinie Ihres Core-Geräts aufnehmen, können Sie die andere AWS IoT Richtlinie trennen und löschen.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "arn:aws:iot:region:account-id:client/core-device-thing-name*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive",
                "iot:Publish"
            ],
            "Resource": [
```

```
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-
name/greengrass/health/json",
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-
name/greengrassv2/health/json",
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-
name/jobs/*",
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-
name/shadow/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topicfilter/$aws/things/core-device-
thing-name/jobs/*",
                "arn:aws:iot:region:account-id:topicfilter/$aws/things/core-device-
thing-name/shadow/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iot:AssumeRoleWithCertificate",
            "Resource": "arn:aws:iot:region:account-id:rolealias/token-exchange-role-
alias-name"
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:GetComponentVersionArtifact",
                "greengrass:ResolveComponentCandidates",
                "greengrass:GetDeploymentConfiguration",
                "greengrass:ListThingGroupsForCoreDevice"
            ],
            "Resource": "*"
        }
    ]
}
```

Minimale AWS IoT Richtlinie zur Unterstützung von Client-Geräten

Die folgende Beispielrichtlinie enthält den Mindestsatz von Aktionen, die erforderlich sind, um die Interaktion mit Clientgeräten auf einem Core-Gerät zu unterstützen. Um Client-Geräte zu unterstützen, muss ein Core-Gerät zusätzlich zur AWS IoTMinimalrichtlinie für den grundlegenden Betrieb über die Berechtigungen in dieser AWS IoT Richtlinie verfügen.

- Die Richtlinie ermöglicht es dem Core-Gerät, seine eigenen Konnektivitätsinformationen zu aktualisieren. Diese Berechtigung (`greengrass:UpdateConnectivityInfo`) ist nur erforderlich, wenn Sie die IP-Detektorkomponente auf dem Core-Gerät bereitstellen.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-name-gci/shadow/get"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topicfilter/$aws/things/core-device-thing-name-gci/shadow/update/delta",
                "arn:aws:iot:region:account-id:topicfilter/$aws/things/core-device-thing-name-gci/shadow/get/accepted"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
```

```
            "Resource": [
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-
name-gci/shadow/update/delta",
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-
name-gci/shadow/get/accepted"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:PutCertificateAuthorities",
                "greengrass:VerifyClientDeviceIdentity"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:VerifyClientDeviceIoTCertificateAssociation"
            ],
            "Resource": "arn:aws:iot:region:account-id:thing/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:GetConnectivityInfo",
                "greengrass:UpdateConnectivityInfo"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:thing/core-device-thing-name"
            ]
        }
    ]
}
```

Minimale AWS IoT Richtlinie für Client-Geräte

Die folgende Beispielrichtlinie enthält den Mindestsatz von Aktionen, die ein Client-Gerät benötigt, um Core-Geräte zu erkennen, auf denen sie sich über MQTT verbinden und kommunizieren. Die AWS IoT Richtlinie des Client-Geräts muss die -`greengrass:Discover`Aktion enthalten, damit das Gerät Konnektivitätsinformationen für die zugehörigen Greengrass-Core-Geräte erkennen kann. Geben Sie

im `Resource` Abschnitt den Amazon-Ressourcennamen (ARN) des Client-Geräts an, nicht den ARN des Greengrass-Core-Geräts.

- Die Richtlinie ermöglicht die Kommunikation zu allen MQTT-Themen. Um bewährte Sicherheitsmethoden zu befolgen, beschränken Sie die `iot:Receive` Berechtigungen `iot:Publishiot:Subscribe`, und auf den minimalen Satz von Themen, die ein Client-Gerät für Ihren Anwendungsfall benötigt.
- Die Richtlinie ermöglicht es dem Objekt, Kerngeräte für alle AWS IoT Dinge zu erkennen. Um bewährte Sicherheitsmethoden zu befolgen, beschränken Sie die `greengrass:Discover` Berechtigung auf das AWS IoT Objekt des Client-Geräts oder einen Platzhalter, der einer Reihe von AWS IoT Objekten entspricht.

  > **Important**
  >
  > Objektrichtlinienvariablen (`iot:Connection.Thing.*`) werden für in AWS IoT Richtlinien für -Core-Geräte oder Greengrass-Operationen auf Datenebene nicht unterstützt. Stattdessen können Sie einen Platzhalter verwenden, der mehreren Geräten entspricht, die ähnliche Namen haben. Sie können beispielsweise angeben, `MyGreengrassDevice*` dass mit `MyGreengrassDevice1`, `MyGreengrassDevice2`usw. übereinstimmt.

- Die AWS IoT Richtlinie eines Client-Geräts erfordert in der Regel keine Berechtigungen für `iot:GetThingShadow`-`iot:UpdateThingShadow`, - oder -`iot:DeleteThingShadow`Aktionen, da das Greengrass-Core-Gerät Schattensynchronisierungsvorgänge für Client-Geräte verarbeitet. Damit das Core-Gerät Client-Geräteschatten verarbeiten kann, überprüfen Sie, ob die AWS IoT Richtlinie des Core-Geräts diese Aktionen zulässt und ob der `Resource` Abschnitt die ARNs der Client-Geräte enthält.

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

```
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topic/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topicfilter/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topic/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:Discover"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:thing/*"
            ]
        }
    ]
}
```

Identity and Access Management für AWS IoT Greengrass

AWS Identity and Access Management (IAM) ist ein AWS-Service, mit dem Administratoren den Zugriff auf AWS-Ressourcen sicher steuern können. IAM-Administratoren steuern, wer authentifiziert

(angemeldet) und autorisiert (Berechtigungen besitzt) ist, um AWS IoT Greengrass Ressourcen zu nutzen. IAM ist ein AWS-Service, den Sie ohne zusätzliche Kosten verwenden können.

> **Note**
>
> In diesem Thema werden IAM-Konzepte und -Funktionen beschrieben. Informationen zu den von unterstützten IAM-Funktionen AWS IoT Greengrassfinden Sie unter the section called “Funktionsweise von AWS IoT Greengrass mit IAM”.

Zielgruppe

Wie Sie AWS Identity and Access Management (IAM) verwenden, unterscheidet sich je nach Ihrer Arbeit in AWS IoT Greengrass.

Service-Benutzer: Wenn Sie den AWS IoT Greengrass-Service zur Ausführung von Aufgaben verwenden, stellt Ihnen Ihr Administrator die Anmeldeinformationen und Berechtigungen bereit, die Sie benötigen. Wenn Sie für Ihre Arbeit weitere AWS IoT Greengrass-Funktionen ausführen, benötigen Sie möglicherweise zusätzliche Berechtigungen. Wenn Sie die Funktionsweise 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 Funktion in AWS IoT Greengrass haben.

Service-Administrator: Wenn Sie in Ihrem Unternehmen für AWS IoT Greengrass-Ressourcen verantwortlich sind, haben Sie wahrscheinlich vollständigen Zugriff auf AWS IoT Greengrass. Es ist Ihre Aufgabe, zu bestimmen, auf welche AWS IoT Greengrass-Funktionen und Ressourcen Ihre Service-Benutzer zugreifen sollen. Sie müssen dann Anträge an Ihren IAM-Administrator stellen, um die Berechtigungen Ihrer Servicenutzer zu ändern. Lesen Sie die Informationen auf dieser Seite, um die Grundkonzepte von IAM nachzuvollziehen. Weitere Informationen dazu, wie Ihr Unternehmen IAM mit AWS IoT Greengrass verwenden kann, finden Sie unter Funktionsweise von AWS IoT Greengrass mit IAM.

IAM-Administrator: Wenn Sie als IAM-Administrator fungieren, sollten Sie Einzelheiten dazu kennen, wie Sie Richtlinien zur Verwaltung des Zugriffs auf AWS IoT Greengrass verfassen können. Beispiele für identitätsbasierte AWS IoT Greengrass-Richtlinien, die Sie in IAM verwenden können, finden Sie unter Beispiele für identitätsbasierte Richtlinien für AWS IoT Greengrass.

Authentifizierung mit Identitäten

Authentifizierung ist die Art, wie Sie sich mit Ihren Anmeldeinformationen bei AWS anmelden. Die Authentifizierung (Anmeldung bei AWS) muss als Root-Benutzer des AWS-Kontos, als IAM-Benutzer oder durch Übernahme einer IAM-Rolle erfolgen.

Sie können sich bei AWS als Verbundidentität mit Anmeldeinformationen anmelden, die über eine Identitätsquelle bereitgestellt werden. Benutzer von AWS IAM Identity Center (IAM Identity Center), die Single-Sign-on-Authentifizierung Ihres Unternehmens und Anmeldeinformationen für Google oder Facebook sind Beispiele für Verbundidentitäten. Wenn Sie sich als Verbundidentität anmelden, hat der Administrator vorher mithilfe von IAM-Rollen einen Identitätsverbund eingerichtet. Wenn Sie auf AWS mithilfe des Verbunds zugreifen, übernehmen Sie indirekt eine Rolle.

Je nachdem, welcher Benutzertyp Sie sind, können Sie sich bei der AWS Management Console oder beim AWS-Zugriffsportal anmelden. Weitere Informationen zum Anmelden bei AWS finden Sie unter So melden Sie sich bei Ihrem AWS-Konto an im Benutzerhandbuch von AWS-Anmeldung.

Bei programmgesteuertem Zugriff auf AWS bietet AWS ein Software Development Kit (SDK) und eine Command Line Interface (CLI, Befehlszeilenschnittstelle) zum kryptographischen Signieren Ihrer Anfragen mit Ihren Anmeldeinformationen. Wenn Sie keine AWS-Tools verwenden, müssen Sie Anforderungen selbst signieren. Weitere Informationen zur Verwendung der empfohlenen Methode zum eigenen Signieren von Anforderungen finden Sie unter Signieren von AWS-API-Anforderungen im IAM-Benutzerhandbuch.

Unabhängig von der verwendeten Authentifizierungsmethode müssen Sie möglicherweise zusätzliche Sicherheitsinformationen angeben. AWS empfiehlt beispielsweise die Verwendung von Multi-Faktor Authentifizierung (MFA), um die Sicherheit Ihres Kontos zu verbessern. Weitere Informationen finden Sie unter Multi-Faktor-Authentifizierung im AWS IAM Identity Center-Benutzerhandbuch und Verwenden der Multi-Faktor-Authentifizierung (MFA) in AWS im IAM-Benutzerhandbuch.

AWS-Konto-Root-Benutzer

Wenn Sie ein AWS-Konto neu erstellen, beginnen Sie mit einer Anmeldeidentität, die vollständigen Zugriff auf alle AWS-Services und Ressourcen des Kontos hat. Diese Identität wird als AWS-Konto-Root-Benutzer bezeichnet. Für den Zugriff auf den Root-Benutzer müssen Sie sich mit der E-Mail-Adresse und dem Passwort anmelden, die zur Erstellung des Kontos verwendet wurden. 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 unter Aufgaben, die Root-Benutzer-Anmeldeinformationen erfordern im IAM-Benutzerhandbuch.

IAM-Benutzer und -Gruppen

Ein IAM-Benutzer ist eine Identität in Ihrem AWS-Konto mit bestimmten Berechtigungen für eine einzelne Person oder eine einzelne Anwendung. Wenn möglich, empfehlen wir, temporäre Anmeldeinformationen zu verwenden, anstatt IAM-Benutzer zu erstellen, die langfristige Anmeldeinformationen wie Passwörter und Zugriffsschlüssel haben. Bei speziellen Anwendungsfällen, die langfristige Anmeldeinformationen mit IAM-Benutzern erfordern, empfehlen wir jedoch, die Zugriffsschlüssel zu rotieren. Weitere Informationen finden Sie unter Regelmäßiges Rotieren von Zugriffsschlüsseln für Anwendungsfälle, die langfristige Anmeldeinformationen erfordern im IAM-Benutzerhandbuch.

Eine IAM-Gruppe ist eine Identität, die eine Sammlung von IAM-Benutzern angibt. 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 einer Gruppe mit dem Namen IAMAdmins Berechtigungen zum Verwalten 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 Erstellen eines IAM-Benutzers (anstatt einer Rolle) im IAM-Benutzerhandbuch.

IAM-Rollen

Eine IAM-Rolle ist eine Identität in Ihrem AWS-Konto mit spezifischen Berechtigungen. Sie ist einem IAM-Benutzer vergleichbar, ist aber nicht mit einer bestimmten Person verknüpft. Sie können vorübergehend eine IAM-Rolle in der AWS Management Console übernehmen, indem Sie Rollen wechseln. Sie können eine Rolle annehmen, indem Sie eine AWS CLI oder AWS-API-Operation aufrufen oder eine benutzerdefinierte URL verwenden. Weitere Informationen zu Methoden für die Verwendung von Rollen finden Sie unter Verwenden von IAM-Rollen im IAM-Benutzerhandbuch.

IAM-Rollen mit temporären Anmeldeinformationen sind in folgenden Situationen hilfreich:

- 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 unter Erstellen von Rollen für externe Identitätsanbieter im IAM-Benutzerhandbuch. Wenn Sie IAM Identity Center verwenden, konfigurieren Sie einen Berechtigungssatz. Wenn Sie steuern möchten, 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 andere Berechtigungen für eine bestimmte Aufgabe zu erhalten.
- Kontoübergreifender Zugriff – Sie können eine IAM-Rolle verwenden, um einem vertrauenswürdigen Prinzipal in einem anderen Konto den Zugriff auf Ressourcen in Ihrem Konto zu ermöglichen. Rollen stellen die primäre Möglichkeit dar, um kontoübergreifendem Zugriff zu gewähren. In einigen AWS-Services können Sie jedoch eine Richtlinie direkt an eine Ressource anfügen (anstatt eine Rolle als Proxy zu verwenden). Informationen zu den Unterschieden zwischen Rollen und ressourcenbasierten Richtlinien für den kontoübergreifenden Zugriff finden Sie unter So unterscheiden sich IAM-Rollen von ressourcenbasierten Richtlinien im IAM-Benutzerhandbuch.
- Serviceübergreifender Zugriff: Einige AWS-Services verwenden Features in anderen AWS-Services. Wenn Sie beispielsweise einen Aufruf in einem Service tätigen, führt dieser Service häufig Anwendungen in Amazon EC2 aus oder speichert Objekte in Amazon S3. Ein Dienst kann dies mit den Berechtigungen des aufrufenden Prinzipals mit einer Servicerolle oder mit einer serviceverknüpften Rolle tun.
  - Forward access sessions (FAS) – Wenn Sie einen IAM-Benutzer oder eine IAM-Rolle zum Ausführen von Aktionen in AWS verwenden, gelten Sie als Prinzipal. Bei einigen Services könnte es Aktionen geben, die dann eine andere Aktion in einem anderen Service auslösen. FAS verwendet die Berechtigungen des Prinzipals, der einen AWS-Service aufruft, in Kombination mit der Anforderung an den AWS-Service, Anforderungen an nachgelagerte Services zu stellen. FAS-Anfragen werden nur dann gestellt, wenn ein Service eine Anfrage erhält, die eine Interaktion mit anderen AWS-Services oder -Ressourcen erfordert. In diesem Fall müssen Sie über Berechtigungen zum Ausführen beider Aktionen verfügen. Einzelheiten zu den Richtlinien für FAS-Anfragen finden Sie unter Zugriffssitzungen weiterleiten.
  - Servicerolle: Eine Servicerolle ist eine IAM-Rolle, die ein Service übernimmt, um Aktionen in Ihrem Namen auszuführen. Ein IAM-Administrator kann eine Servicerolle innerhalb von IAM erstellen, ändern und löschen. Weitere Informationen finden Sie unter Erstellen einer Rolle zum Delegieren von Berechtigungen an einen AWS-Service im IAM-Benutzerhandbuch.

  - Serviceverknüpfte Rolle: Eine serviceverknüpfte Rolle ist ein Typ von Servicerolle, die mit einem AWS-Service verknüpft ist. Der Service kann die Rolle übernehmen, um eine Aktion in Ihrem Namen auszuführen. Serviceverknüpfte Rollen werden in Ihrem AWS-Konto angezeigt und gehören zum Service. Ein IAM-Administrator kann die Berechtigungen für serviceverbundene Rollen anzeigen, aber nicht bearbeiten.
- Anwendungen in Amazon EC2: Sie können eine IAM-Rolle verwenden, um temporäre Anmeldeinformationen für Anwendungen zu verwalten, die auf einer EC2-Instance ausgeführt werden und AWS CLI- oder AWS-API-Anforderungen durchführen. Das ist eher zu empfehlen, als Zugriffsschlüssel innerhalb der EC2-Instance zu speichern. Erstellen Sie ein Instance-Profil, das an die Instance angefügt ist, um eine AWS-Rolle einer EC2-Instance zuzuweisen und die Rolle für sämtliche Anwendungen der Instance bereitzustellen. Ein Instance-Profil enthält die Rolle und ermöglicht, dass Programme, die in der EC2-Instance ausgeführt werden, temporäre Anmeldeinformationen erhalten. Weitere Informationen finden Sie unter Verwenden einer IAM-Rolle zum Erteilen von Berechtigungen für Anwendungen, die auf Amazon EC2-Instances ausgeführt werden im IAM-Benutzerhandbuch.

Informationen dazu, wann Sie IAM-Rollen oder IAM-Benutzer verwenden sollten, finden Sie unter Erstellen einer IAM-Rolle (anstatt eines Benutzers) im IAM-Benutzerhandbuch.

Verwalten des Zugriffs mit Richtlinien

Für die Zugriffssteuerung in AWS erstellen Sie Richtlinien und weisen diese den AWS-Identitäten oder -Ressourcen zu. Eine Richtlinie ist ein Objekt in AWS, das, wenn es einer Identität oder Ressource zugeordnet wird, deren Berechtigungen definiert. AWS wertet diese Richtlinien aus, wenn ein Prinzipal (Benutzer, Root-Benutzer oder Rollensitzung) eine Anforderung stellt. Berechtigungen in den Richtlinien bestimmen, ob die Anforderung zugelassen oder abgelehnt wird. Die meisten Richtlinien werden in AWS als JSON-Dokumente gespeichert. Weitere Informationen zu Struktur und Inhalten von JSON-Richtliniendokumenten finden Sie unter Übersicht über JSON-Richtlinien im IAM-Benutzerhandbuch.

Administratoren können mithilfe von AWS-JSON-Richtlinien festlegen, wer zum Zugriff auf was berechtigt ist. Das bedeutet, welcher Prinzipal kann Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen.

Standardmäßig haben Benutzer, Gruppen und Rollen keine Berechtigungen. Ein IAM-Administrator muss IAM-Richtlinien erstellen, die Benutzern die Berechtigung erteilen, Aktionen für die Ressourcen

auszuführen, die sie benötigen. Der Administrator kann dann die IAM-Richtlinien zu Rollen hinzufügen, und Benutzer können die Rollen annehmen.

IAM-Richtlinien definieren Berechtigungen für eine Aktion unabhängig von der Methode, die Sie zur Ausführung der Aktion verwenden. Angenommen, es gibt eine Richtlinie, die Berechtigungen für die `iam:GetRole`-Aktion erteilt. Ein Benutzer mit dieser Richtlinie kann Benutzerinformationen über die AWS Management Console, die AWS CLI oder die AWS -API abrufen.

Identitätsbasierte Richtlinien

Identitätsbasierte Richtlinien sind JSON-Berechtigungsrichtliniendokumente, die Sie einer Identität anfügen können, wie z. B. IAM-Benutzern, -Benutzergruppen oder -Rollen. Diese Richtlinien steuern, welche Aktionen die Benutzer und Rollen für welche Ressourcen und unter welchen Bedingungen ausführen können. Informationen zum Erstellen identitätsbasierter Richtlinien finden Sie unter Erstellen von IAM-Richtlinien im IAM-Benutzerhandbuch.

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 AWS-Konto anfügen können. Verwaltete Richtlinien umfassen von AWS verwaltete und von Kunden verwaltete Richtlinien. Informationen dazu, wie Sie zwischen einer verwalteten Richtlinie und einer eingebundenen Richtlinie wählen, finden Sie unter Auswahl zwischen verwalteten und eingebundenen Richtlinien im IAM-Benutzerhandbuch.

Ressourcenbasierte Richtlinien

Ressourcenbasierte Richtlinien sind JSON-Richtliniendokumente, die Sie an eine Ressource anfügen. Beispiele für ressourcenbasierte Richtlinien sind IAM-Rollen-Vertrauensrichtlinien und Amazon-S3-Bucket-Richtlinien. In Services, die ressourcenbasierte Richtlinien unterstützen, können Service-Administratoren sie verwenden, um den Zugriff auf eine bestimmte Ressource zu steuern. 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. Prinzipale können Konten, Benutzer, Rollen, Verbundbenutzer oder AWS-Services umfassen.

Ressourcenbasierte Richtlinien sind Richtlinien innerhalb dieses Diensts. Sie können verwaltete AWS-Richtlinien von IAM nicht in einer ressourcenbasierten Richtlinie verwenden.

Zugriffssteuerungslisten (ACLs)

Zugriffssteuerungslisten (ACLs) steuern, welche Prinzipale (Kontomitglieder, Benutzer oder Rollen) auf eine Ressource zugreifen können. ACLs sind ähnlich wie ressourcenbasierte Richtlinien, verwenden jedoch nicht das JSON-Richtliniendokumentformat.

Amazon S3, AWS WAF und Amazon VPC sind Beispiele für Dienste, die ACLs unterstützen. Weitere Informationen“ zu ACLs finden Sie unter Zugriffskontrollliste (ACL) – Übersicht (Access Control List) im Amazon-Simple-Storage-Service-Entwicklerhandbuch.

Weitere Richtlinientypen

AWS unterstützt zusätzliche, weniger häufig verwendete Richtlinientypen. Diese Richtlinientypen können die maximalen Berechtigungen festlegen, die Ihnen von den häufiger verwendeten Richtlinientypen erteilt werden können.

- Berechtigungsgrenzen:Eine Berechtigungsgrenze ist ein erweitertes Feature, mit der Sie die maximalen Berechtigungen festlegen können, die eine identitätsbasierte Richtlinie einer IAM-Entität (IAM-Benutzer oder -Rolle) erteilen 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 über Berechtigungsgrenzen finden Sie unter Berechtigungsgrenzen für IAM-Entitäten im IAM-Benutzerhandbuch.
- Service-Kontrollrichtlinien (SCPs) – SCPs sind JSON-Richtlinien, die die maximalen Berechtigungen für eine Organisation oder Organisationseinheit (OE) in AWS Organizations angeben. AWS Organizations ist ein Dienst für die Gruppierung und zentrale Verwaltung mehrerer AWS-Konten Ihres Unternehmens. Wenn Sie innerhalb einer Organisation alle Features aktivieren, können Sie Service-Kontrollrichtlinien (SCPs) auf alle oder einzelne Ihrer Konten anwenden. SCPs schränken Berechtigungen für Entitäten in Mitgliedskonten einschließlich des jeweiligen Root-Benutzer des AWS-Kontos ein. Weitere Informationen zu Organizations und SCPs finden Sie unter Funktionsweise von SCPs 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 unter Sitzungsrichtlinien im IAM-Benutzerhandbuch.

Mehrere Richtlinientypen

Wenn mehrere auf eine Anforderung mehrere Richtlinientypen angewendet werden können, sind die entsprechenden Berechtigungen komplizierter. Informationen dazu, wie AWS die Zulässigkeit einer Anforderung ermittelt, wenn mehrere Richtlinientypen beteiligt sind, finden Sie unter Logik für die Richtlinienauswertung im IAM-Benutzerhandbuch.

Weitere Informationen finden Sie auch unter

- the section called “Funktionsweise von AWS IoT Greengrass mit IAM”
- the section called “Beispiele für identitätsbasierte Richtlinien”
- the section called “Problembehandlung bei Identitäts- und Zugriffsproblemen”

Funktionsweise von AWS IoT Greengrass mit IAM

Bevor Sie IAM zum Verwalten des Zugriffs auf verwendenAWS IoT Greengrass, sollten Sie die IAM-Funktionen verstehen, die Sie verwenden könnenAWS IoT Greengrass.

IAM-Funktion	Von Greengrass unterstützt?
Identitätsbasierte Richtlinien mit Berechtigungen auf Ressourcenebene	Ja
Ressourcenbasierte Richtlinien	Nein
Zugriffskontrolllisten (ACLs)	Nein
Auf Tags basierende Autorisierung	Ja
Temporäre Anmeldeinformationen	Ja
Service-verknüpfte Rollen	Nein
Servicerollen	Ja

Eine Übersicht darüber, wie andereAWS Dienste mit IAM funktionieren, finden Sie unter AWS-Services, die mit IAM funktionieren im IAM-Benutzerhandbuch.

Identitätsbasierte Richtlinien für AWS IoT Greengrass

Mit identitätsbasierten IAM-Richtlinien können Sie festlegen, welche Aktionen und Ressourcen gewährt oder abgelehnt werden. Darüber hinaus können Sie die Bedingungen festlegen, unter denen Aktionen zugelassen oder abgelehnt werden. AWS IoT Greengrass unterstützt bestimmte Aktionen, Ressourcen und Bedingungsschlüssel. Informationen zu sämtlichen Elementen, die Sie in einer Richtlinie verwenden, finden Sie in der IAM-Referenz für JSON-Richtlinienelemente im IAM-Benutzerhandbuch.

Aktionen

Administratoren können mit AWS-JSON-Richtlinien festlegen, welche Personen zum Zugriff auf welche Ressourcen berechtigt sind. Das heißt, welcher Prinzipal kann Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen.

Das Element `Action` einer JSON-Richtlinie beschreibt die Aktionen, mit denen Sie den Zugriff in einer Richtlinie zulassen oder verweigern können. Richtlinienaktionen haben normalerweise denselben Namen wie die zugehörige AWS-API-Operation. Es gibt einige Ausnahmen, z. B. Aktionen, die nur mit Genehmigung durchgeführt werden können und für die es keinen passenden API-Vorgang gibt. 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.

Richtlinienaktionen für AWS IoT Greengrass verwenden ein `greengrass:`-Präfix vor der Aktion. Um jemandem beispielsweise die`ListCoreDevices` -API-Operation zum Auflisten der eigenen Kerngeräte in seiner -Richtlinie zu ermöglichenAWS-Konto, fügen Sie die`greengrass:ListCoreDevices` Aktion in seine Richtlinie ein. 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 einzelnen einzelnen Anweisung anzugeben, listen Sie sie in Klammern (`[]`) und trennen Sie sie wie folgt durch Kommata:

```
"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”.

Eine vollständige Liste derAWS IoT Greengrass Aktionen finden SieAWS IoT Greengrass im IAM-Benutzerhandbuch unter Definierte Aktionen von.

Ressourcen

Administratoren können mit AWS-JSON-Richtlinien festlegen, welche Personen zum Zugriff auf welche Ressourcen berechtigt sind. Das bedeutet die Festlegung, welcher Prinzipal Aktionen für welche Ressourcen unter welchen Bedingungen ausführen kann.

Das JSON-Richtlinienelement `Resource` gibt die Objekte an, auf welche die Aktion angewendet wird. Anweisungen müssen entweder ein `Resource`- oder ein `NotResource`-Element enthalten. Als bewährte Methode geben Sie eine Ressource mit dem zugehörigen Amazon-Ressourcennamen (ARN) an. 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-Ressourcen-ARNs, die im `Resource`-Element einer Richtlinienanweisung verwendet werden können. Eine Zuordnung der unterstützten

Berechtigungen auf Ressourcenebene fürAWS IoT Greengrass Aktionen finden SieAWS IoT Greengrass im IAM-Benutzerhandbuch unter Definierte Aktionen von.

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 wie folgt in Klammern (`[]`) auf und trennen Sie sie durch Kommas:

```
"Resource": [
  "resource-arn1",
  "resource-arn2",
  "resource-arn3"
]
```

Weitere Informationen über ARN-Formate finden Sie unter Amazon-Ressourcennamen (ARNs) und AWS-Service-Namespaces im Allgemeine Amazon Web Services-Referenz.

Bedingungsschlüssel

Administratoren können mithilfe von AWS-JSON-Richtlinien festlegen, wer zum Zugriff auf was berechtigt ist. 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, wertet AWS die Bedingung mittels 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. Beispielsweise können Sie einem IAM-Benutzer die Berechtigung für den Zugriff auf eine Ressource nur dann

gewähren, wenn sie mit dessen IAM-Benutzernamen gekennzeichnet ist. Weitere Informationen finden Sie unter IAM-Richtlinienelemente: Variablen und Tags im IAM-Benutzerhandbuch.

AWS unterstützt globale Bedingungsschlüssel und servicespezifische Bedingungsschlüssel. Eine Liste aller globalen AWS-Bedingungsschlüssel finden Sie unter Globale AWS-Bedingungskontextschlüssel im IAM-Benutzerhandbuch.

Beispiele

Beispiele für identitätsbasierte AWS IoT Greengrass-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 keine ACLs.

Autorisierung auf der Basis von AWS IoT Greengrass-Tags

Sie können Tags an unterstützte AWS IoT Greengrass-Ressourcen anfügen oder Tags in einer Anforderung an AWS IoT Greengrass übergeben. 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 -Ressourcen mit Tags (Markierungen).

IAM-Rollen fürAWS IoT Greengrass

Eine IAM-Rolle ist eine Entität in Ihrem AWS-Konto mit spezifischen Berechtigungen.

Verwenden temporärer Anmeldeinformationen mit AWS IoT Greengrass

Temporäre Anmeldeinformationen werden verwendet, um sich über einen Verbund anzumelden, eine IAM-Rolle anzunehmen oder eine kontenübergreifende Rolle anzunehmen. Temporäre Sicherheitsanmeldeinformationen erhalten Sie durch Aufrufen vonAWS STS -API-Vorgängen wie AssumeRoleoder GetFederationToken.

Auf dem Greengrass-Kern werden den Greengrass-Komponenten temporäre Anmeldeinformationen für die Geräterolle zur Verfügung gestellt. Wenn Ihre Komponenten dasAWS SDK verwenden, müssen Sie keine Logik hinzufügen, um die Anmeldeinformationen abzurufen, da dasAWS SDK dies für Sie erledigt.

Serviceverknüpfte Rollen

AWS IoT Greengrass unterstützt keine serviceverknüpften Rollen.

Servicerollen

Diese Funktion ermöglicht einem Service das Annehmen einer Servicerolle in Ihrem Namen. Diese Rolle gewährt dem Service Zugriff auf Ressourcen in anderen Services, um eine Aktion in Ihrem Namen auszuführen. Servicerollen werden in Ihrem IAM-Konto angezeigt und gehören zum Konto. Dies bedeutet, dass ein IAM-Administrator die Berechtigungen für diese Rolle ändern kann. Dies kann jedoch die Funktionalität des Services beeinträchtigen.

AWS IoT GreengrassKerngeräte verwenden eine Servicerolle, die es Greengrass-Komponenten und Lambda-Funktionen ermöglicht, in Ihrem Namen auf einige IhrerAWS Ressourcen zuzugreifen. Weitere Informationen finden Sie unter the section called “Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS”.

AWS IoT Greengrass verwendet eine Servicerolle, um auf einige Ihrer AWS-Ressourcen in Ihrem Namen zuzugreifen. Weitere Informationen finden Sie unter Greengrass-Servicerolle.

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.

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": "*"
        }
    ]
}
```

Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS

AWS IoT Greengrass Kerngeräte verwenden den Anbieter AWS IoT Core für Anmeldeinformationen, um Anrufe an Dienste zu autorisieren. AWS Der Anbieter AWS IoT Core für Anmeldeinformationen ermöglicht es Geräten, ihre X.509-Zertifikate als eindeutige Geräteidentität zur Authentifizierung AWS von Anfragen zu verwenden. Dadurch entfällt die Notwendigkeit, eine AWS Zugriffsschlüssel-ID und einen geheimen Zugriffsschlüssel auf Ihren AWS IoT Greengrass Kerngeräten zu speichern. Weitere Informationen finden Sie im AWS IoT Core Entwicklerhandbuch unter Autorisieren von direkten Aufrufen von AWS Diensten.

Wenn Sie die AWS IoT Greengrass Core-Software ausführen, können Sie wählen, ob Sie die AWS Ressourcen bereitstellen möchten, die das Kerngerät benötigt. Dazu gehört auch die AWS Identity and Access Management (IAM-) Rolle, die Ihr Kerngerät über den Anbieter für AWS IoT Core Anmeldeinformationen übernimmt. Verwenden Sie das `--provision true` Argument, um eine Rolle und Richtlinien zu konfigurieren, die es dem Kerngerät ermöglichen, temporäre AWS Anmeldeinformationen abzurufen. Dieses Argument konfiguriert auch einen AWS IoT Rollenalias, der auf diese IAM-Rolle verweist. Sie können den Namen der zu verwendenden IAM-Rolle und den AWS IoT Rollenalias angeben. Wenn Sie diese anderen Namensparameter `--provision true` ohne diese anderen Namensparameter angeben, erstellt und verwendet das Greengrass-Core-Gerät die folgenden Standardressourcen:

- IAM-Rolle: `GreengrassV2TokenExchangeRole`

Diese Rolle hat eine benannte Richtlinie `GreengrassV2TokenExchangeRoleAccess` und eine Vertrauensbeziehung, die es ermöglicht, diese Rolle `credentials.iot.amazonaws.com` zu übernehmen. Die Richtlinie umfasst die Mindestberechtigungen für das Kerngerät.

> **Important**
>
> Diese Richtlinie beinhaltet nicht den Zugriff auf Dateien in S3-Buckets. Sie müssen der Rolle Berechtigungen hinzufügen, damit Kerngeräte Komponentenartefakte aus S3-Buckets abrufen können. Weitere Informationen finden Sie unter Erlauben Sie den Zugriff auf S3-Buckets für Komponentenartefakte.

- AWS IoT Rollen-Alias: `GreengrassV2TokenExchangeRoleAlias`

  Dieser Rollenalias bezieht sich auf die IAM-Rolle.

Weitere Informationen finden Sie unter Schritt 3: Installieren derAWS IoT Greengrass Core-Software: Installieren der Kern-Software: Installieren.

Sie können den Rollenalias auch für ein vorhandenes Kerngerät festlegen. Konfigurieren Sie dazu den `iotRoleAlias` Konfigurationsparameter der Greengrass Nucleus-Komponente.

Sie können temporäre AWS Anmeldeinformationen für diese IAM-Rolle erwerben, um AWS Operationen in Ihren benutzerdefinierten Komponenten auszuführen. Weitere Informationen finden Sie unter Interagieren mit -AWSServices.

Themen

- Berechtigungen für Servicerollen für Kerngeräte
- Erlauben Sie den Zugriff auf S3-Buckets für Komponentenartefakte

Berechtigungen für Servicerollen für Kerngeräte

Die Rolle ermöglicht es dem folgenden Dienst, die Rolle zu übernehmen:

- `credentials.iot.amazonaws.com`

Wenn Sie die AWS IoT Greengrass Core-Software verwenden, um diese Rolle zu erstellen, verwendet sie die folgende Berechtigungsrichtlinie, damit Core-Geräte eine Verbindung herstellen

und Protokolle an diese senden können AWS. Der Name der Richtlinie ist standardmäßig der Name der IAM-Rolle, der mit endet. `Access` Wenn Sie beispielsweise den Standard-IAM-Rollennamen verwenden, lautet der Name dieser Richtlinie. `GreengrassV2TokenExchangeRoleAccess`

Greengrass nucleus v2.5.0 and later

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

v2.4.x

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:DescribeCertificate",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

Earlier than v2.4.0

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:DescribeCertificate",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "iot:Connect",
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

Erlauben Sie den Zugriff auf S3-Buckets für Komponentenartefakte

Die standardmäßige Core-Geräterolle erlaubt es Core-Geräten nicht, auf S3-Buckets zuzugreifen. Um Komponenten bereitzustellen, die Artefakte in S3-Buckets enthalten, müssen Sie die `s3:GetObject` Berechtigung hinzufügen, Kerngeräten das Herunterladen von Komponentenartefakten zu gestatten. Sie können der Core-Geräterolle eine neue Richtlinie hinzufügen, um diese Berechtigung zu erteilen.

Um eine Richtlinie hinzuzufügen, die den Zugriff auf Komponentenartefakte in Amazon S3 ermöglicht

1. Erstellen Sie eine Datei mit dem Namen `component-artifact-policy.json` und kopieren Sie den folgenden JSON-Code in die Datei. Diese Richtlinie ermöglicht den Zugriff auf alle Dateien in einem S3-Bucket. Ersetzen Sie DOC-EXAMPLE-BUCKET durch den Namen des S3-Buckets, damit das Kerngerät darauf zugreifen kann.

   ```
   {
     "Version": "2012-10-17",
   ```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
    }
  ]
}
```

2. Führen Sie den folgenden Befehl aus, um die Richtlinie aus dem Richtliniendokument in zu erstellen. `component-artifact-policy.json`

   Linux or Unix

   ```
   aws iam create-policy \
     --policy-name MyGreengrassV2ComponentArtifactPolicy \
     --policy-document file://component-artifact-policy.json
   ```

   Windows Command Prompt (CMD)

   ```
   aws iam create-policy ^
     --policy-name MyGreengrassV2ComponentArtifactPolicy ^
     --policy-document file://component-artifact-policy.json
   ```

   PowerShell

   ```
   aws iam create-policy `
     --policy-name MyGreengrassV2ComponentArtifactPolicy `
     --policy-document file://component-artifact-policy.json
   ```

   Kopieren Sie den Amazon Resource Name (ARN) der Richtlinie aus den Richtlinienmetadaten in der Ausgabe. Sie verwenden diesen ARN, um diese Richtlinie im nächsten Schritt an die zentrale Geräterolle anzuhängen.

3. Führen Sie den folgenden Befehl aus, um die Richtlinie an die zentrale Geräterolle anzuhängen. Ersetzen Sie *`GreengrassV2 TokenExchangeRole`* durch den Namen der Rolle, die Sie bei der Ausführung der AWS IoT Greengrass Core-Software angegeben haben. Ersetzen Sie dann den Richtlinien-ARN durch den ARN aus dem vorherigen Schritt.

Linux or Unix

```
aws iam attach-role-policy \
  --role-name GreengrassV2TokenExchangeRole \
  --policy-arn
 arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
```

Windows Command Prompt (CMD)

```
aws iam attach-role-policy ^
  --role-name GreengrassV2TokenExchangeRole ^
  --policy-arn
 arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
```

PowerShell

```
aws iam attach-role-policy `
  --role-name GreengrassV2TokenExchangeRole `
  --policy-arn
 arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
```

Wenn der Befehl keine Ausgabe hat, war er erfolgreich, und Ihr Kerngerät kann auf Artefakte zugreifen, die Sie in diesen S3-Bucket hochladen.

Minimale IAM-Richtlinie für das Installationsprogramm zur Bereitstellung von Ressourcen

Wenn Sie die AWS IoT Greengrass Core-Software installieren, können Sie die erforderlichen AWS Ressourcen wie ein AWS IoT Ding und eine IAM-Rolle für Ihr Gerät bereitstellen. Sie können auch lokale Entwicklungstools auf dem Gerät bereitstellen. Das Installationsprogramm benötigt AWS Anmeldeinformationen, damit es diese Aktionen in Ihrem ausführen kann AWS-Konto. Weitere Informationen finden Sie unter Installieren Sie die AWS IoT Greengrass Core-Software..

Die folgende Beispielrichtlinie umfasst die Mindestanzahl an Aktionen, die das Installationsprogramm zur Bereitstellung dieser Ressourcen benötigt. Diese Berechtigungen sind erforderlich, wenn Sie das `--provision` Argument für das Installationsprogramm angeben. Ersetzen Sie *`Account-ID`*

durch Ihre AWS-Konto ID und *GreenGrassV2 TokenExchangeRole* durch den Namen der Token-Austauschrolle, die Sie mit dem Installer-Argument angeben. `--tes-role-name`

> **Note**
>
> Die `DeployDevTools` Richtlinienerklärung ist nur erforderlich, wenn Sie das `--deploy-dev-tools` Argument für das Installationsprogramm angeben.

Greengrass nucleus v2.5.0 and later

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CreateTokenExchangeRole",
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy",
                "iam:CreatePolicy",
                "iam:CreateRole",
                "iam:GetPolicy",
                "iam:GetRole",
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::account-id:role/GreengrassV2TokenExchangeRole",
                "arn:aws:iam::account-
id:policy/GreengrassV2TokenExchangeRoleAccess",
                "arn:aws:iam::aws:policy/GreengrassV2TokenExchangeRoleAccess"
            ]
        },
        {
            "Sid": "CreateIoTResources",
            "Effect": "Allow",
            "Action": [
                "iot:AddThingToThingGroup",
                "iot:AttachPolicy",
                "iot:AttachThingPrincipal",
                "iot:CreateKeysAndCertificate",
                "iot:CreatePolicy",
                "iot:CreateRoleAlias",
```

```
                "iot:CreateThing",
                "iot:CreateThingGroup",
                "iot:DescribeEndpoint",
                "iot:DescribeRoleAlias",
                "iot:DescribeThingGroup",
                "iot:GetPolicy"
            ],
            "Resource": "*"
        },
        {
            "Sid": "DeployDevTools",
            "Effect": "Allow",
            "Action": [
                "greengrass:CreateDeployment",
                "iot:CancelJob",
                "iot:CreateJob",
                "iot:DeleteThingShadow",
                "iot:DescribeJob",
                "iot:DescribeThing",
                "iot:DescribeThingGroup",
                "iot:GetThingShadow",
                "iot:UpdateJob",
                "iot:UpdateThingShadow"
            ],
            "Resource": "*"
        }
    ]
}
```

Earlier than v2.5.0

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CreateTokenExchangeRole",
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy",
                "iam:CreatePolicy",
                "iam:CreateRole",
                "iam:GetPolicy",
                "iam:GetRole",
```

```
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::account-id:role/GreengrassV2TokenExchangeRole",
                "arn:aws:iam::account-
id:policy/GreengrassV2TokenExchangeRoleAccess",
                "arn:aws:iam::aws:policy/GreengrassV2TokenExchangeRoleAccess"
            ]
        },
        {
            "Sid": "CreateIoTResources",
            "Effect": "Allow",
            "Action": [
                "iot:AddThingToThingGroup",
                "iot:AttachPolicy",
                "iot:AttachThingPrincipal",
                "iot:CreateKeysAndCertificate",
                "iot:CreatePolicy",
                "iot:CreateRoleAlias",
                "iot:CreateThing",
                "iot:CreateThingGroup",
                "iot:DescribeEndpoint",
                "iot:DescribeRoleAlias",
                "iot:DescribeThingGroup",
                "iot:GetPolicy"
            ],
            "Resource": "*"
        },
        {
            "Sid": "DeployDevTools",
            "Effect": "Allow",
            "Action": [
                "greengrass:CreateDeployment",
                "iot:CancelJob",
                "iot:CreateJob",
                "iot:DeleteThingShadow",
                "iot:DescribeJob",
                "iot:DescribeThing",
                "iot:DescribeThingGroup",
                "iot:GetThingShadow",
                "iot:UpdateJob",
                "iot:UpdateThingShadow"
            ],
            "Resource": "*"
```

```
        }
    ]
}
```

Greengrass-Servicerolle

Die Greengrass-Servicerolle ist eine AWS Identity and Access Management (IAM)-Servicerolle, die autorisiertAWS IoT Greengrass, in Ihrem Namen auf Ressourcen von -AWSServices zuzugreifen. Diese Rolle ermöglicht es dem , die Identität von Clientgeräten AWS IoT Greengrass zu überprüfen und die Konnektivitätsinformationen des Core-Geräts zu verwalten.

> **Note**
>
> AWS IoT Greengrass V1 verwendet diese Rolle auch, um wichtige Aufgaben auszuführen. Weitere Informationen finden Sie unter Greengrass-Servicerolle im AWS IoT Greengrass V1 - Entwicklerhandbuch.

Damit AWS IoT Greengrass auf Ihre -Ressourcen zugreifen kann, muss die Greengrass-Servicerolle Ihrem zugeordnet sein AWS-Konto und AWS IoT Greengrass als vertrauenswürdige Entität angeben. Die Rolle muss die -AWSGreengrassResourceAccessRolePolicyverwaltete Richtlinie oder eine benutzerdefinierte Richtlinie enthalten, die gleichwertige Berechtigungen für die von Ihnen verwendeten AWS IoT Greengrass Funktionen definiert. AWS verwaltet diese Richtlinie, die den Satz von Berechtigungen definiert, die für den Zugriff auf Ihre -AWSRessourcen AWS IoT Greengrass verwendet. Weitere Informationen finden Sie unter AWS verwaltete Richtlinie: AWSGreengrassResourceAccessRolePolicy.

Sie können dieselbe Greengrass-Servicerolle in wiederverwendenAWS-Regionen, müssen sie jedoch Ihrem Konto in jeder zuordnenAWS-Region, in der Sie verwendenAWS IoT Greengrass. Wenn die Servicerolle nicht in der aktuellen konfiguriert istAWS-Region, können -Core-Geräte Client-Geräte nicht überprüfen und die Konnektivitätsinformationen nicht aktualisieren.

In den folgenden Abschnitten wird beschrieben, wie Sie die Greengrass-Servicerolle mit der AWS Management Console oder der erstellen und verwaltenAWS CLI.

Themen

- Verwalten der Greengrass-Servicerolle (Konsole)
- Verwalten der Greengrass-Servicerolle (CLI)

- Weitere Informationen finden Sie auch unter

> **Note**
>
> Zusätzlich zu der Servicerolle, die den Zugriff auf Serviceebene autorisiert, weisen Sie Greengrass-Core-Geräten eine Token-Exchange-Rolle zu. Die Tokenaustauschrolle ist eine separate IAM-Rolle, die steuert, wie Greengrass-Komponenten und Lambda-Funktionen auf dem Core-Gerät auf -AWSServices zugreifen können. Weitere Informationen finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

Verwalten der Greengrass-Servicerolle (Konsole)

Die AWS IoT-Konsole vereinfacht die Verwaltung Ihrer Greengrass-Servicerolle. Wenn Sie beispielsweise die Client-Geräteerkennung für ein Core-Gerät konfigurieren, prüft die Konsole, ob Ihr an eine Greengrass-Servicerolle in der aktuellen angefügt AWS-Konto istAWS-Region. Andernfalls kann die Konsole eine Servicerolle für Sie erstellen und konfigurieren. Weitere Informationen finden Sie unter the section called “Erstellen der Greengrass-Servicerolle”.

Sie können die Konsole für die folgenden Aufgaben zur Rollenverwaltung verwenden:

Themen

- Suchen Ihrer Greengrass-Servicerolle (Konsole)
- Erstellen der Greengrass-Servicerolle (Konsole)
- Ändern der Greengrass-Servicerolle (Konsole)
- Trennen der Greengrass-Servicerolle (Konsole)

> **Note**
>
> Der Benutzer, der bei der Konsole angemeldet ist, muss über Berechtigungen zum Anzeigen, Erstellen oder Ändern der Servicerolle verfügen.

Suchen Ihrer Greengrass-Servicerolle (Konsole)

Gehen Sie wie folgt vor, um die Servicerolle zu finden, die in der aktuellen AWS IoT Greengrass verwendetAWS-Region.

1. Navigieren Sie zur AWS IoT-Konsole.
2. Wählen Sie im Navigationsbereich Settings (Einstellungen).
3. Scrollen Sie zum Abschnitt Greengrass service role (Greengrass-Servicerolle), um Ihre Servicerolle und deren Richtlinien anzuzeigen.

   Wenn Ihnen keine Servicerolle angezeigt wird, kann die Konsole eine für Sie erstellen oder konfigurieren. Weitere Informationen finden Sie unter Erstellen der Greengrass-Servicerolle.

Erstellen der Greengrass-Servicerolle (Konsole)

Die Konsole kann eine standardmäßige Greengrass-Servicerolle für Sie erstellen und konfigurieren. Diese Rolle hat die folgenden Eigenschaften.

Eigenschaft	Wert
Name	`Greengrass_ServiceRole`
Trusted entity (Vertrauenswürdige Entität)	`AWS service: greengrass`
Richtlinie	AWSGreengrassResourceAccessRolePolicy

> **Note**
>
> Wenn Sie diese Rolle mit dem AWS IoT Greengrass V1 Geräteeinrichtungsskript erstellen, lautet der Rollenname `GreengrassServiceRole_`*`random-string`*.

Wenn Sie die Client-Geräteerkennung für ein Core-Gerät konfigurieren, prüft die Konsole, ob Ihrem AWS-Konto in der aktuellen eine Greengrass-Servicerolle zugeordnet istAWS-Region. Andernfalls werden Sie von der Konsole aufgefordert, AWS IoT Greengrass das Lesen und Schreiben von AWS Services in Ihrem Namen zu erlauben.

Wenn Sie die Berechtigung erteilen, prüft die Konsole, ob eine Rolle mit dem Namen in Ihrem `Greengrass_ServiceRole` vorhanden istAWS-Konto.

- Wenn die Rolle vorhanden ist, fügt die Konsole die Servicerolle an Ihre AWS-Konto in der aktuellen anAWS-Region.

- Wenn die Rolle nicht vorhanden ist, erstellt die Konsole eine standardmäßige Greengrass-Servicerolle und fügt sie Ihrem AWS-Konto in der aktuellen anAWS-Region.

> **Note**
>
> Wenn Sie eine Servicerolle mit benutzerdefinierten Rollenrichtlinien erstellen möchten, verwenden Sie die IAM-Konsole, um die Rolle zu erstellen oder zu ändern. Weitere Informationen finden Sie unter Erstellen einer Rolle zum Delegieren von Berechtigungen an einen -AWSService oder Ändern einer Rolle im IAM-Benutzerhandbuch. https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_manage_modify.html 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:SourceAccount` globalen Bedingungskontextschlüssel `aws:SourceArn` und in Ihre Vertrauensrichtlinie aufzunehmen, um das Sicherheitsproblem des verwirrten Stellvertreters zu vermeiden. Die Bedingungskontextschlüssel schränken den Zugriff so ein, dass nur die Anforderungen zugelassen werden, die vom angegebenen Konto und Greengrass Workspace stammen. Weitere Informationen zum Confused-Deputy-Problem finden Sie Vermeidung des Problems des verwirrten Stellvertreters (dienstübergreifend).
> Wenn Sie eine Servicerolle erstellen, kehren Sie zur -AWS IoTKonsole zurück und fügen Sie die Rolle an Ihre anAWS-Konto. Sie können dies unter Greengrass-Servicerolle auf der Seite Einstellungen tun.

Ändern der Greengrass-Servicerolle (Konsole)

Gehen Sie wie folgt vor, um eine andere Greengrass-Servicerolle auszuwählen, die an Ihre AWS-Konto in der AWS-Region aktuell ausgewählten in der Konsole angefügt werden soll.

1. Navigieren Sie zur AWS IoT-Konsole.
2. Wählen Sie im Navigationsbereich Settings (Einstellungen).
3. Wählen Sie unter Greengrass-Servicerolle die Option Rolle ändern aus.

   Das Dialogfeld Greengrass-Servicerolle aktualisieren wird geöffnet und zeigt die IAM-Rollen in Ihrem anAWS-Konto, die AWS IoT Greengrass als vertrauenswürdige Entität definieren.
4. Wählen Sie die anzufügende Greengrass-Servicerolle aus.
5. Wählen Sie Rolle anfügen aus.

Trennen der Greengrass-Servicerolle (Konsole)

Gehen Sie wie folgt vor, um die Greengrass-Servicerolle von Ihrem AWS Konto in der aktuellen zu trennenAWS-Region. Dadurch werden Berechtigungen für für den Zugriff AWS IoT Greengrass auf AWS Services in der aktuellen widerrufenAWS-Region.

> **Important**
>
> Durch das Trennen der Servicerolle können aktive Operationen unterbrochen werden.

1. Navigieren Sie zur AWS IoT-Konsole.
2. Wählen Sie im Navigationsbereich Settings (Einstellungen).
3. Wählen Sie unter Greengrass-Servicerolle die Option Rolle trennen aus.
4. 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. Um alle Rollen AWS IoT Greengrass zu finden, die zulassen, dass Berechtigungen in Ihrem Namen übernimmt, suchen Sie in der IAM-Konsole auf der Seite Rollen nach Rollen, die AWS service: greengrass in der Spalte Vertrauenswürdige Entitäten enthalten.

Verwalten der Greengrass-Servicerolle (CLI)

In den folgenden Verfahren gehen wir davon aus, dass installiert und für die Verwendung Ihres konfiguriert AWS Command Line Interface istAWS-Konto. Weitere Informationen finden Sie unter Installieren, Aktualisieren und Deinstallieren der AWS CLI und Konfigurieren der AWS CLI im AWS Command Line Interface -Benutzerhandbuch.

Sie können die AWS CLI für die folgenden Rollenverwaltungsaufgaben verwenden:

Themen

Abrufen der Greengrass-Servicerolle (CLI)

Gehen Sie wie folgt vor, um herauszufinden, ob eine Greengrass-Servicerolle Ihrem AWS-Konto in einem zugeordnet istAWS-Region.

- Rufen Sie die Servicerolle ab. Ersetzen Sie *region* durch Ihre AWS-Region (z. B. `us-west-2`).

  ```
  aws greengrassv2 get-service-role-for-account --region region
  ```

  Wenn Ihrem Konto bereits eine Greengrass-Servicerolle zugeordnet ist, gibt die Anforderung die folgenden Rollenmetadaten zurück.

  ```
  {
    "associatedAt": "timestamp",
    "roleArn": "arn:aws:iam::account-id:role/path/role-name"
  }
  ```

  Wenn die Anforderung keine Rollenmetadaten zurückgibt, müssen Sie die Servicerolle (falls sie nicht vorhanden ist) erstellen und Ihrem Konto in der zuordnenAWS-Region.

Erstellen der Greengrass-Servicerolle (CLI)

Führen Sie die folgenden Schritte aus, um eine Rolle zu erstellen und sie Ihrem zuzuordnenAWS-Konto.

So erstellen Sie die Servicerolle mit IAM

1. Erstellen Sie die Rolle mit einer Vertrauensrichtlinie, die AWS IoT Greengrass die Annahme der Rolle erlaubt. In diesem Beispiel wird eine Rolle namens `Greengrass_ServiceRole` erstellt, aber Sie können einen anderen Namen verwenden. Wir empfehlen Ihnen, auch die `aws:SourceAccount` globalen Bedingungskontextschlüssel `aws:SourceArn` und in Ihre Vertrauensrichtlinie aufzunehmen, um das Sicherheitsproblem des verwirrten Stellvertreters zu vermeiden. Die Bedingungskontextschlüssel schränken den Zugriff so ein, dass nur die Anforderungen zugelassen werden, die vom angegebenen Konto und Greengrass Workspace

stammen. Weitere Informationen zum Confused-Deputy-Problem finden Sie Vermeidung des Problems des verwirrten Stellvertreters (dienstübergreifend).

Linux or Unix

```
aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-
document '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "greengrass.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
        },
        "StringEquals": {
          "aws:SourceAccount": "account-id"
        }
      }
    }
  ]
}'
```

Windows Command Prompt (CMD)

```
aws iam create-role --role-name Greengrass_ServiceRole --assume-role-
policy-document "{\"Version\":\"2012-10-17\",\"Statement\":[{\"Effect
\":\"Allow\",\"Principal\":{\"Service\":\"greengrass.amazonaws.com\"},
\"Action\":\"sts:AssumeRole\",\"Condition\":{\"ArnLike\":{\"aws:SourceArn
\":\"arn:aws:greengrass:region:account-id:*\"},\"StringEquals\":
{\"aws:SourceAccount\":\"account-id\"}}}]}"
```

PowerShell

```
aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-
document '{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "greengrass.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
        },
        "StringEquals": {
          "aws:SourceAccount": "account-id"
        }
      }
    }
  ]
}'
```

2. 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 Ihrem AWS-Konto

- Weisen Sie die Rolle Ihrem Konto zu. Ersetzen Sie *`role-arn`* durch den ARN der Servicerolle und *`region`* durch Ihre AWS-Region (z. B. `us-west-2`).

```
aws greengrassv2 associate-service-role-to-account --role-arn role-arn --
region region
```

Bei Erfolg gibt die Anforderung die folgende Antwort zurück.

```
{
  "associatedAt": "timestamp"
}
```

Entfernen der Greengrass-Servicerolle (CLI)

Führen Sie die folgenden Schritte aus, um die Zuordnung der Greengrass-Servicerolle zu Ihrem aufzuhebenAWS-Konto.

- Haben Sie die Zuordnung der Servicerolle bei Ihrem Konto auf. Ersetzen Sie *region* durch Ihre AWS-Region (z. B. `us-west-2`).

  ```
  aws greengrassv2 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 nicht in einem verwendenAWS-Region. Verwenden Sie zuerst delete-role-policy, um die verwaltete `AWSGreengrassResourceAccessRolePolicy`-Richtlinie von der Rolle zu lösen, und verwenden Sie dann delete-role, um die Rolle zu löschen. Weitere Informationen zum Löschen von Rollen finden Sie unter Löschen von Rollen oder Instance-Profilen im IAM-Benutzerhandbuch.

Weitere Informationen finden Sie auch unter

- Erstellen einer Rolle zum Delegieren von Berechtigungen an einen -AWSService im IAM-Benutzerhandbuch
- Ändern einer Rolle im IAM-Benutzerhandbuch
- Löschen von Rollen oder Instance-Profilen im IAM-Benutzerhandbuch
- AWS IoT Greengrass -Befehle in der -AWS CLIBefehlsreferenz
  - associate-service-role-to-Konto
  - disassociate-service-role-from-Konto
  - get-service-role-for-Konto

- IAM-Befehle in der -AWS CLIBefehlsreferenz
  - attach-role-policy
  - create-role
  - delete-role
  - delete-role-policy

AWS Von verwaltete Richtlinien für AWS IoT Greengrass

Eine von AWS verwaltete Richtlinie ist eine eigenständige Richtlinie, die von AWS erstellt und verwaltet wird. Von AWS verwaltete Richtlinien stellen Berechtigungen für viele häufige Anwendungsfälle bereit, damit Sie beginnen können, Benutzern, Gruppen und Rollen Berechtigungen zuzuweisen.

Beachten Sie, dass AWS-verwaltete Richtlinien möglicherweise nicht die geringsten Berechtigungen für Ihre spezifischen Anwendungsfälle gewähren, da sie für alle AWS-Kunden verfügbar sind. Wir empfehlen Ihnen, die Berechtigungen weiter zu reduzieren, indem Sie kundenverwaltete Richtlinien definieren, die speziell auf Ihre Anwendungsfälle zugeschnitten sind.

Die Berechtigungen, die in den von AWS verwalteten Richtlinien definiert sind, können nicht geändert werden. Wenn AWS Berechtigungen aktualisiert, die in einer von AWS verwalteten Richtlinie definiert werden, wirkt sich das Update auf alle Prinzipalidentitäten (Benutzer, Gruppen und Rollen) aus, denen die Richtlinie zugeordnet ist. AWSaktualisiert am wahrscheinlichsten eine von AWS verwaltete Richtlinie, wenn ein neuer AWS-Service gestartet wird oder neue API-Operationen für bestehende Services verfügbar werden.

Weitere Informationen finden Sie unter Von AWS verwaltete Richtlinien im IAM-Benutzerhandbuch.

Themen

- AWS verwaltete Richtlinie: AWSGreengrassFullAccess
- AWS verwaltete Richtlinie: AWSGreengrassReadOnlyAccess
- AWS verwaltete Richtlinie: AWSGreengrassResourceAccessRolePolicy
- AWS IoT Greengrass-Aktualisierungen für AWS verwaltete Richtlinien

AWS verwaltete Richtlinie: AWSGreengrassFullAccess

Sie können die `AWSGreengrassFullAccess`-Richtlinie an Ihre IAM-Identitäten anfügen.

Diese Richtlinie gewährt Administratorberechtigungen, die einem Schulleiter vollen Zugriff auf alle gewährenAWS IoT GreengrassAktionen.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen:

- `greengrass`— Ermöglicht Schulleitern vollen Zugriff auf alleAWS IoT GreengrassAktionen.

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

AWS verwaltete Richtlinie: AWSGreengrassReadOnlyAccess

Sie können die `AWSGreengrassReadOnlyAccess`-Richtlinie an Ihre IAM-Identitäten anfügen.

Diese Richtlinie gewährt nur Leserechte, die es einem Schulleiter ermöglichen, Informationen inAWS IoT Greengrass. Principals mit diesen Berechtigungen können beispielsweise die Liste der Komponenten einsehen, die auf einem Greengrass-Core-Gerät bereitgestellt wurden, aber sie können kein Deployment erstellen, um die Komponenten zu ändern, die auf diesem Gerät ausgeführt werden.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen:

- `greengrass`— Ermöglicht es Schulleitern, Aktionen auszuführen, die entweder eine Liste von Elementen oder Details zu einem Element zurückgeben. Dazu gehören API-Operationen, die beginnen mit`List`oder`Get`.

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

AWS verwaltete Richtlinie: AWSGreengrassResourceAccessRolePolicy

Sie können das anhängen`AWSGreengrassResourceAccessRolePolicy`Richtlinie für Ihre IAM-Entitäten.AWS IoT Greengrassordnet diese Richtlinie auch einer Servicerolle zu, die Folgendes ermöglichtAWS IoT Greengrassum Aktionen in Ihrem Namen durchzuführen. Weitere Informationen finden Sie unter Greengrass-Servicerolle.

Diese Richtlinie gewährt Administratorberechtigungen, die Folgendes ermöglichenAWS IoT Greengrassum wichtige Aufgaben auszuführen, wie z. B. das Abrufen Ihrer Lambda-Funktionen, die VerwaltungAWS IoTGerätebeschattung und Überprüfung von Greengrass-Client-Geräten.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen.

- `greengrass`— Verwalte die Greengrass-Ressourcen.
- `iot`(`*Shadow`) — VerwaltenAWS IoTSchatten, deren Namen die folgenden speziellen Kennungen enthalten. Diese Berechtigungen sind erforderlich, damitAWS IoT Greengrasskann mit Kerngeräten kommunizieren.
  - `*-gci`—AWS IoT Greengrassverwendet diesen Shadow, um Informationen zur Konnektivität von Kerngeräten zu speichern, sodass Client-Geräte die Hauptgeräte erkennen und eine Verbindung zu ihnen herstellen können.
  - `*-gcm`—AWS IoT Greengrass V1verwendet diesen Schatten, um das Kerngerät darüber zu informieren, dass das Zertifikat der Zertifizierungsstelle (CA) der Greengrass-Gruppe rotiert wurde.

  - `*-gda`—AWS IoT Greengrass V1verwendet diesen Schatten, um das Kerngerät über eine Bereitstellung zu informieren.
  - `GG_*`— Unbenutzt.
- `iot`(`DescribeThing`und`DescribeCertificate`) — Informationen abrufen überAWS IoTDinge und Zertifikate. Diese Berechtigungen sind erforderlich, damitAWS IoT Greengrasskann Client-Geräte überprüfen, die eine Verbindung zu einem Core-Gerät herstellen. Weitere Informationen finden Sie unter Interagieren mit lokalen IoT-Geräten.
- `lambda`— Informationen abrufen überAWS LambdaFunktionen. Diese Erlaubnis ist erforderlich, damitAWS IoT Greengrass V1kann Lambda-Funktionen auf Greengrass-Kernen bereitstellen. Weitere Informationen finden Sie unterFühren Sie die Lambda-Funktion auf dem ausAWS IoT GreengrassKernin derAWS IoT Greengrass V1Leitfaden für Entwickler.
- `secretsmanager`— Ruft den Wert von abAWS Secrets ManagerGeheimnisse, deren Namen beginnen mit`greengrass-`. Diese Erlaubnis ist erforderlich, damitAWS IoT Greengrass V1kann Secrets Manager-Geheimnisse für Greengrass-Kerne bereitstellen. Weitere Informationen finden Sie unterVerteile Geheimnisse für dieAWS IoT GreengrassKernin derAWS IoT Greengrass V1Leitfaden für Entwickler.
- `s3`— Ruft Dateiobjekte aus S3-Buckets ab, deren Namen Folgendes enthalten`greengrass`oder`sagemaker`. Diese Berechtigungen sind erforderlich, damitAWS IoT Greengrass V1kann Ressourcen für maschinelles Lernen bereitstellen, die Sie in S3-Buckets speichern. Weitere Informationen finden Sie unterRessourcen für maschinelles Lernenin derAWS IoT Greengrass V1Leitfaden für Entwickler.
- `sagemaker`— Informationen über Amazon abrufenSageMakerInferenzmodelle für maschinelles Lernen. Diese Erlaubnis ist erforderlich, damitAWS IoT Greengrass V1kann ML-Modelle auf Greengrass-Kernen bereitstellen. Weitere Informationen finden Sie unterInferenz für maschinelles Lernen durchführenin derAWS IoT Greengrass V1Leitfaden für Entwickler.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowGreengrassAccessToShadows",
            "Action": [
                "iot:DeleteThingShadow",
                "iot:GetThingShadow",
                "iot:UpdateThingShadow"
            ],
```

```
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iot:*:*:thing/GG_*",
                "arn:aws:iot:*:*:thing/*-gcm",
                "arn:aws:iot:*:*:thing/*-gda",
                "arn:aws:iot:*:*:thing/*-gci"
            ]
        },
        {
            "Sid": "AllowGreengrassToDescribeThings",
            "Action": [
                "iot:DescribeThing"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iot:*:*:thing/*"
        },
        {
            "Sid": "AllowGreengrassToDescribeCertificates",
            "Action": [
                "iot:DescribeCertificate"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iot:*:*:cert/*"
        },
        {
            "Sid": "AllowGreengrassToCallGreengrassServices",
            "Action": [
                "greengrass:*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Sid": "AllowGreengrassToGetLambdaFunctions",
            "Action": [
                "lambda:GetFunction",
                "lambda:GetFunctionConfiguration"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Sid": "AllowGreengrassToGetGreengrassSecrets",
            "Action": [
```

```
                "secretsmanager:GetSecretValue"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:secretsmanager:*:*:secret:greengrass-*"
        },
        {
            "Sid": "AllowGreengrassAccessToS3Objects",
            "Action": [
                "s3:GetObject"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::*Greengrass*",
                "arn:aws:s3:::*GreenGrass*",
                "arn:aws:s3:::*greengrass*",
                "arn:aws:s3:::*Sagemaker*",
                "arn:aws:s3:::*SageMaker*",
                "arn:aws:s3:::*sagemaker*"
            ]
        },
        {
            "Sid": "AllowGreengrassAccessToS3BucketLocation",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Sid": "AllowGreengrassAccessToSageMakerTrainingJobs",
            "Action": [
                "sagemaker:DescribeTrainingJob"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:sagemaker:*:*:training-job/*"
            ]
        }
    ]
}
```

AWS IoT Greengrass-Aktualisierungen für AWS verwaltete Richtlinien

Einzelheiten zu Updates finden Sie unterAWSverwaltete Richtlinien fürAWS IoT Greengrassab dem Zeitpunkt, an dem dieser Dienst begann, diese Änderungen zu verfolgen. Um automatische Benachrichtigungen über Änderungen an dieser Seite zu erhalten, abonnieren Sie den RSS-Feed auf derAWS IoT Greengrass V2Seite „Dokumentenverlauf“.

Änderung	Beschreibung	Datum
AWS IoT Greengrass hat die Änderungsverfolgung gestartet	AWS IoT Greengrass hat mit der Verfolgung von Änderungen für seine AWS-verwalteten Richtlinien begonnen.	2. Juli 2021

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::account-id:*`.

Für ein Beispiel für eine Richtlinie, die die`aws:SourceArn`und`aws:SourceAccount`Globale Bedingungskontextschlüssel sieheErstellen der Greengrass-Servicerolleaus.

Problembehandlung bei Identitäts- und Zugriffsproblemen für AWS IoT Greengrass

Verwenden Sie die folgenden Informationen, um häufige Probleme zu diagnostizieren und zu beheben, die beim Arbeiten mit AWS IoT Greengrass und IAM auftreten könnten.

Problembereiche

- Ich bin nicht autorisiert, eine Aktion in AWS IoT Greengrass auszuführen
- Ich bin nicht zur Ausführung von iam autorisiert:PassRole
- Ich bin Administrator und möchte anderen Zugriff auf AWS IoT Greengrass gewähren.
- Ich möchte Personen außerhalb meines AWS-Konto Zugriff auf meine AWS IoT Greengrass-Ressourcen gewähren

Hilfe zur allgemeinen Problembehandlung finden Sie unter Fehlerbehebung.

Ich bin nicht autorisiert, eine Aktion in AWS IoT Greengrass auszuführen

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`mateojackson`IAM-Benutzer versucht, Details zu einem Core-Gerät anzuzeigen, hat aber keine`greengrass:GetCoreDevice`Berechtigungen.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to
 perform: greengrass:GetCoreDevice on resource: arn:aws:greengrass:us-
west-2:123456789012:coreDevices/MyGreengrassCore
```

In diesem Fall bittet Mateo seinen Administrator um die Aktualisierung seiner Richtlinien, um unter Verwendung der Aktion `arn:aws:greengrass:us-west-2:123456789012:coreDevices/MyGreengrassCore` auf die Ressource `greengrass:GetCoreDevice` zugreifen zu können.

Im Folgenden sind allgemeine IAM-Probleme mitAWS IoT Greengrass.

Ich bin nicht zur Ausführung von iam autorisiert:PassRole

Wenn Sie die Fehlermeldung erhalten, dass Sie nicht zur Ausführung von nicht autorisiert sind`iam:PassRole`-Aktion müssen Ihre Richtlinien aktualisiert werden, um eine Rolle an übergeben zu könnenAWS IoT Greengrass.

Einige AWS-Services erlauben die Übergabe einer vorhandenen Rolle an diesen Service, sodass keine neue Servicerolle oder serviceverknüpfte Rolle erstellt werden muss. Hierzu benötigen Sie Berechtigungen für die Übergabe der Rolle an den Service.

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 Service.

```
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.

Wenden Sie sich an Ihren AWS-Administrator, falls Sie weitere Unterstützung benötigen. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen odzur Verfügung gestellt.

Ich bin Administrator und möchte anderen Zugriff auf AWS IoT Greengrass gewähren.

Um anderen Personen oder einer Anwendung Zugriff auf AWS IoT Greengrass zu gewähren, müssen Sie eine IAM-Entität (Benutzer oder Rolle) für die Person oder Anwendung erstellen, die Zugriff benötigt. Sie werden die Anmeldeinformationen für diese Einrichtung verwenden, um auf AWS zuzugreifen. Anschließend müssen Sie der Entität eine Richtlinie anfügen, die dieser die korrekten Berechtigungen in AWS IoT Greengrass gewährt.

Informationen zum Einstieg finden Sie unter Erstellen Ihrer ersten delegierten IAM-Benutzer und -Gruppen im IAM-Benutzerhandbuch.

Ich möchte Personen außerhalb meines AWS-Konto Zugriff auf meine AWS IoT Greengrass-Ressourcen gewähren

Sie können eine IAM-Rolle erstellen, die Benutzer in anderen Konten oder Personen außerhalb Ihrer Organisation für den Zugriff auf IhreAWSRessourcen. Sie können angeben, welchen Personen vertraut werden soll, so dass diese die Rolle übernehmen können. Weitere Informationen finden Sie unterGewähren von Zugriff für einen IAM-Benutzer auf einen anderenAWS-Kontodie du besitztundGewähren von Zugriff aufAWS-Kontoist im Besitz von Drittenin derIAM User Guide.

AWS IoT Greengrass unterstützt keinen kontenübergreifenden Zugriff auf der Grundlage ressourcenbasierter Richtlinien oder Zugriffskontrolllisten (Access Control Lists, ACLs).

Zulassen von Gerätedatenverkehr über einen Proxy oder eine Firewall

Greengrass-Core-Geräte und Greengrass-Komponenten führen ausgehende Anfragen an -AWSServices und andere Websites aus. Als Sicherheitsmaßnahme können Sie ausgehenden Datenverkehr auf eine kleine Anzahl von Endpunkten und Ports beschränken. Sie können die folgenden Informationen über Endpunkte und Ports verwenden, um den Geräteverkehr über einen Proxy, eine Firewall oder eine Amazon-VPC-Sicherheitsgruppe einzuschränken. Weitere Informationen zum Konfigurieren eines Core-Geräts für die Verwendung eines Proxys finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy.

Themen

- Endpunkte für den grundlegenden Betrieb
- Endpunkte für die Installation mit automatischer Bereitstellung
- Endpunkte für AWSvon bereitgestellte Komponenten

Endpunkte für den grundlegenden Betrieb

Greengrass-Core-Geräte verwenden die folgenden Endpunkte und Ports für den grundlegenden Betrieb.

Abrufen von AWS IoT Endpunkten

Rufen Sie die AWS IoT Endpunkte für Ihr ab AWS-Kontound speichern Sie sie zur späteren Verwendung. Ihr Gerät verwendet diese Endpunkte, um eine Verbindung zu herzustellenAWS IoT. Gehen Sie wie folgt vor:

1. Rufen Sie den AWS IoT Datenendpunkt für Ihr abAWS-Konto.

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

   Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anforderung erfolgreich ist.

   ```
   {
     "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
   }
   ```

2. Rufen Sie den AWS IoT Anmeldeinformationsendpunkt für Ihr abAWS-Konto.

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

   Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anforderung erfolgreich ist.

   ```
   {
     "endpointAddress": "device-credentials-prefix.credentials.iot.us-
   west-2.amazonaws.com"
   }
   ```

Endpunkt	Port	Erforderlich	Beschreibung
`greengrass-ats.iot. region.amazonaws.com`	8443 oder 443	Ja	Wird für Operationen auf Datenebene verwendet, z. B. für die Installat

Endpunkt	Port	Erforderlich	Beschreibung
			ion von Bereitstellungen und die Arbeit mit Clientgeräten.
device-data-prefix -ats.iot.*region*.amazonaws.com	MQTT: 8883 oder 443 HTTPS: 8443 oder 443	Ja	Wird für Operationen auf Datenebene für die Geräteverwaltung verwendet, z. B. MQTT-Kommunikation und Schattensynchronisierung mit AWS IoT Core.

Endpunkt	Port	Erforderlich	Beschreibung
device-credentials-prefix .credentials.iot. *region*.amazonaws.com	443	Ja	Wird verwendet, um AWS Anmeldeinformationen zu erhalten, die das Core-Gerät verwendet, um Komponentenartefakte von Amazon S3 herunterzuladen und andere Vorgänge auszuführen. Weitere Informationen finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit

Endpunkt	Port	Erforderlich	Beschreibung
			Diensten AWS.

`*.s3.amazonaws.com`
 	
`*.s3.`*`region`*`.amazonaws.com`
	443	Ja	Wird für Bereitstellungen verwendet. Dieses Format enthält das * Zeichen , da Endpunktpräfixe intern gesteuert werden und sich jederzeit ändern können.

Endpunkt	Port	Erforderlich	Beschreibung
`data.iot.` *region*`.amazonaws.com`	443	Nein	Erforderlich, wenn das Core-Gerät eine Version des Greengrass-Kerns vor v2.4.0 ausführt und für die Verwendung eines Netzwerk-Proxys konfiguriert ist. Das Core-Gerät verwendet diesen Endpunkt für die MQTT-Kommunikation mit AWS IoT Core, wenn es sich hinter einem Proxy befindet. Weitere

Endpunkt	Port	Erforderlich	Beschreibung
			Informationen finden Sie unter Konfigurieren Sie einen Netzwerk-Proxy.

Endpunkte für die Installation mit automatischer Bereitstellung

Greengrass-Core-Geräte verwenden die folgenden Endpunkte und Ports, wenn Sie die AWS IoT Greengrass Core-Software mit automatischer Ressourcenbereitstellung installieren.

Endpunkt	Port	Erforderlich	Beschreibung
`iot.region.amazonaws.com`	443	Ja	Wird verwendet, um AWS IoT Ressourcen zu erstellen und Informationen über vorhandene AWS IoT Ressourcen abzurufen.

Endpunkt	Port	Erforderlich	Beschreibung
`iam.amazonaws.com`	443	Ja	Wird verwendet, um IAM-Ressourcen zu erstellen und Informationen über vorhandene IAM-Ressourcen abzurufen.
`sts.region.amazonaws.com`	443	Ja	Wird verwendet, um die ID Ihres AWS-Kontos abzurufen.

Endpunkt	Port	Erforderlich	Beschreibung
`greengrass. region.amazonaws.com`	443	Nein	Erforderlich, wenn Sie das `--deploy-dev-tools` Argument verwenden, um die Greengrass-CLI-Komponente auf dem Core-Gerät bereitzustellen.

Endpunkte für AWSvon bereitgestellte Komponenten

Greengrass-Core-Geräte verwenden je nachdem, welche Softwarekomponenten sie ausführen, zusätzliche Endpunkte. Die Endpunkte, die jede von AWSbereitgestellte Komponente benötigt, finden Sie im Abschnitt Anforderungen auf der Seite jeder Komponente in diesem Entwicklerhandbuch. Weitere Informationen finden Sie unter AWSVon bereitgestellte Komponenten.

Konformitätsvalidierung für AWS IoT Greengrass

Informationen darüber, ob AWS-Service ein AWS-Services in den Geltungsbereich bestimmter Compliance-Programme fällt, finden Sie unter Umfang nach Compliance-Programm AWS-Services unter . Wählen Sie dort das Compliance-Programm aus, an dem Sie interessiert sind. Allgemeine Informationen finden Sie unter AWS Compliance-Programme AWS .

Sie können Prüfberichte von Drittanbietern unter herunterladen AWS Artifact. Weitere Informationen finden Sie unter Berichte herunterladen unter .

Ihre Verantwortung für die Einhaltung der Vorschriften bei der Nutzung AWS-Services hängt von der Vertraulichkeit Ihrer Daten, den Compliance-Zielen Ihres Unternehmens und den geltenden Gesetzen und Vorschriften ab. 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 HIPAA-fähige Anwendungen erstellen AWS können.

  > **Note**
  >
  > AWS-Services Nicht alle sind HIPAA-fähig. Weitere Informationen finden Sie in der Referenz für HIPAA-berechtigte Services.

- AWS Compliance-Ressourcen — Diese Sammlung von Arbeitsmappen und Leitfäden gilt möglicherweise für Ihre Branche und Ihren Standort.
- 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 den 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)) zugeordnet.
- 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

wie PCI DSS zu erfüllen, indem es die in bestimmten Compliance-Frameworks vorgeschriebenen Anforderungen zur Erkennung von Eindringlingen erfüllt.

- 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 finden Sie unter AWS-Globale Infrastruktur.

Zusätzlich zur globalen AWS-Infrastruktur stellt AWS IoT Greengrass verschiedene Funktionen bereit, um Ihren Anforderungen in Bezug auf Ausfallsicherheit und Datensicherung zu erfüllen.

- Sie können ein Greengrass-Kerngerät so konfigurieren, dass Protokolle in das lokale Dateisystem und CloudWatch Protokolle. Wenn das Core-Gerät Konnektivität verliert, kann es weiterhin Nachrichten im Dateisystem protokollieren. Wenn es sich wieder verbindet, schreibt es die Protokollmeldungen in CloudWatch Protokolle. Weitere Informationen finden Sie unter Überwachen von AWS IoT Greengrass Protokollen.
- Wenn ein Kerngerät während einer Bereitstellung an Strom verliert, nimmt es die Bereitstellung nach demAWS IoT GreengrassDie Kernsoftware startet erneut.
- Wenn ein Core-Gerät seine Verbindung zum Internet verliert, können Greengrass-Clientgeräte weiterhin über das lokale Netzwerk kommunizieren.
- Sie können Greengrass-Komponenten erstellen, die lesenStream-Managerstreamt und sendet die Daten an lokale Speicherziele.

Sicherheit der Infrastruktur in AWS IoT Greengrass

Als verwalteter Service ist AWS IoT Greengrass durch die globalen Verfahren zur Gewährleistung der Netzwerksicherheit von AWS geschützt, die im Whitepaper Amazon Web Services: Übersicht über die Sicherheitsprozesse beschrieben werden.

Sie verwenden durch AWS veröffentlichte API-Aufrufe, um über das Netzwerk auf AWS IoT Greengrass zuzugreifen. Clients müssen Transport Layer Security (TLS) 1.2 oder höher unterstützen. Wir empfehlen TLS 1.3 oder höher. Clients müssen außerdem Cipher Suites mit PFS (Perfect Forward Secrecy) wie DHE (Ephemeral Diffie-Hellman) oder ECDHE (Elliptic Curve Ephemeral Diffie-Hellman) unterstützen. Die meisten modernen Systemen wie Java 7 und höher unterstützen diese Modi.

Anforderungen müssen mit einer Zugriffsschlüssel-ID und einem geheimen Zugriffsschlüssel signiert sein, der mit einem IAM-Prinzipal verknüpft ist. 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 eine Verbindung herzustellen und sich bei der 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. iWeitere Informationen finden Sie unter den folgenden Themen:

- DieDevice Defender-Komponente
- AWS IoT Device DefenderimAWS IoT CoreEntwicklerhandbuchaus.

In AWS IoT Greengrass-Umgebungen sollten Sie die folgenden Überlegungen berücksichtigen:

- Es ist Ihre Reponierbarkeit, um Ihre physischen Geräte, das Dateisystem auf Ihren Geräten und das lokale Netzwerk zu sichern.
- AWS IoT Greengrasserzwingt keine Netzwerkisolierung für benutzerdefinierte Greengrass-Komponenten, unabhängig davon, ob sie in einem Greengrass-Container ausgeführt werden oder nicht. Daher ist es möglich, dass Greengrass-Komponenten mit anderen Prozessen kommunizieren, die im System oder außerhalb des Systems über das Netzwerk ausgeführt werden.

Code-Integrität inAWS IoT Greengrass V2

AWS IoT Greengrassstellt Softwarekomponenten aus demAWS Cloudauf Geräten, die dieAWS IoT GreengrassCore-Software. Diese Softwarekomponenten umfassenAWS-bereitgestellte Komponentenundkundenspezifische Komponentendie du auf deine hochlädstAWS-Kontoaus. Jede Komponente besteht aus einem Rezept. Das Rezept definiert die Metadaten der Komponente und eine beliebige Anzahl von Artefakten, die Komponentenbinärdateien sind, wie kompilierter Code und statische Ressourcen. Komponentenartefakte werden in Amazon S3 gespeichert.

Während Sie Greengrass-Komponenten entwickeln und bereitstellen, befolgen Sie diese grundlegenden Schritte, die mit Komponentenartefakten in IhremAWS-Kontound auf Ihren Geräten:

1. Erstellen und laden Sie Artefakte in S3-Buckets hoch.
2. Erstellen Sie eine Komponente aus einem Rezept und Artefakten imAWS IoT GreengrassService, der einkryptografischer Hashvon jedem Artefakt.
3. Stellen Sie eine Komponente auf Greengrass-Kerngeräten bereit, die die Integrität jedes Artefakts herunterladen und überprüfen.

AWSist verantwortlich für die Aufrechterhaltung der Integrität von Artefakten, nachdem Sie Artefakte in S3-Buckets hochgeladen haben, einschließlich der Bereitstellung von Komponenten auf Greengrass-Kerngeräten. Sie sind dafür verantwortlich, Software-Artefakte zu sichern, bevor Sie die Artefakte in S3-Buckets hochladen. Sie sind auch dafür verantwortlich, den Zugriff auf Ressourcen in IhremAWS-Konto, einschließlich der S3-Buckets, in die Sie Komponentenartefakte hochladen.

> **Note**
>
> Amazon S3 bietet eine Funktion namens S3 Object Lock, mit der Sie vor Änderungen an Komponentenartefakten in S3-Buckets schützen könnenAWS-Kontoaus. Mit der S3-Objektsperre können Sie verhindern, dass Komponentenartefakte gelöscht oder überschrieben werden. Weitere Informationen finden Sie unterVerwenden der S3-ObjektsperreimAmazon Simple Storage Service — Benutzerhandbuchaus.

WannAWSveröffentlicht eine öffentliche Komponente und wenn Sie eine benutzerdefinierte Komponente hochladen,AWS IoT Greengrassberechnet einen kryptografischen Digest für jedes Komponentenartefakt.AWS IoT Greengrassaktualisiert das Komponentenrezept so, dass es den Digest jedes Artefakts und den Hash-Algorithmus enthält, der zur Berechnung dieses Digest verwendet wird. Dieser Digest garantiert die Integrität des Artefakts, denn wenn sich das Artefakt imAWS Cloudoder während des Downloads stimmt der Dateidigst nicht mit dem Digest übereinAWS IoT Greengrassspeichert im Komponentenrezept. Weitere Informationen finden Sie unterArtefakte in der Komponentenrezeptreferenzaus.

Wenn Sie eine Komponente auf einem Kerngerät bereitstellen, wird derAWS IoT GreengrassDie Kernsoftware lädt das Komponentenrezept und jedes Komponentenartefakt herunter, das das Rezept definiert. DieAWS IoT GreengrassDie Kernsoftware berechnet den Digest jeder heruntergeladenen Artefaktdatei und vergleicht sie mit dem Digest dieses Artefakts im Rezept. Wenn die Digests nicht übereinstimmen, schlägt die Bereitstellung fehl und dieAWS IoT GreengrassDie Kernsoftware löscht die heruntergeladenen Artefakte aus dem Dateisystem des Geräts. Weitere Informationen über die Verbindungserstellung zwischen Core-Devices undAWS IoT Greengrasssind gesichert, siheVerschlüsselung während der Übertragungaus.

Sie sind dafür verantwortlich, Komponentenartefaktdateien auf den Dateisystemen Ihrer Kerngeräte zu sichern. DieAWS IoT GreengrassKernsoftware speichert Artefakte `impackages`-Ordner im Greengrass-Stammordner. Sie können es verwendenAWS IoT Device Defenderum Kerngeräte zu analysieren, zu prüfen und zu überwachen. Weitere Informationen finden Sie unter Konfigurations- und Schwachstellenanalyse in AWS IoT Greengrass.

AWS IoT Greengrass und Schnittstellen-VPC-Endpunkte (AWS PrivateLink)

Sie können eine private Verbindung zwischen Ihrer VPC und der AWS IoT Greengrass Steuerebene herstellen, indem Sie einen Schnittstellen-VPC-Endpunkt erstellen. Sie können diesen Endpunkt verwenden, um Komponenten, Bereitstellungen und Core-Geräte im AWS IoT Greengrass Service zu verwalten. Schnittstellenendpunkte werden von unterstütztAWS PrivateLink, einer Technologie, mit der Sie privat ohne Internet-Gateway, NAT-Gerät, VPN-Verbindung oder AWS Direct-Connect-Verbindung auf AWS IoT Greengrass APIs zugreifen können. 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.

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

Bevor Sie einen Schnittstellen-VPC-Endpunkt für einrichtenAWS IoT Greengrass, lesen Sie die Eigenschaften und Einschränkungen von Schnittstellenendpunkten im Amazon-VPC-Benutzerhandbuch. Beachten Sie außerdem die folgenden Überlegungen:

- AWS IoT Greengrass unterstützt Aufrufe an alle API-Aktionen der Steuerebene von Ihrer VPC aus. Die Steuerebene umfasst Operationen wie CreateDeployment und ListEffectiveDeployments. Die Steuerebene enthält keine Operationen wie ResolveComponentCandidates und Discover, bei denen es sich um Operationen auf Datenebene handelt.
- VPC-Endpunkte für AWS IoT Greengrass werden derzeit in AWS China-Regionen nicht unterstützt.

Erstellen eines Schnittstellen-VPC-Endpunkts für Operationen auf AWS IoT Greengrass Steuerebene

Sie können einen VPC-Endpunkt für die AWS IoT Greengrass Steuerebene entweder über die Amazon-VPC-Konsole oder die AWS Command Line Interface (AWS CLI) erstellen. Weitere Informationen finden Sie unter Erstellung eines Schnittstellenendpunkts im Benutzerhandbuch für Amazon VPC.

Erstellen Sie einen VPC-Endpunkt für 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 anfügen, der den Zugriff auf Operationen auf AWS IoT Greengrass Steuerebene steuert. Die Richtlinie gibt die folgenden Informationen an:

- Prinzipal, der die Aktionen ausführen kann
- Die Aktionen, die der Prinzipal ausführen kann.
- Die Ressourcen, für die 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 für AWS IoT Greengrass-Aktionen

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:ListEffectiveDeployments"
            ],
            "Resource": "*"
        }
    ]
}
```

Betreiben eines -AWS IoT GreengrassCore-Geräts in der VPC

Sie können ein Greengrass-Core-Gerät betreiben und Bereitstellungen in VPC ohne öffentlichen Internetzugang durchführen. Sie müssen mindestens die folgenden VPC-Endpunkte mit den entsprechenden DNS-Aliassen einrichten. Weitere Informationen zum Erstellen und Verwenden von VPC-Endpunkten finden Sie unter Erstellen eines VPC-Endpunkts im Amazon-VPC-Benutzerhandbuch.

> **Note**
>
> Die VPC-Funktion zum automatischen Erstellen eines DNS-Datensatzes ist für - AWS IoT data und -AWS IoTAnmeldeinformationen deaktiviert. Um diese Endpunkte zu verbinden, müssen Sie manuell einen privaten DNS-Datensatz erstellen. Weitere Informationen finden Sie unter Privates DNS für Schnittstellenendpunkte. Weitere Informationen zu AWS IoT Core VPC-Einschränkungen finden Sie unter Einschränkungen von VPC-Endpunkten.

Voraussetzungen

- Sie müssen die AWS IoT Greengrass Core-Software mithilfe der manuellen Bereitstellungsschritte installieren. Weitere Informationen finden Sie unter Installieren Sie die AWS IoT Greengrass Core-Software mit manueller Ressourcenbereitstellung.

Einschränkungen

- Der Betrieb eines Greengrass-Core-Geräts in VPC wird in den China-Regionen und nicht unterstütztAWS GovCloud (US) Regions.
- Weitere Informationen zu den Einschränkungen von AWS IoT data und VPCAWS IoT-Endpunkten des Anmeldeinformationsanbieters finden Sie unter Einschränkungen.

Einrichten Ihres Greengrass-Core-Geräts für den Betrieb in der VPC

1. Rufen Sie die AWS IoT Endpunkte für Ihr ab AWS-Kontound speichern Sie sie zur späteren Verwendung. Ihr Gerät verwendet diese Endpunkte, um eine Verbindung zu herzustellenAWS IoT. Gehen Sie wie folgt vor:

   a. Rufen Sie den AWS IoT Datenendpunkt für Ihr abAWS-Konto.

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

   Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anforderung erfolgreich ist.

   ```
   {
     "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
   }
   ```

   b. Rufen Sie den AWS IoT Anmeldeinformationsendpunkt für Ihr abAWS-Konto.

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

   Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anforderung erfolgreich ist.

   ```
   {
     "endpointAddress": "device-credentials-prefix.credentials.iot.us-
   west-2.amazonaws.com"
   }
   ```

2. Erstellen Sie eine Amazon-VPC-Schnittstelle für - AWS IoT data und -AWS IoTAnmeldeinformationsendpunkte:

a. Navigieren Sie zur VPC-Endpunkte-Konsole, wählen Sie im linken Menü unter Virtual Private Cloud die Option Endpunkte und dann Endpunkt erstellen aus.

b. Geben Sie auf der Seite Endpunkt erstellen die folgenden Informationen an:

- Wählen Sie AWS-Services als Servicekategorie aus.
- Suchen Sie nach dem Servicenamen, indem Sie das Schlüsselwort `iot` eingeben. Wählen Sie in der Liste der angezeigten `iot`-Services den Endpunkt aus.

  Wenn Sie einen VPC-Endpunkt für die AWS IoT Core-Datenebene erstellen, wählen Sie den API-Endpunkt der AWS IoT Core-Datenebene für Ihre Region aus. Der Endpunkt wird das Format `com.amazonaws.`*`region`*`.iot.data` haben.

  Wenn Sie einen VPC-Endpunkt für den AWS IoT Core-Anmeldeinformationsanbieter erstellen, wählen Sie den Endpunkt des AWS IoT Core-Anmeldeinformationsanbieter für Ihre Region aus. Der Endpunkt wird das Format `com.amazonaws.`*`region`*`.iot.credentials` haben.

  > **Note**
  >
  > Der Dienstname für die AWS IoT Core-Datenebene in der Region China wird das folgende Format `cn.com.amazonaws.`*`region`*`.iot.data` haben. Das Erstellen von VPC-Endpunkten für den AWS IoT Core-Anmeldeinformationsanbieter wird in der Region China nicht unterstützt.

- Wählen Sie für VPC und Subnetze die VPC aus, in der Sie den Endpunkt erstellen möchten, und die Availability Zones (AZs), in denen Sie das Endpunktnetzwerk einrichten möchten.
- Wählen Sie für DNS-Namen aktivieren die Option Für diesen Endpunkt aktivieren. Weder die AWS IoT Core-Datenebene noch der AWS IoT Core-Anmeldeinformationsanbieter unterstützen bisher private DNS-Namen.
- Wählen Sie für Sicherheitsgruppe die Sicherheitsgruppen aus, die Sie den Endpunktnetzwerkschnittstellen zuordnen möchten.
- Optional können Sie Tags hinzufügen oder entfernen. Tags sind Name-Wert-Paare, die Sie verwenden, um sie Ihrem Endpunkt zuzuordnen.

c. Wählen Sie VPC-Endpunkt erstellen, um den Schnittstellenendpunkt zu erstellen.

3. Nachdem Sie den AWS PrivateLink-Endpunkt erstellt haben, sehen Sie auf der Registerkarte Details Ihres Endpunkts eine Liste mit DNS-Namen. Sie können einen dieser DNS-Namen verwenden, die Sie in diesem Abschnitt erstellt haben, um Ihre private gehostete Zone zu konfigurieren.
4. Erstellen Sie einen Amazon S3-Endpunkt. Weitere Informationen finden Sie unter Erstellen eines VPC-Endpunkts für Amazon S3.
5. Wenn Sie von AWSbereitgestellte Greengrass-Komponenten verwenden, sind möglicherweise zusätzliche Endpunkte und Konfigurationen erforderlich. Um die Endpunktanforderungen anzuzeigen, wählen Sie die Komponente aus der Liste der von AWSbereitgestellten Komponenten aus und sehen Sie sich den Abschnitt Anforderungen an. Die Anforderungen an die Log Manager-Komponente weisen beispielsweise darauf hin, dass diese Komponente in der Lage sein muss, ausgehende Anfragen an den Endpunkt auszuführen`logs.`*`region`*`.amazonaws.com`.

   Wenn Sie Ihre eigene Komponente verwenden, müssen Sie möglicherweise die Abhängigkeiten überprüfen und zusätzliche Tests durchführen, um festzustellen, ob zusätzliche Endpunkte erforderlich sind.
6. In der Greengrass-Kernkonfiguration `greengrassDataPlaneEndpoint` muss auf gesetzt sein**`iotdata`**. Weitere Informationen finden Sie unter Greengrass-Kernkonfiguration.
7. Wenn Sie sich in der `us-east-1` Region befinden, legen Sie den Konfigurationsparameter **`REGIONAL`** in der Greengrass-Kernkonfiguration `s3EndpointType` auf fest. Diese Funktion ist für Greengrass-Kernversionen 2.11.3 oder höher verfügbar.

Example Beispiel: Komponentenkonfiguration

```
{
"aws.greengrass.Nucleus": {
   "configuration": {
      "awsRegion": "us-east-1",
      "iotCredEndpoint": "xxxxxx.credentials.iot.region.amazonaws.com",
      "iotDataEndpoint": "xxxxxx-ats.iot.region.amazonaws.com",
      "greengrassDataPlaneEndpoint": "iotdata",
      "s3EndpointType": "REGIONAL"
      ...
     }
   }
}
```

Die folgende Tabelle enthält Informationen zu den entsprechenden benutzerdefinierten privaten DNS-Aliassen.

Service	Name des VPC-Endpunkt-Service	VPC-Endpunkttyp	Benutzerdefinierter privater DNS-Alias	Hinweise
AWS IoT data	`com.amazonaws.`*`region`*`.iot.d`	Schnittstelle	*`prefix`*`-ats.iot.`*`region`*`.amazonaws.com`	Der private DNS-Datensatz sollte mit dem AWS IoT data Endpunkt Ihres Kontos übereinstimmen: `aws iot describe-endpoint --endpoint-type iot:Data-ATS` .
AWS IoT-Anmeldedaten	`com.amazonaws.`*`region`*`.iot.credentials`	Schnittstelle	*`prefix`*`.credentials.iot.amazonaws.com`	Der private DNS-Eintrag sollte

Service	Name des VPC-Endpunkt-Service	VPC-Endpunkttyp	Benutzerdefinierter privater DNS-Alias	Hinweise
				mit dem Endpunkt Ihrer AWS IoT Kontoanmeldeinformationen übereinstimmen: `aws iot describe-endpoint --endpoint-type iot:CredentialProvider .`
Amazon S3	`com.amazonaws.` *`region`*`.s3`	Schnittstelle		Der DNS-Datensatz wird automatisch erstellt.

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 Rechte für Ihre Komponenten, indem Sie sie als Benutzer ohne Rechte ausführen. Komponenten sollten nicht als Root-Benutzer ausgeführt werden, es sei denn, dies ist unbedingt erforderlich.

Verwenden Sie die Mindestanzahl an Berechtigungen für IAM-Rollen. Beschränken Sie die Verwendung von*Platzhalter für`Action`und`Resource`Eigenschaften 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 Rechten gilt auch fürAWS IoTRichtlinien, die Sie an Ihren Greengrass-Kern anhängen.

Kodieren Sie Anmeldeinformationen nicht fest in Greengrass-Komponenten

Kodieren Sie Anmeldeinformationen nicht fest in Ihren benutzerdefinierten Greengrass-Komponenten. So schützen Sie Ihre Anmeldeinformationen besser:

- Um zu interagieren mitAWSDienste, definieren Sie Berechtigungen für bestimmte Aktionen und Ressourcen imDie zentrale Rolle des Greengrass-Gerätedienstes.
- Benutze dieSecret Manager-Komponenteum Ihre Zugangsdaten zu speichern. Oder, wenn die Funktion das verwendetAWSSDK, verwenden Sie Anmeldeinformationen aus der Standard-Anmeldeinformationsanbieterkette.

Keine Protokollierung sensibler Informationen

Sie sollten die Protokollierung von Anmeldeinformationen und anderen persönlich identifizierbaren Informationen (PII) verhindern. Wir empfehlen Ihnen, die folgenden Sicherheitsvorkehrungen zu implementieren, auch wenn für den Zugriff auf lokale Protokolle auf einem Kerngerät Root-Rechte und Zugriff auf erforderlich sind CloudWatch Für Protokolle sind IAM-Berechtigungen erforderlich.

- 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 vertraulichen Informationen in Ihren benutzerdefinierten Greengrass-Komponenten oder Lambda-Funktionen.

- Verwenden Sie keine vertraulichen Informationen in den Namen und IDs von Greengrass-Ressourcen:
  - Kerngeräte
  - Komponenten
  - Bereitstellungen
  - Logger

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.

Empfehlungen für die Cipher Suite

Greengrass wählt standardmäßig die neuesten TLS Cipher Suites aus, die auf dem Gerät verfügbar sind. Erwägen Sie, die Verwendung älterer Cipher Suites auf dem Gerät zu deaktivieren. Zum Beispiel CBC-Verschlüsselungssammlungen.

Weitere Informationen finden Sie imKonfiguration der Java-Kryptografie.

Weitere Informationen finden Sie auch unter

- Bewährte Sicherheitsverfahren inAWS IoT Corein derAWS IoTLeitfaden für Entwickler
- Zehn goldene Sicherheitsregeln für industrielle IoT-Lösungenauf derInternet der Dinge aufAWSOffizieller Blog

AWS IoT Device Tester Für AWS IoT Greengrass V2 verwenden

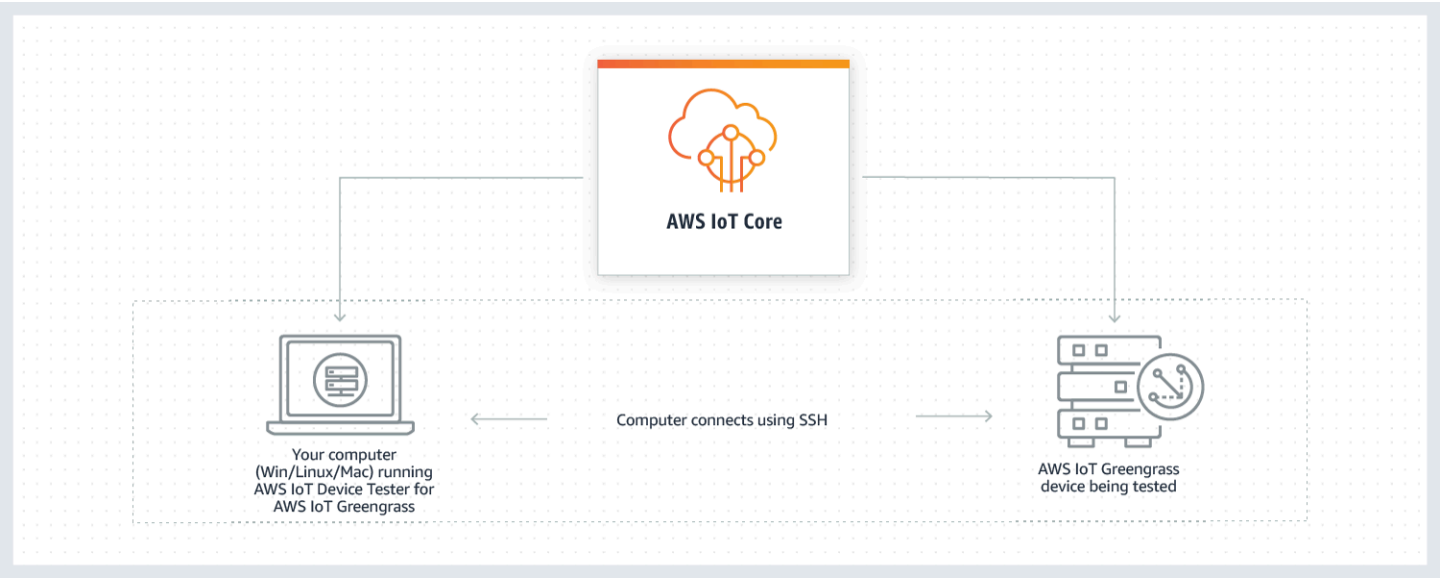
AWS IoT Device Tester (IDT) ist ein herunterladbares Test-Framework, mit dem Sie IoT-Geräte validieren können. Sie können IDT verwenden, AWS IoT Greengrass um die AWS IoT Greengrass Qualifizierungssuite auszuführen und benutzerdefinierte Testsuiten für Ihre Geräte zu erstellen und auszuführen.

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 AWS IoT Device Tester für AWS IoT Greengrass V2, um zu überprüfen, ob die AWS IoT Greengrass Core-Software auf Ihrer Hardware läuft und mit der kommunizieren kann AWS Cloud. Es führt auch end-to-end Tests mit durch AWS IoT Core. Es überprüft beispielsweise, ob Ihr Gerät Komponenten bereitstellen und aktualisieren kann.

Wenn Sie Ihre Hardware zum AWS Partner Gerätekatalog hinzufügen möchten, führen Sie die AWS IoT Greengrass Qualification Suite aus, um Testberichte zu erstellen, an die Sie sie einreichen können. AWS IoT Weitere Informationen finden Sie unter AWS Device Qualification Program.



IDT for AWS IoT Greengrass V2 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 besteht aus einzelnen Tests, die sich auf eine bestimmte Funktion beziehen, z. B. auf die Bereitstellung von Komponenten.

Weitere Informationen finden Sie unter Verwenden Sie IDT, um die AWS IoT Greengrass Qualification Suite auszuführen.

Benutzerdefinierte Testsuiten

Ab IDT v4.0.1 kombiniert IDT for AWS IoT Greengrass V2 ein standardisiertes Konfigurations-Setup und ein standardisiertes Ergebnisformat mit einer Testsuite-Umgebung, mit der Sie benutzerdefinierte Testsuiten für Ihre Geräte und Gerätesoftware entwickeln können. Sie können benutzerdefinierte Tests für Ihre eigene interne Validierung hinzufügen oder sie Ihren Kunden zur Geräteverifizierung zur Verfügung stellen.

Wie ein Testautor eine benutzerdefinierte Testsuite konfiguriert, bestimmt die Einstellungskonfigurationen, die für die Ausführung benutzerdefinierter Testsuiten erforderlich sind. Weitere Informationen finden Sie unter Verwenden Sie IDT, um Ihre eigenen Testsuiten zu entwickeln und auszuführen.

Unterstützte Versionen von AWS IoT Device Tester für AWS IoT Greengrass V2

In diesem Thema werden die unterstützten Versionen von IDT für AWS IoT Greengrass V2 aufgeführt. Als bewährte Methode empfehlen wir, dass Sie die neueste Version von IDT für AWS IoT Greengrass V2 verwenden, die Ihre Zielversion von AWS IoT Greengrass V2 unterstützt. Bei neuen Versionen von müssen Sie AWS IoT Greengrass möglicherweise eine neue Version von IDT für AWS IoT Greengrass V2 herunterladen. Sie erhalten eine Benachrichtigung, wenn Sie einen Testlauf starten, wenn IDT für AWS IoT Greengrass V2 nicht mit der von AWS IoT Greengrass Ihnen verwendeten Version kompatibel ist.

Durch das Herunterladen der Software stimmen Sie der AWS IoT Device Tester Lizenzvereinbarung zu.

> **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.

Aktuelle IDT-Version für V2 AWS IoT Greengrass

Sie können diese Version von IDT für AWS IoT Greengrass V2 mit der hier aufgeführten AWS IoT Greengrass Version verwenden.

IDT v4.9.4 für AWS IoT Greengrass

Unterstützte Versionen: AWS IoT Greengrass

- Greengrass Nucleus v2.12.0, v2.11.0, v2.10.0 und v2.9.5

IDT-Softwaredownloads:

- IDT v4.9.4 mit der Testsuite GGV2Q_2.5.4 für Linux
- IDT v4.9.4 mit Testsuite GGV2Q_2.5.4 für macOS
- IDT v4.9.4 mit der Testsuite GGV2Q_2.5.4 für Windows

Versionshinweise:

- Ermöglicht die Gerätevalidierung und Qualifizierung für Geräte, auf denen die AWS IoT Greengrass Core-Softwareversionen 2.12.0, 2.11.0, 2.10.0 und 2.9.5 ausgeführt werden.
- Entfernt Stream Manager- und Machine-Learning-Testgruppen.

Weitere Hinweise:

- Wenn Ihr Gerät ein HSM verwendet und Sie Nucleus 2.10.x verwenden, migrieren Sie zu Greengrass Nucleus Version 2.11.0 oder höher.

Testsuite-Version:

`GGV2Q_2.5.4`

- Veröffentlicht am 2024.05.03

Frühere IDT-Versionen für AWS IoT Greengrass

Die folgenden früheren Versionen von IDT für AWS IoT Greengrass V2 werden ebenfalls unterstützt.

IDT v4.9.3 für AWS IoT Greengrass

Unterstützte Versionen: AWS IoT Greengrass

- Greengrass Nucleus v2.12.0, v2.11.0, v2.10.0 und v2.9.5

IDT-Softwaredownloads:

- IDT v4.9.3 mit der Testsuite GGV2Q_2.5.3 für Linux
- IDT v4.9.3 mit Testsuite GGV2Q_2.5.3 für macOS
- IDT v4.9.3 mit der Testsuite GGV2Q_2.5.3 für Windows

Versionshinweise:

- Behebt ein Problem bei den Komponententests beim Testen eines Linux-Geräts von einem Windows-Host aus oder umgekehrt.
- Entfernt den `localcomponent` Testfall aus der `component` Testgruppe. Dieser Testfall ist für die Qualifizierung nicht mehr erforderlich.

Weitere Hinweise:

- Wenn Ihr Gerät ein HSM verwendet und Sie Nucleus 2.10.x verwenden, migrieren Sie zu Greengrass Nucleus Version 2.11.0 oder höher.

Testsuite-Version:

`GGV2Q_2.5.3`

- Veröffentlicht 2024.04.05

Nicht unterstützte Versionen von für V2 AWS IoT Device TesterAWS IoT Greengrass

In diesem Thema werden nicht unterstützte Versionen von IDT für V2 aufgeführt. AWS IoT Greengrass Nicht unterstützte Versionen erhalten keine Fehlerbehebungen oder Updates. Weitere Informationen finden Sie unter the section called “Support-Richtlinie für AWS IoT Device Tester für AWS IoT Greengrass”.

IDT v4.9.2 für AWS IoT Greengrass

Versionshinweise:

- Behebt ein Problem, bei dem die Lambda-Testsuite fehlschlägt, weil Java 8 veraltet ist.

Testsuite-Version:

`GGV2Q_2.5.2`

- Veröffentlicht am 2024.03.18

IDT v4.9.1 für AWS IoT Greengrass

Versionshinweise:

- Ermöglicht die Validierung und Qualifizierung von Geräten, auf denen die AWS IoT Greengrass Core-Softwareversionen 2.12.0, 2.11.0, 2.10.0 und 2.9.5 ausgeführt werden.
- Kleinere Fehlerbehebungen.

Testsuite-Version:

`GGV2Q_2.5.1`

- Veröffentlicht 2023.10.05

IDT v4.7.0 für AWS IoT Greengrass

Unterstützte Versionen: AWS IoT Greengrass

- Greengrass Nucleus v2.11.0, v2.10.0 und v2.9.5

Versionshinweise:

- Ermöglicht die Validierung und Qualifizierung von Geräten, auf denen die AWS IoT Greengrass Core-Softwareversionen 2.11.0, 2.10.0 und 2.9.5 ausgeführt werden.
- Fügt Unterstützung hinzu, um IDT-Benutzerdatenwerte im AWS Systems Manager Parameter Store zu speichern und sie mithilfe der Platzhaltersyntax in die Konfiguration abzurufen.
- Kleinere Fehlerbehebungen.

Testsuite-Version:

`GGV2Q_2.5.0`

- Veröffentlicht 2022.12.13

IDT v4.5.11 für AWS IoT Greengrass

Versionshinweise:

- Ermöglicht die Validierung und Qualifizierung von Geräten, auf denen die AWS IoT Greengrass Core-Softwareversionen 2.9.1, 2.9.0, 2.8.1, 2.8.0, 2.7.0 und 2.6.0 ausgeführt werden.
- Fügt Unterstützung hinzu, um PreInstalled Greengrass auf einem Kerngerät zu testen.
- Kleinere Fehlerbehebungen.

Testsuite-Version:

`GGV2Q_2.4.1`

- Veröffentlicht am 13.10.2022

IDT v4.5.8 für AWS IoT Greengrass

Versionshinweise:

- Ermöglicht die Validierung und Qualifizierung von Geräten, auf denen die AWS IoT Greengrass Core-Softwareversionen 2.7.0, 2.6.0 und 2.5.6 ausgeführt werden.
- Ermöglicht das Testen mit PreInstalled Greengrass auf einem Core-Gerät.
- Kleinere Fehlerbehebungen.

Testsuite-Version:

`GGV2Q_2.4.0`

- Veröffentlicht am 12.08.2022

IDT v4.5.3 für AWS IoT Greengrass

Versionshinweise:

- Ermöglicht die Validierung und Qualifizierung von Geräten, auf denen die AWS IoT Greengrass Core-Softwareversionen 2.7.0, 2.6.0, 2.5.6, 2.5.5, 2.5.4 und 2.5.3 ausgeführt werden.
- Aktualisiert den DockerApplicationManager Test zur Verwendung eines ECR-basierten Docker-Images.
- Kleinere Fehlerbehebungen.

Testsuite-Version:

`GGV2Q_2.3.1`

- Veröffentlicht am 15.04.2022

IDT v4.5.1 für AWS IoT Greengrass

Versionshinweise:

- Ermöglicht die Validierung und Qualifizierung von Geräten, auf denen die AWS IoT Greengrass Core-Software v2.5.3 ausgeführt wird.
- Integriert die Unterstützung für die Validierung und Qualifizierung von Linux-basierten Geräten, die ein Hardware-Sicherheitsmodul (HSM) zum Speichern des privaten Schlüssels und Zertifikats verwenden, die von der Core-Software verwendet werden. AWS IoT Greengrass

- Implementiert den neuen IDT-Testorchestrator für die Konfiguration benutzerdefinierter Testsuiten. Weitere Informationen finden Sie unter Konfigurieren Sie den IDT-Test-Orchestrator.
- Zusätzliche kleinere Fehlerkorrekturen.

Testsuite-Version:

`GGV2Q_2.3.0`

- Veröffentlicht am 11.01.2022

IDT v4.4.1 für AWS IoT Greengrass

Versionshinweise:

- Ermöglicht die Validierung und Qualifizierung von Geräten, auf denen die AWS IoT Greengrass Core-Software v2.5.2 ausgeführt wird.
- Integriert die Unterstützung für die Verwendung einer benutzerdefinierten IAM-Rolle als Token-Austauschrolle, die das getestete Gerät für die Interaktion mit Ressourcen annimmt. AWS

  Sie können die IAM-Rolle in der Datei angeben. `userdata.json` Wenn Sie eine benutzerdefinierte Rolle angeben, verwendet IDT diese Rolle, anstatt während des Testlaufs die Standard-Token-Exchange-Rolle zu erstellen.
- Zusätzliche kleinere Bugfixes.

Testsuite-Version:

`GGV2Q_2.2.1`

- Veröffentlicht 2021.12.12

IDT v4.4.0 für AWS IoT Greengrass

Versionshinweise:

- Ermöglicht die Validierung und Qualifizierung von Geräten, auf denen die AWS IoT Greengrass Core-Software v2.5.0 ausgeführt wird.
- Integriert die Unterstützung für die Validierung und Qualifizierung von Geräten, auf denen AWS IoT Greengrass Core-Software unter Windows ausgeführt wird.
- Unterstützt die Verwendung der Validierung öffentlicher Schlüssel für Secure Shell (SSH) - Geräteverbindungen.
- Verbessert die IAM-Richtlinie für IDT-Berechtigungen mit bewährten Sicherheitsmethoden.
- Zusätzliche kleinere Fehlerkorrekturen.

Testsuite-Version:

`GGV2Q_2.1.0`

- Veröffentlicht 2021.11.19

IDT v4.2.0 für AWS IoT Greengrass

Versionshinweise:

- Beinhaltet Unterstützung für die Qualifizierung der folgenden Funktionen auf Geräten, auf denen die AWS IoT Greengrass Core-Software v2.2.0 und spätere Versionen ausgeführt wird:
  - Docker — Überprüft, ob Geräte ein Docker-Container-Image von Amazon Elastic Container Registry (Amazon ECR) herunterladen können.
  - Maschinelles Lernen — Überprüft, ob Geräte mithilfe der Frameworks Deep Learning Runtime oder Lite ML Inferenzen aus maschinellem Lernen (ML) durchführen können. TensorFlow
  - Stream Manager — Überprüft, ob Geräte den Stream Manager herunterladen, installieren und ausführen können. AWS IoT Greengrass
- Ermöglicht die Validierung und Qualifizierung von Geräten, auf denen die AWS IoT Greengrass Core-Software v2.4.0, v2.3.0, v2.2.0 und v2.1.0 ausgeführt wird.
- *Gruppiert die Testprotokolle für jeden Testfall in einem separaten Ordner < > innerhalb des Verzeichnisses. test-case-id <device-tester-extract-location>*/results/*<execution-id>*/logs/*<test-group-id>*
- Zusätzliche kleinere Bugfixes.

Testsuite-Version:

`GGV2Q_2.0.1`

- Veröffentlicht 2021.08.31

IDT v4.1.0 für AWS IoT Greengrass

Versionshinweise:

- Ermöglicht die Validierung und Qualifizierung von Geräten, auf denen die AWS IoT Greengrass Core-Software v2.3.0, v2.2.0, v2.1.0 und v2.0.5 ausgeführt wird.
- Verbessert die `userdata.json` Konfiguration, da die Angabe der Eigenschaften und nicht mehr erforderlich ist. `GreengrassNucleusVersion` `GreengrassCLIVersion`
- Beinhaltet Unterstützung für Lambda und MQTT-Funktionsqualifizierung für AWS IoT Greengrass Core-Software v2.1.0 und spätere Versionen. Sie können jetzt IDT for AWS

IoT Greengrass V2 verwenden, um zu überprüfen, ob Ihr Kerngerät Lambda-Funktionen ausführen kann und ob das Gerät AWS IoT Core MQTT-Themen veröffentlichen und abonnieren kann.

- Verbessert die Protokollierungsfunktionen.
- Zusätzliche kleinere Fehlerkorrekturen.

Testsuite-Version:

`GGV2Q_1.1.1`

- Veröffentlicht 2021.06.18

IDT v4.0.2 für AWS IoT Greengrass

Versionshinweise:

- Ermöglicht die Validierung und Qualifizierung von Geräten, auf denen die AWS IoT Greengrass Core-Software v2.1.0 ausgeführt wird.
- Integriert die Unterstützung für Lambda- und MQTT-Funktionsqualifizierung für AWS IoT Greengrass Core-Software v2.1.0 und spätere Versionen. Sie können jetzt IDT for AWS IoT Greengrass V2 verwenden, um zu überprüfen, ob Ihr Kerngerät Lambda-Funktionen ausführen kann und ob das Gerät AWS IoT Core MQTT-Themen veröffentlichen und abonnieren kann.
- Verbessert die Protokollierungsfunktionen.
- Zusätzliche kleinere Fehlerkorrekturen.

Testsuite-Version:

`GGV2Q_1.1.1`

- Veröffentlicht 2021.05.05

IDT v4.0.1 für AWS IoT Greengrass

Versionshinweise:

- Ermöglicht die Validierung und Qualifizierung von Geräten, auf denen Software der AWS IoT Greengrass Version 2 ausgeführt wird.
- Ermöglicht es Ihnen, Ihre benutzerdefinierten Testsuiten mit AWS IoT Device Tester for zu entwickeln und auszuführen AWS IoT Greengrass. Weitere Informationen finden Sie unter Verwenden Sie IDT, um Ihre eigenen Testsuiten zu entwickeln und auszuführen.
- Stellt codesignierte IDT-Anwendungen für macOS und Windows bereit. Unter macOS müssen Sie möglicherweise eine Sicherheitsausnahme für IDT gewähren. Weitere Informationen finden Sie unter Sicherheitsausnahme auf macOS.

Testsuite-Version:

`GGV2Q_1.0.0`

- Veröffentlicht am 22.12.2020
- Die Testsuite führt nur die für die Qualifizierung erforderlichen Tests aus, es sei denn, Sie setzen die entsprechenden Tests `value` im `features` Array auf. `yes`

Laden Sie IDT für AWS IoT Greengrass V2 herunter

In diesem Thema werden die Optionen zum Herunterladen AWS IoT Device Tester für AWS IoT Greengrass V2 beschrieben. Sie können entweder einen der folgenden Links zum Herunterladen von Software verwenden oder den Anweisungen folgen, um IDT programmgesteuert herunterzuladen.

Themen

Durch das Herunterladen der Software stimmen Sie der AWS IoT Device Tester Lizenzvereinbarung zu.

> **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.

Laden Sie IDT manuell herunter

In diesem Thema werden die unterstützten Versionen von IDT für AWS IoT Greengrass V2 aufgeführt. Als bewährte Methode empfehlen wir, dass Sie die neueste Version von IDT für AWS IoT Greengrass V2 verwenden, die Ihre Zielversion von AWS IoT Greengrass V2 unterstützt. Bei neuen Versionen von müssen Sie AWS IoT Greengrass möglicherweise eine neue Version von IDT für AWS IoT Greengrass V2 herunterladen. Sie erhalten eine Benachrichtigung, wenn Sie einen Testlauf starten, wenn IDT für AWS IoT Greengrass V2 nicht mit der von AWS IoT Greengrass Ihnen verwendeten Version kompatibel ist.

IDT v4.9.4 für AWS IoT Greengrass

Unterstützte Versionen: AWS IoT Greengrass

- Greengrass Nucleus v2.12.0, v2.11.0, v2.10.0 und v2.9.5

IDT-Softwaredownloads:

- IDT v4.9.4 mit der Testsuite GGV2Q_2.5.4 für Linux
- IDT v4.9.4 mit Testsuite GGV2Q_2.5.4 für macOS
- IDT v4.9.4 mit der Testsuite GGV2Q_2.5.4 für Windows

Versionshinweise:

- Ermöglicht die Gerätevalidierung und Qualifizierung für Geräte, auf denen die AWS IoT Greengrass Core-Softwareversionen 2.12.0, 2.11.0, 2.10.0 und 2.9.5 ausgeführt werden.
- Entfernt Stream Manager- und Machine-Learning-Testgruppen.

Weitere Hinweise:

- Wenn Ihr Gerät ein HSM verwendet und Sie Nucleus 2.10.x verwenden, migrieren Sie zu Greengrass Nucleus Version 2.11.0 oder höher.

Testsuite-Version:

`GGV2Q_2.5.4`

- Veröffentlicht am 2024.05.03

Laden Sie IDT programmgesteuert herunter

IDT bietet eine API-Operation, mit der Sie eine URL abrufen können, über die Sie IDT programmgesteuert herunterladen können. Sie können diesen API-Vorgang auch verwenden, um zu überprüfen, ob Sie über die neueste Version von IDT verfügen. Dieser API-Vorgang hat den folgenden Endpunkt.

```
https://download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt
```

Um diesen API-Vorgang aufzurufen, benötigen Sie die Berechtigung, die **`iot-device-tester:LatestIdt`** Aktion auszuführen. Fügen Sie Ihre AWS Signatur hinzu und verwenden Sie `iot-device-tester` sie als Dienstnamen.

API-Anfrage

HostOs — Das Betriebssystem des Host-Computers. Wählen Sie aus den folgenden Optionen aus:

- `mac`
- `linux`
- `windows`

TestSuiteType — Der Typ der Testsuite. Wählen Sie die folgende Option:

`GGV2`— IDT für V2 AWS IoT Greengrass

ProductVersion

(Optional) Die Version des Greengrass-Kerns. Der Dienst gibt die neueste kompatible Version von IDT für diese Version von Greengrass Nucleus zurück. Wenn Sie diese Option nicht angeben, gibt der Dienst die neueste Version von IDT zurück.

API-Antwort

Die API-Antwort hat das folgende Format. Das `DownloadURL` beinhaltet eine Zip-Datei.

```
{
    "Success": True or False,
    "Message": Message,
    "LatestBk": {
        "Version": The version of the IDT binary,
        "TestSuiteVersion": The version of the test suite,
        "DownloadURL": The URL to download the IDT Bundle, valid for one hour
    }
 }
```

Beispiele

Sie können die folgenden Beispiele verwenden, um IDT programmgesteuert herunterzuladen. In diesen Beispielen werden Anmeldeinformationen verwendet, die Sie in den Umgebungsvariablen `AWS_ACCESS_KEY_ID` und `AWS_SECRET_ACCESS_KEY` speichern. Speichern Sie Ihre Anmeldeinformationen nicht in Ihrem Code, um die besten Sicherheitsvorkehrungen zu befolgen.

Example Beispiel: Herunterladen mit cURL Version 7.75.0 oder höher (Mac und Linux)

Wenn Sie die cURL-Version 7.75.0 oder höher haben, können Sie das `aws-sigv4` Flag verwenden, um die API-Anfrage zu signieren. In diesem Beispiel wird jq verwendet, um die Download-URL aus der Antwort zu analysieren.

> **Warning**
>
> Das `aws-sigv4` Flag erfordert, dass die Abfrageparameter der curl-GET-Anfrage in der Reihenfolge oder angegeben werden. HostOs/ProductVersion/TestSuiteType HostOs/TestSuiteType Bestellungen, die nicht konform sind, führen zu einem Fehler beim Abrufen nicht übereinstimmender Signaturen für den Canonical String vom API Gateway.
> Wenn der optionale Parameter enthalten ProductVersion ist, müssen Sie eine unterstützte Produktversion verwenden, wie unter Unterstützte Versionen von für V2 dokumentiert. AWS IoT Device Tester AWS IoT Greengrass

- Ersetzen Sie *`us-west-2 durch Ihre`*. AWS-Region Eine Liste der Regionscodes finden Sie unter Regionale Endpunkte.
- Ersetzen Sie *`Linux`* durch das Betriebssystem Ihres Host-Computers.
- Ersetzen Sie *`2.5.3`* durch Ihre Version von AWS IoT Greengrass Nucleus.

```
url=$(curl --request GET "https://
download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt?
HostOs=linux&ProductVersion=2.5.3&TestSuiteType=GGV2" \
--user $AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY \
--aws-sigv4 "aws:amz:us-west-2:iot-device-tester" \
| jq -r '.LatestBk["DownloadURL"]')

curl $url --output devicetester.zip
```

Example Beispiel: Herunterladen mit einer früheren Version von cURL (Mac und Linux)

Sie können den folgenden cURL-Befehl mit einer AWS Signatur verwenden, die Sie signieren und berechnen. Weitere Informationen zum Signieren und Berechnen einer AWS Signatur finden Sie unter AWS API-Anfragen signieren.

- Ersetzen Sie *`Linux`* durch das Betriebssystem Ihres Host-Computers.

- Ersetzen Sie *Timestamp* durch Datum und Uhrzeit, z. B. **20220210T004606Z**
- Ersetzen Sie *Datum* durch das Datum, z. B. **20220210**
- Ersetze es *AWSRegion*durch dein AWS-Region. Eine Liste der Regionscodes finden Sie unter Regionale Endpunkte.
- *AWSSignature*Ersetzen Sie es durch die AWS Signatur, die Sie generieren.

```
curl --location --request GET 'https://
download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt?
HostOs=linux&TestSuiteType=GGV2' \
--header 'X-Amz-Date: Timestamp \
--header 'Authorization: AWS4-HMAC-SHA256 Credential=$AWS_ACCESS_KEY_ID/Date/AWSRegion/
iot-device-tester/aws4_request, SignedHeaders=host;x-amz-date, Signature=AWSSignature'
```

Example Beispiel: Herunterladen mit einem Python-Skript

In diesem Beispiel wird die Python-Anforderungsbibliothek verwendet. Dieses Beispiel ist an das Python-Beispiel angepasst, um eine AWS API-Anfrage zu signieren in der AWS Allgemeinen Referenz.

- Ersetzen Sie *us-west-2* durch Ihre Region. Eine Liste der Regionscodes finden Sie unter Regionale Endpunkte.
- Ersetzen Sie *Linux* durch das Betriebssystem Ihres Host-Computers.

```
# Copyright 2010-2022 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#
# This file is licensed under the Apache License, Version 2.0 (the "License").
# You may not use this file except in compliance with the License. A copy of the
#License is located at
#
# http://aws.amazon.com/apache2.0/
#
# This file is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS
# OF ANY KIND, either express or implied. See the License for the specific
# language governing permissions and limitations under the License.

# See: http://docs.aws.amazon.com/general/latest/gr/sigv4_signing.html
# This version makes a GET request and passes the signature
# in the Authorization header.
```

```
import sys, os, base64, datetime, hashlib, hmac
import requests # pip install requests
# ************* REQUEST VALUES *************
method = 'GET'
service = 'iot-device-tester'
host = 'download.devicetester.iotdevicesecosystem.amazonaws.com'
region = 'us-west-2'
endpoint = 'https://download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt'
request_parameters = 'HostOs=linux&TestSuiteType=GGV2'

# Key derivation functions. See:
# http://docs.aws.amazon.com/general/latest/gr/signature-v4-examples.html#signature-v4-
examples-python
def sign(key, msg):
    return hmac.new(key, msg.encode('utf-8'), hashlib.sha256).digest()

def getSignatureKey(key, dateStamp, regionName, serviceName):
    kDate = sign(('AWS4' + key).encode('utf-8'), dateStamp)
    kRegion = sign(kDate, regionName)
    kService = sign(kRegion, serviceName)
    kSigning = sign(kService, 'aws4_request')
    return kSigning

# Read AWS access key from env. variables or configuration file. Best practice is NOT
# to embed credentials in code.
access_key = os.environ.get('AWS_ACCESS_KEY_ID')
secret_key = os.environ.get('AWS_SECRET_ACCESS_KEY')
if access_key is None or secret_key is None:
    print('No access key is available.')
    sys.exit()

# Create a date for headers and the credential string
t = datetime.datetime.utcnow()
amzdate = t.strftime('%Y%m%dT%H%M%SZ')
datestamp = t.strftime('%Y%m%d') # Date w/o time, used in credential scope

# ************* TASK 1: CREATE A CANONICAL REQUEST *************
# http://docs.aws.amazon.com/general/latest/gr/sigv4-create-canonical-request.html
# Step 1 is to define the verb (GET, POST, etc.)--already done.
# Step 2: Create canonical URI--the part of the URI from domain to query
# string (use '/' if no path)
canonical_uri = '/latestidt'
# Step 3: Create the canonical query string. In this example (a GET request),
# request parameters are in the query string. Query string values must
```

```
# be URL-encoded (space=%20). The parameters must be sorted by name.
# For this example, the query string is pre-formatted in the request_parameters
 variable.
canonical_querystring = request_parameters
# Step 4: Create the canonical headers and signed headers. Header names
# must be trimmed and lowercase, and sorted in code point order from
# low to high. Note that there is a trailing \n.
canonical_headers = 'host:' + host + '\n' + 'x-amz-date:' + amzdate + '\n'
# Step 5: Create the list of signed headers. This lists the headers
# in the canonical_headers list, delimited with ";" and in alpha order.
# Note: The request can include any headers; canonical_headers and
# signed_headers lists those that you want to be included in the
# hash of the request. "Host" and "x-amz-date" are always required.
signed_headers = 'host;x-amz-date'
# Step 6: Create payload hash (hash of the request body content). For GET
# requests, the payload is an empty string ("").
payload_hash = hashlib.sha256(('').encode('utf-8')).hexdigest()
# Step 7: Combine elements to create canonical request
canonical_request = method + '\n' + canonical_uri + '\n' + canonical_querystring + '\n'
 + canonical_headers + '\n' + signed_headers + '\n' + payload_hash

# ************* TASK 2: CREATE THE STRING TO SIGN*************
# Match the algorithm to the hashing algorithm you use, either SHA-1 or
# SHA-256 (recommended)
algorithm = 'AWS4-HMAC-SHA256'
credential_scope = datestamp + '/' + region + '/' + service + '/' + 'aws4_request'
string_to_sign = algorithm + '\n' +  amzdate + '\n' +  credential_scope + '\n' +
 hashlib.sha256(canonical_request.encode('utf-8')).hexdigest()
# ************* TASK 3: CALCULATE THE SIGNATURE *************
# Create the signing key using the function defined above.
signing_key = getSignatureKey(secret_key, datestamp, region, service)
# Sign the string_to_sign using the signing_key
signature = hmac.new(signing_key, (string_to_sign).encode('utf-8'),
 hashlib.sha256).hexdigest()

# ************* TASK 4: ADD SIGNING INFORMATION TO THE REQUEST *************
# The signing information can be either in a query string value or in
# a header named Authorization. This code shows how to use a header.
# Create authorization header and add to request headers
authorization_header = algorithm + ' ' + 'Credential=' + access_key + '/' +
 credential_scope + ', ' +  'SignedHeaders=' + signed_headers + ', ' + 'Signature=' +
 signature
# The request can include any headers, but MUST include "host", "x-amz-date",
# and (for this scenario) "Authorization". "host" and "x-amz-date" must
```

```
# be included in the canonical_headers and signed_headers, as noted
# earlier. Order here is not significant.
# Python note: The 'host' header is added automatically by the Python 'requests'
 library.
headers = {'x-amz-date':amzdate, 'Authorization':authorization_header}

# ************* SEND THE REQUEST *************
request_url = endpoint + '?' + canonical_querystring
print('\nBEGIN REQUEST++++++++++++++++++++++++++++++++++++')
print('Request URL = ' + request_url)
response = requests.get(request_url, headers=headers)
print('\nRESPONSE++++++++++++++++++++++++++++++++++++')
print('Response code: %d\n' % response.status_code)
print(response.text)

download_url = response.json()["LatestBk"]["DownloadURL"]
r = requests.get(download_url)
open('devicetester.zip', 'wb').write(r.content)
```

Verwenden Sie IDT, um die AWS IoT Greengrass Qualification Suite auszuführen

Sie können AWS IoT Device Tester für AWS IoT Greengrass V2 verwenden, 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 durch AWS IoT Core. Es überprüft beispielsweise, ob Ihr Gerät Komponenten bereitstellen und aktualisieren kann.

IDT for AWS IoT Greengrass V2 testet nicht nur Geräte, sondern erstellt auch Ressourcen (z. B. AWS IoT Dinge, Gruppen usw.) in Ihrem System, um den AWS-Konto Qualifizierungsprozess zu erleichtern.

Um diese Ressourcen zu erstellen, verwendet IDT for AWS IoT Greengrass V2 die in der `config.json` Datei konfigurierten AWS Anmeldeinformationen, um API-Aufrufe in Ihrem Namen durchzuführen. Diese Ressourcen werden zu verschiedenen Zeiten während eines Tests bereitgestellt.

Wenn Sie IDT for AWS IoT Greengrass V2 verwenden, um die AWS IoT Greengrass Qualification Suite auszuführen, führt sie 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 Board die für die Qualifikation erforderlichen Tests bestanden hat.

Test-Suite-Versionen

IDT for AWS IoT Greengrass V2 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 besteht aus einzelnen Tests, die sich auf eine bestimmte Funktion beziehen, z. B. auf die Bereitstellung von Komponenten.

Testsuiten werden beispielsweise anhand eines `major.minor.patch` Formats versioniert. `GGV2Q_1.0.0` Wenn Sie IDT herunterladen, enthält das Paket die neueste Version der Greengrass Qualification Suite.

> **Important**
>
> Tests von nicht unterstützten Testsuite-Versionen sind für die Gerätequalifizierung nicht gültig. IDT druckt keine Qualifizierungsberichte für nicht unterstützte Versionen. Weitere Informationen finden Sie unter the section called “Support-Richtlinie für AWS IoT Device Tester für AWS IoT Greengrass”.
> Sie können ausführen`list-supported-products`, um die Versionen von AWS IoT Greengrass und die Testsuiten aufzulisten, die von Ihrer aktuellen Version von IDT unterstützt werden.

Beschreibung der Testgruppen

Erforderliche Testgruppen für die Core-Qualifizierung

Diese Testgruppen sind erforderlich, um Ihr AWS IoT Greengrass V2-Gerät für den AWS Partner Gerätekatalog zu qualifizieren.

Kernabhängigkeiten

Überprüft, ob das Gerät alle Software- und Hardwareanforderungen für die AWS IoT Greengrass Core-Software erfüllt. Diese Testgruppe umfasst den folgenden Testfall:

Java-Version

Überprüft, ob die erforderliche Java-Version auf dem zu testenden Gerät installiert ist. AWS IoT Greengrass benötigt Java 8 oder höher.

PreTest Validierung

Überprüft, ob das Gerät die Softwareanforderungen für die Durchführung von Tests erfüllt.

- Bei Linux-basierten Geräten überprüft dieser Test, ob das Gerät die folgenden Linux-Befehle ausführen kann:

  `chmod`, `cp`, `echo`, `grep`, `kill`, `ln`, `mkinfo`, `ps`, `rm`, `sh`, `uname`

- Bei Windows-basierten Geräten überprüft dieser Test, ob auf dem Gerät die folgende Microsoft-Software installiert ist:

  Powershell v5.1 oder höher, .NET v4.6.1 oder höher, Visual C++ 2017 oder höher, Hilfsprogramm PsExec

Versionsprüfung

Überprüft, ob die AWS IoT Greengrass bereitgestellte Version mit der von Ihnen verwendeten AWS IoT Device Tester-Version kompatibel ist.

Komponente

Überprüft, ob das Gerät Komponenten bereitstellen und aktualisieren kann. Diese Testgruppe umfasst die folgenden Tests:

Cloud-Komponente

Überprüft die Gerätefähigkeit für Cloud-Komponenten.

Lokale Komponente

Überprüft die Gerätefähigkeit für lokale Komponenten.

Lambda

Dieser Test gilt nicht für Windows-basierte Geräte.

Überprüft, ob das Gerät Lambda-Funktionskomponenten bereitstellen kann, die die Java-Laufzeit verwenden, und ob die Lambda-Funktionen AWS IoT Core MQTT-Themen als Ereignisquellen für geschäftliche Nachrichten verwenden können.

MQTT

Überprüft, ob das Gerät MQTT-Themen abonnieren und veröffentlichen kann. AWS IoT Core

Optionale Testgruppen

> **Note**
>
> Diese Testgruppen sind optional und werden nur für die Qualifizierung von Linux-basierten Greengrass-Core-Geräten verwendet. Wenn Sie sich für optionale Tests qualifizieren, wird Ihr Gerät mit zusätzlichen Funktionen im AWS Partner Gerätekatalog aufgeführt.

Docker-Abhängigkeiten

Überprüft, ob das Gerät alle erforderlichen technischen Abhängigkeiten erfüllt, um die von AWS-bereitgestellte Komponente Docker Application Manager () zu verwenden. `aws.greengrass.DockerApplicationManager`

Qualifikation für den Docker-Anwendungsmanager

Überprüft, ob das Gerät ein Docker-Container-Image von Amazon ECR herunterladen kann.

Abhängigkeiten Machine Learning

> **Note**
>
> Die optionale Testgruppe für maschinelles Lernen wird nur in IDT v4.9.3 unterstützt.

Überprüft, ob das Gerät alle erforderlichen technischen Abhängigkeiten erfüllt, um die AWS bereitgestellten Komponenten für maschinelles Lernen (ML) verwenden zu können.

Inferenztests für Machine Learning

> **Note**
>
> Die optionale Testgruppe für maschinelles Lernen wird nur in IDT v4.9.3 unterstützt.

Überprüft, ob das Gerät mithilfe der Deep Learning Runtime - und Lite ML-Frameworks ML-Inferenz durchführen kann. TensorFlow

Stream Manager-Abhängigkeiten

> **Note**
>
> Die optionale Testgruppe des Stream-Managers wird nur in IDT v4.9.3 unterstützt.

Überprüft, ob das Gerät den Stream-Manager herunterladen, installieren und ausführen kann.AWS IoT Greengrass

Integration von Hardware-Sicherheit (Hardware Security Integration, HSI)

> **Note**
>
> Dieser Test ist in IDT v4.9.3 und höher nur für Linux-basierte Geräte verfügbar. AWS IoT Greengrass unterstützt derzeit keine Hardwaresicherheitsintegration für Windows-Geräte.

Überprüft, ob das Gerät Verbindungen zu den AWS IoT und AWS IoT Greengrass Diensten mithilfe eines privaten Schlüssels und Zertifikats authentifizieren kann, die in einem Hardware-Sicherheitsmodul (HSM) gespeichert sind. Mit diesem Test wird auch überprüft, ob die vom Anbieter AWS bereitgestellte PKCS #11 -Anbieterkomponente über eine vom Hersteller bereitgestellte PKCS #11 -Bibliothek eine Schnittstelle zum HSM herstellen kann. Weitere Informationen finden Sie unter Integration von Hardware-Sicherheit.

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) für AWS IoT Greengrass beschrieben.

Laden Sie die neueste Version von für AWS IoT Device Tester herunter AWS IoT Greengrass

Laden Sie die neueste Version von IDT herunter und extrahieren Sie die Software an einen Speicherort (*< device-tester-extract-location >*) auf Ihrem Dateisystem, für den Sie Lese-/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.

Laden Sie die Software herunter AWS IoT Greengrass

IDT for AWS IoT Greengrass V2 testet Ihr Gerät auf Kompatibilität mit einer bestimmten Version von AWS IoT Greengrass. Führen Sie den folgenden Befehl aus, um die AWS IoT Greengrass Core-Software in eine Datei mit dem Namen `aws.greengrass.nucleus.zip` herunterzuladen. Ersetzen Sie die *Version* durch eine unterstützte Nucleus-Komponentenversion für Ihre IDT-Version.

Linux or Unix

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip >
 aws.greengrass.nucleus.zip
```

Windows Command Prompt (CMD)

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip >
 aws.greengrass.nucleus.zip
```

PowerShell

```
iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip -
OutFile aws.greengrass.nucleus.zip
```

Platzieren Sie die heruntergeladene `aws.greengrass.nucleus.zip` Datei in dem *<device-tester-extract-location>*`/products/` Ordner.

> **Note**
>
> Legen Sie nicht mehrere Dateien in diesem Verzeichnis für das gleiche Betriebssystem und die gleiche Architektur ab.

Erstellen und konfigurieren Sie ein AWS-Konto

Bevor Sie AWS IoT Device Tester For AWS IoT Greengrass V2 verwenden können, müssen Sie die folgenden Schritte ausführen:

1. Richten Sie eine 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 und AWS IoT Greengrass Komponenten zu erstellen.

Um diese Ressourcen zu erstellen, verwendet IDT for AWS IoT Greengrass V2 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 AWS kostenlose Kontingent in Frage kommen, müssen Sie eine Kreditkarte angeben, wenn Sie sich für ein AWS-Konto kostenloses Kontingent anmelden. Weitere Informationen finden Sie unter Warum benötige ich eine Zahlungsmethode, wenn mein Konto vom kostenlosen Kontingent abgedeckt ist?.

Schritt 1: Richten Sie ein AWS-Konto

In diesem Schritt erstellen und konfigurieren Sie eine AWS-Konto. Wenn Sie bereits über ein AWS-Konto verfügen, fahren Sie direkt mit the section called “Schritt 2: Konfigurieren von Berechtigungen für IDT” fort.

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.

Wählen Sie zum Erstellen eines Administratorbenutzers eine der folgenden Optionen aus.

Wählen Sie eine Möglichkeit zur Verwaltung Ihres Administrators aus.	Bis	Von	Sie können auch
Im IAM Identity Center (Empfohlen)	Verwendung von kurzfristigen Anmeldeinformationen für den Zugriff auf AWS. Dies steht im Einklang mit den bewährten Methoden für die Sicherheit. Weitere Informationen zu	Beachtung der Anweisungen unter Erste Schritte im AWS IAM Identity Center - Benutzerhandbuch.	Konfigurieren Sie den programmatischen Zugriff, indem Sie den AWS IAM Identity Center im AWS Command Line Interface Benutzerhandbuch AWS CLI zu verwendenden konfigurieren.

Wählen Sie eine Möglichkeit zur Verwaltung Ihres Administrators aus.	Bis	Von	Sie können auch
	bewährten Methoden finden Sie unter Bewährte Methoden für die Sicherheit in IAM im IAM-Benutzerhandbuch.		
In IAM (Nicht empfohlen)	Verwendung von langfristigen Anmeldeinformationen für den Zugriff auf AWS.	Beachtung der Anweisungen unter Erstellen Ihres ersten IAM-Administratorbenutzers und Ihrer ersten Benutzergruppe im IAM-Benutzerhandbuch.	Programmgesteuerten Zugriff unter Verwendung der Informationen unter Verwalten der Zugriffsschlüssel für IAM-Benutzer im IAM-Benutzerhandbuch konfigurieren.

Schritt 2: Konfigurieren von Berechtigungen für IDT

In diesem Schritt konfigurieren Sie die Berechtigungen, die IDT für AWS IoT Greengrass V2 verwendet, um Tests durchzuführen und IDT-Nutzungsdaten zu sammeln. Sie können das AWS Management Consoleoder 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 fort. Konfigurieren Sie Ihr Gerät für die Ausführung von IDT-Tests

So konfigurieren Sie Berechtigungen für IDT (Konsole)

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. Wenn Sie den Platzhalterinhalt nicht verwenden PreInstalled, ersetzen Sie auf der Registerkarte JSON den Platzhalterinhalt durch die folgende Richtlinie. Wenn Sie verwenden PreInstalled, fahren Sie mit dem folgenden Schritt fort.

```
{
    "Version":"2012-10-17",
    "Statement":[
    {
      "Sid":"passRoleForResources",
      "Effect":"Allow",
      "Action":"iam:PassRole",
      "Resource":"arn:aws:iam::*:role/idt-*",
      "Condition":{
        "StringEquals":{
          "iam:PassedToService":[
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
          ]
        }
      }
    },
    {
      "Sid":"lambdaResources",
      "Effect":"Allow",
      "Action":[
        "lambda:CreateFunction",
        "lambda:PublishVersion",
        "lambda:DeleteFunction",
        "lambda:GetFunction"
      ],
      "Resource":[
        "arn:aws:lambda:*:*:function:idt-*"
      ]
    },
    {
      "Sid":"iotResources",
```

```
      "Effect":"Allow",
      "Action":[
        "iot:CreateThing",
        "iot:DeleteThing",
        "iot:DescribeThing",
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
        "iot:AddThingToThingGroup",
        "iot:RemoveThingFromThingGroup",
        "iot:AttachThingPrincipal",
        "iot:DetachThingPrincipal",
        "iot:UpdateCertificate",
        "iot:DeleteCertificate",
        "iot:CreatePolicy",
        "iot:AttachPolicy",
        "iot:DetachPolicy",
        "iot:DeletePolicy",
        "iot:GetPolicy",
        "iot:Publish",
        "iot:TagResource",
        "iot:ListThingPrincipals",
        "iot:ListAttachedPolicies",
        "iot:ListTargetsForPolicy",
        "iot:ListThingGroupsForThing",
        "iot:ListThingsInThingGroup",
        "iot:CreateJob",
        "iot:DescribeJob",
        "iot:DescribeJobExecution",
        "iot:CancelJob"
      ],
      "Resource":[
        "arn:aws:iot:*:*:thing/idt-*",
        "arn:aws:iot:*:*:thinggroup/idt-*",
        "arn:aws:iot:*:*:policy/idt-*",
        "arn:aws:iot:*:*:cert/*",
        "arn:aws:iot:*:*:topic/idt-*",
        "arn:aws:iot:*:*:job/*"
      ]
    },
    {
      "Sid":"s3Resources",
      "Effect":"Allow",
      "Action":[
```

```
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObjectVersion",
        "s3:DeleteObject",
        "s3:CreateBucket",
        "s3:ListBucket",
        "s3:ListBucketVersions",
        "s3:DeleteBucket",
        "s3:PutObjectTagging",
        "s3:PutBucketTagging"
      ],
      "Resource":"arn:aws:s3::*:idt-*"
    },
    {
      "Sid":"roleAliasResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateRoleAlias",
        "iot:DescribeRoleAlias",
        "iot:DeleteRoleAlias",
        "iot:TagResource",
        "iam:GetRole"
      ],
      "Resource":[
        "arn:aws:iot:*:*:rolealias/idt-*",
        "arn:aws:iam::*:role/idt-*"
      ]
    },
    {
      "Sid":"idtExecuteAndCollectMetrics",
      "Effect":"Allow",
      "Action":[
        "iot-device-tester:SendMetrics",
        "iot-device-tester:SupportedVersion",
        "iot-device-tester:LatestIdt",
        "iot-device-tester:CheckVersion",
        "iot-device-tester:DownloadTestSuite"
      ],
      "Resource":"*"
    },
    {
      "Sid":"genericResources",
      "Effect":"Allow",
      "Action":[
```

```
        "greengrass:*",
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:ListThings",
        "iot:DescribeEndpoint",
        "iot:CreateKeysAndCertificate"
      ],
      "Resource":"*"
    },
    {
      "Sid":"iamResourcesUpdate",
      "Effect":"Allow",
      "Action":[
        "iam:CreateRole",
        "iam:DeleteRole",
        "iam:CreatePolicy",
        "iam:DeletePolicy",
        "iam:AttachRolePolicy",
        "iam:DetachRolePolicy",
        "iam:TagRole",
        "iam:TagPolicy",
        "iam:GetPolicy",
        "iam:ListAttachedRolePolicies",
        "iam:ListEntitiesForPolicy"
      ],
      "Resource":[
        "arn:aws:iam::*:role/idt-*",
        "arn:aws:iam::*:policy/idt-*"
      ]
    }
  ]
}
```

c. Wenn Sie verwenden PreInstalled, ersetzen Sie auf der Registerkarte JSON den Platzhalterinhalt durch die folgende Richtlinie. Stellen Sie sicher, dass Sie:

- Ersetzen Sie *`thingName`* und *`ThingGroup`* in der `iotResources` Anweisung durch den Dingnamen und die Dinggruppe, die während der Greengrass-Installation auf Ihrem zu testenden Gerät (DUT) erstellt wurden, um Berechtigungen hinzuzufügen.
- Ersetzen Sie *`PassRole`* und *`roleAlias`* in der `roleAliasResources` Anweisung und der Anweisung durch die `passRoleForResources` Rollen, die während der Greengrass-Installation auf Ihrem DUT erstellt wurden.

```
{
    "Version":"2012-10-17",
    "Statement":[
    {
      "Sid":"passRoleForResources",
      "Effect":"Allow",
      "Action":"iam:PassRole",
      "Resource":"arn:aws:iam::*:role/passRole",
      "Condition":{
        "StringEquals":{
          "iam:PassedToService":[
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
          ]
        }
      }
    },
    {
      "Sid":"lambdaResources",
      "Effect":"Allow",
      "Action":[
        "lambda:CreateFunction",
        "lambda:PublishVersion",
        "lambda:DeleteFunction",
        "lambda:GetFunction"
      ],
      "Resource":[
        "arn:aws:lambda:*:*:function:idt-*"
      ]
    },
    {
      "Sid":"iotResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateThing",
        "iot:DeleteThing",
        "iot:DescribeThing",
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
        "iot:AddThingToThingGroup",
```

```
        "iot:RemoveThingFromThingGroup",
        "iot:AttachThingPrincipal",
        "iot:DetachThingPrincipal",
        "iot:UpdateCertificate",
        "iot:DeleteCertificate",
        "iot:CreatePolicy",
        "iot:AttachPolicy",
        "iot:DetachPolicy",
        "iot:DeletePolicy",
        "iot:GetPolicy",
        "iot:Publish",
        "iot:TagResource",
        "iot:ListThingPrincipals",
        "iot:ListAttachedPolicies",
        "iot:ListTargetsForPolicy",
        "iot:ListThingGroupsForThing",
        "iot:ListThingsInThingGroup",
        "iot:CreateJob",
        "iot:DescribeJob",
        "iot:DescribeJobExecution",
        "iot:CancelJob"
      ],
      "Resource":[
        "arn:aws:iot:*:*:thing/thingName",
        "arn:aws:iot:*:*:thinggroup/thingGroup",
        "arn:aws:iot:*:*:policy/idt-*",
        "arn:aws:iot:*:*:cert/*",
        "arn:aws:iot:*:*:topic/idt-*",
        "arn:aws:iot:*:*:job/*"
      ]
    },
    {
      "Sid":"s3Resources",
      "Effect":"Allow",
      "Action":[
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObjectVersion",
        "s3:DeleteObject",
        "s3:CreateBucket",
        "s3:ListBucket",
        "s3:ListBucketVersions",
        "s3:DeleteBucket",
        "s3:PutObjectTagging",
```

```
        "s3:PutBucketTagging"
      ],
      "Resource":"arn:aws:s3::*:idt-*"
    },
    {
      "Sid":"roleAliasResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateRoleAlias",
        "iot:DescribeRoleAlias",
        "iot:DeleteRoleAlias",
        "iot:TagResource",
        "iam:GetRole"
      ],
      "Resource":[
        "arn:aws:iot:*:*:rolealias/roleAlias",
        "arn:aws:iam::*:role/idt-*"
      ]
    },
    {
      "Sid":"idtExecuteAndCollectMetrics",
      "Effect":"Allow",
      "Action":[
        "iot-device-tester:SendMetrics",
        "iot-device-tester:SupportedVersion",
        "iot-device-tester:LatestIdt",
        "iot-device-tester:CheckVersion",
        "iot-device-tester:DownloadTestSuite"
      ],
      "Resource":"*"
    },
    {
      "Sid":"genericResources",
      "Effect":"Allow",
      "Action":[
        "greengrass:*",
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:ListThings",
        "iot:DescribeEndpoint",
        "iot:CreateKeysAndCertificate"
      ],
      "Resource":"*"
    },
```

```
    {
      "Sid":"iamResourcesUpdate",
      "Effect":"Allow",
      "Action":[
        "iam:CreateRole",
        "iam:DeleteRole",
        "iam:CreatePolicy",
        "iam:DeletePolicy",
        "iam:AttachRolePolicy",
        "iam:DetachRolePolicy",
        "iam:TagRole",
        "iam:TagPolicy",
        "iam:GetPolicy",
        "iam:ListAttachedRolePolicies",
        "iam:ListEntitiesForPolicy"
      ],
      "Resource":[
        "arn:aws:iam::*:role/idt-*",
        "arn:aws:iam::*:policy/idt-*"
      ]
    }
  ]
}
```

> **Note**
>
> Wenn Sie eine benutzerdefinierte IAM-Rolle als Token-Austauschrolle für Ihr zu testendes Gerät verwenden möchten, stellen Sie sicher, dass Sie die `roleAliasResources` Anweisung und die `passRoleForResources` Anweisung in Ihrer Richtlinie aktualisieren, um Ihre benutzerdefinierte IAM-Rollenressource zuzulassen.

d. Wählen Sie Richtlinie 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 Set permissions (Berechtigungen einrichten) die Option Attach existing policies to user directly (Vorhandene Richtlinien dem Benutzer direkt anfügen) aus.

   ii. Suchen Sie nach der IDTGreenGrassiamPermissions-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.

4. Nächster Schritt: Konfigurieren Sie Ihr physisches Gerät.

So konfigurieren Sie Berechtigungen für IDT (AWS CLI)

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.

a. Wenn Sie es nicht verwenden PreInstalled, öffnen Sie einen Texteditor und speichern Sie den folgenden Richtlinieninhalt in einer JSON-Datei. Wenn Sie dies verwenden PreInstalled, fahren Sie mit dem folgenden Schritt fort.

```
{
    "Version":"2012-10-17",
    "Statement":[
    {
      "Sid":"passRoleForResources",
      "Effect":"Allow",
      "Action":"iam:PassRole",
      "Resource":"arn:aws:iam::*:role/idt-*",
      "Condition":{
        "StringEquals":{
          "iam:PassedToService":[
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
          ]
        }
      }
    },
    {
      "Sid":"lambdaResources",
      "Effect":"Allow",
      "Action":[
        "lambda:CreateFunction",
        "lambda:PublishVersion",
        "lambda:DeleteFunction",
        "lambda:GetFunction"
      ],
      "Resource":[
        "arn:aws:lambda:*:*:function:idt-*"
      ]
    },
    {
      "Sid":"iotResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateThing",
        "iot:DeleteThing",
        "iot:DescribeThing",
```

```
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
        "iot:AddThingToThingGroup",
        "iot:RemoveThingFromThingGroup",
        "iot:AttachThingPrincipal",
        "iot:DetachThingPrincipal",
        "iot:UpdateCertificate",
        "iot:DeleteCertificate",
        "iot:CreatePolicy",
        "iot:AttachPolicy",
        "iot:DetachPolicy",
        "iot:DeletePolicy",
        "iot:GetPolicy",
        "iot:Publish",
        "iot:TagResource",
        "iot:ListThingPrincipals",
        "iot:ListAttachedPolicies",
        "iot:ListTargetsForPolicy",
        "iot:ListThingGroupsForThing",
        "iot:ListThingsInThingGroup",
        "iot:CreateJob",
        "iot:DescribeJob",
        "iot:DescribeJobExecution",
        "iot:CancelJob"
      ],
      "Resource":[
        "arn:aws:iot:*:*:thing/idt-*",
        "arn:aws:iot:*:*:thinggroup/idt-*",
        "arn:aws:iot:*:*:policy/idt-*",
        "arn:aws:iot:*:*:cert/*",
        "arn:aws:iot:*:*:topic/idt-*",
        "arn:aws:iot:*:*:job/*"
      ]
    },
    {
      "Sid":"s3Resources",
      "Effect":"Allow",
      "Action":[
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObjectVersion",
        "s3:DeleteObject",
        "s3:CreateBucket",
```

```
        "s3:ListBucket",
        "s3:ListBucketVersions",
        "s3:DeleteBucket",
        "s3:PutObjectTagging",
        "s3:PutBucketTagging"
      ],
      "Resource":"arn:aws:s3::*:idt-*"
    },
    {
      "Sid":"roleAliasResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateRoleAlias",
        "iot:DescribeRoleAlias",
        "iot:DeleteRoleAlias",
        "iot:TagResource",
        "iam:GetRole"
      ],
      "Resource":[
        "arn:aws:iot:*:*:rolealias/idt-*",
        "arn:aws:iam::*:role/idt-*"
      ]
    },
    {
      "Sid":"idtExecuteAndCollectMetrics",
      "Effect":"Allow",
      "Action":[
        "iot-device-tester:SendMetrics",
        "iot-device-tester:SupportedVersion",
        "iot-device-tester:LatestIdt",
        "iot-device-tester:CheckVersion",
        "iot-device-tester:DownloadTestSuite"
      ],
      "Resource":"*"
    },
    {
      "Sid":"genericResources",
      "Effect":"Allow",
      "Action":[
        "greengrass:*",
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:ListThings",
        "iot:DescribeEndpoint",
```

```
        "iot:CreateKeysAndCertificate"
      ],
      "Resource":"*"
    },
    {
      "Sid":"iamResourcesUpdate",
      "Effect":"Allow",
      "Action":[
        "iam:CreateRole",
        "iam:DeleteRole",
        "iam:CreatePolicy",
        "iam:DeletePolicy",
        "iam:AttachRolePolicy",
        "iam:DetachRolePolicy",
        "iam:TagRole",
        "iam:TagPolicy",
        "iam:GetPolicy",
        "iam:ListAttachedRolePolicies",
        "iam:ListEntitiesForPolicy"
      ],
      "Resource":[
        "arn:aws:iam::*:role/idt-*",
        "arn:aws:iam::*:policy/idt-*"
      ]
    }
  ]
}
```

b. Wenn Sie verwenden PreInstalled, öffnen Sie einen Texteditor und speichern Sie den folgenden Richtlinieninhalt in einer JSON-Datei. Stellen Sie sicher, dass Sie:

- Ersetzen Sie *`thingName`* und *`ThingGroup`* in der `iotResources` Anweisung, die während der Greengrass-Installation auf Ihrem zu testenden Gerät (DUT) erstellt wurden, um Berechtigungen hinzuzufügen.
- Ersetzen Sie *`PassRole`* und *`roleAlias`* in der `roleAliasResources` Anweisung und der Anweisung durch die `passRoleForResources` Rollen, die während der Greengrass-Installation auf Ihrem DUT erstellt wurden.

```
{
    "Version":"2012-10-17",
    "Statement":[
```

```
    {
      "Sid":"passRoleForResources",
      "Effect":"Allow",
      "Action":"iam:PassRole",
      "Resource":"arn:aws:iam::*:role/passRole",
      "Condition":{
        "StringEquals":{
          "iam:PassedToService":[
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
          ]
        }
      }
    },
    {
      "Sid":"lambdaResources",
      "Effect":"Allow",
      "Action":[
        "lambda:CreateFunction",
        "lambda:PublishVersion",
        "lambda:DeleteFunction",
        "lambda:GetFunction"
      ],
      "Resource":[
        "arn:aws:lambda:*:*:function:idt-*"
      ]
    },
    {
      "Sid":"iotResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateThing",
        "iot:DeleteThing",
        "iot:DescribeThing",
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
        "iot:AddThingToThingGroup",
        "iot:RemoveThingFromThingGroup",
        "iot:AttachThingPrincipal",
        "iot:DetachThingPrincipal",
        "iot:UpdateCertificate",
        "iot:DeleteCertificate",
```

```
        "iot:CreatePolicy",
        "iot:AttachPolicy",
        "iot:DetachPolicy",
        "iot:DeletePolicy",
        "iot:GetPolicy",
        "iot:Publish",
        "iot:TagResource",
        "iot:ListThingPrincipals",
        "iot:ListAttachedPolicies",
        "iot:ListTargetsForPolicy",
        "iot:ListThingGroupsForThing",
        "iot:ListThingsInThingGroup",
        "iot:CreateJob",
        "iot:DescribeJob",
        "iot:DescribeJobExecution",
        "iot:CancelJob"
      ],
      "Resource":[
        "arn:aws:iot:*:*:thing/thingName",
        "arn:aws:iot:*:*:thinggroup/thingGroup",
        "arn:aws:iot:*:*:policy/idt-*",
        "arn:aws:iot:*:*:cert/*",
        "arn:aws:iot:*:*:topic/idt-*",
        "arn:aws:iot:*:*:job/*"
      ]
    },
    {
      "Sid":"s3Resources",
      "Effect":"Allow",
      "Action":[
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObjectVersion",
        "s3:DeleteObject",
        "s3:CreateBucket",
        "s3:ListBucket",
        "s3:ListBucketVersions",
        "s3:DeleteBucket",
        "s3:PutObjectTagging",
        "s3:PutBucketTagging"
      ],
      "Resource":"arn:aws:s3::*:idt-*"
    },
    {
```

```
      "Sid":"roleAliasResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateRoleAlias",
        "iot:DescribeRoleAlias",
        "iot:DeleteRoleAlias",
        "iot:TagResource",
        "iam:GetRole"
      ],
      "Resource":[
        "arn:aws:iot:*:*:rolealias/roleAlias",
        "arn:aws:iam::*:role/idt-*"
      ]
    },
    {
      "Sid":"idtExecuteAndCollectMetrics",
      "Effect":"Allow",
      "Action":[
        "iot-device-tester:SendMetrics",
        "iot-device-tester:SupportedVersion",
        "iot-device-tester:LatestIdt",
        "iot-device-tester:CheckVersion",
        "iot-device-tester:DownloadTestSuite"
      ],
      "Resource":"*"
    },
    {
      "Sid":"genericResources",
      "Effect":"Allow",
      "Action":[
        "greengrass:*",
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:ListThings",
        "iot:DescribeEndpoint",
        "iot:CreateKeysAndCertificate"
      ],
      "Resource":"*"
    },
    {
      "Sid":"iamResourcesUpdate",
      "Effect":"Allow",
      "Action":[
        "iam:CreateRole",
```

```
        "iam:DeleteRole",
        "iam:CreatePolicy",
        "iam:DeletePolicy",
        "iam:AttachRolePolicy",
        "iam:DetachRolePolicy",
        "iam:TagRole",
        "iam:TagPolicy",
        "iam:GetPolicy",
        "iam:ListAttachedRolePolicies",
        "iam:ListEntitiesForPolicy"
      ],
      "Resource":[
        "arn:aws:iam::*:role/idt-*",
        "arn:aws:iam::*:policy/idt-*"
      ]
    }
  ]
}
```

> **Note**
>
> Wenn Sie eine benutzerdefinierte IAM-Rolle als Token-Austauschrolle für Ihr zu testendes Gerät verwenden möchten, stellen Sie sicher, dass Sie die `roleAliasResources` Anweisung und die `passRoleForResources` Anweisung in Ihrer Richtlinie aktualisieren, um Ihre benutzerdefinierte IAM-Rollenressource zuzulassen.

c. Führen Sie den folgenden Befehl aus, um eine vom Kunden verwaltete Richtlinie mit dem Namen zu erstellen. `IDTGreengrassIAMPermissions` *`policy.json`*Ersetzen Sie durch den vollständigen Pfad zur JSON-Datei, die Sie im vorherigen Schritt erstellt haben.

```
aws iam create-policy --policy-name IDTGreengrassIAMPermissions --policy-
document file://policy.json
```

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
```

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 IoT Device Tester Berechtigungen

In den folgenden Richtlinien werden AWS IoT Device Tester Berechtigungen beschrieben.

AWS IoT Device Tester benötigt diese Berechtigungen für Funktionen zur Versionsprüfung und automatischen Aktualisierung.

- `iot-device-tester:SupportedVersion`

  Erteilt die AWS IoT Device Tester Erlaubnis, die Liste der unterstützten Produkte, Testsuiten und IDT-Versionen abzurufen.
- `iot-device-tester:LatestIdt`

  AWS IoT Device Tester Erteilt die Erlaubnis, die neueste IDT-Version abzurufen, die zum Herunterladen verfügbar ist.
- `iot-device-tester:CheckVersion`

  AWS IoT Device Tester Erteilt die Erlaubnis, die Versionskompatibilität für IDT, Testsuiten und Produkte zu überprüfen.
- `iot-device-tester:DownloadTestSuite`

  AWS IoT Device Tester Erteilt die Erlaubnis zum Herunterladen von Updates für Testsuiten.

AWS IoT Device Tester verwendet außerdem die folgende Berechtigung für optionale Metrikberichte:

- `iot-device-tester:SendMetrics`

  Erteilt die Erlaubnis AWS , Metriken zur AWS IoT Device Tester internen Nutzung zu sammeln. Wenn diese Erlaubnis nicht erteilt wird, werden diese Messwerte nicht erfasst.

Konfigurieren Sie Ihr Gerät für die Ausführung von IDT-Tests

Damit IDT Tests zur Gerätequalifizierung durchführen kann, müssen Sie Ihren Host-Computer für den Zugriff auf Ihr Gerät konfigurieren und Benutzerberechtigungen auf Ihrem Gerät konfigurieren.

Installieren Sie Java auf dem Host-Computer

Ab IDT v4.2.0 AWS IoT Greengrass erfordern die optionalen Qualifikationstests für die Ausführung von Java.

Sie können Java Version 8 oder höher verwenden. Wir empfehlen, dass Sie die Langzeit-Support-Versionen von Amazon Corretto oder OpenJDK verwenden. Version 8 oder höher ist erforderlich..

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. Beispielsweise:

   $ ssh-copy-id *<remote-ssh-user>*@*<remote-device-ip>*

   Wo *remote-ssh-user* wird der Benutzername verwendet, um sich bei Ihrem zu testenden Gerät anzumelden, und *remote-device-ip* ist die IP-Adresse des zu testenden Geräts, mit dem Tests durchgeführt werden sollen. Beispielsweise:

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 Sie Benutzeranmeldeinformationen für Windows-Geräte

Um ein Windows-basiertes Gerät zu qualifizieren, müssen Sie Benutzeranmeldeinformationen im LocalSystem Konto auf dem zu testenden Gerät für die folgenden Benutzer konfigurieren:

- Der Standard-Greengrass-Benutzer (`ggc_user`).
- Der Benutzer, mit dem Sie eine Verbindung zu dem zu testenden Gerät herstellen. Sie konfigurieren diesen Benutzer in der `device.json`Datei.

Sie müssen jeden Benutzer im LocalSystem Konto auf dem zu testenden Gerät erstellen und dann den Benutzernamen und das Passwort für den Benutzer in der Credential Manager-Instanz für das LocalSystem Konto speichern.

Um Benutzer auf Windows-Geräten zu konfigurieren

1. Öffnen Sie die Windows-Eingabeaufforderung (`cmd.exe`) als Administrator.
2. Erstellen Sie die Benutzer im LocalSystem Konto auf dem Windows-Gerät. Führen Sie den folgenden Befehl für jeden Benutzer aus, den Sie erstellen möchten. Ersetzen Sie für den Standardbenutzer von Greengrass den *Benutzernamen durch*. `ggc_user` Ersetzen Sie *das Passwort* durch ein sicheres Passwort.

   ```
   net user /add user-name password
   ```

3. Laden Sie das PsExecProgramm von Microsoft herunter und installieren Sie es auf dem Gerät.
4. Verwenden Sie das PsExec Hilfsprogramm, um den Benutzernamen und das Passwort für den Standardbenutzer in der Credential Manager-Instanz für das LocalSystem Konto zu speichern.

Führen Sie den folgenden Befehl für jeden Benutzer aus, den Sie in Credential Manager konfigurieren möchten. Ersetzen Sie für den Standardbenutzer von Greengrass den *Benutzernamen durch*. `ggc_user` Ersetzen Sie *das Passwort* durch das Passwort des Benutzers, das Sie zuvor festgelegt haben.

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

Wenn das PsExec License Agreementgeöffnet wird, stimmen Sie Acceptder Lizenz zu und führen Sie den Befehl aus.

> **Note**
>
> Auf Windows-Geräten wird auf dem LocalSystem Konto der Greengrass-Nucleus ausgeführt, und Sie müssen das PsExec Hilfsprogramm verwenden, um Benutzerinformationen im LocalSystem Konto zu speichern. Wenn Sie die Credential Manager-Anwendung verwenden, werden diese Informationen nicht im Konto, sondern im Windows-Konto des aktuell angemeldeten Benutzers gespeichert. LocalSystem

Konfigurieren von Benutzerberechtigungen auf Ihrem Gerät

IDT führt Operationen für verschiedene Verzeichnisse und Dateien auf einem zu testenden Gerät aus. Einige dieser Operationen erfordern höhere Berechtigungen (mit sudo). Um diese Vorgänge zu automatisieren, muss IDT für AWS IoT Greengrass V2 in der Lage sein, Befehle mit sudo auszuführen, ohne nach einem Passwort gefragt zu werden.

Führen Sie die folgenden Schritte auf dem zu testenden Gerät aus, um sudo den Zugriff ohne Aufforderung zur Eingabe eines Passworts zu erlauben.

> **Note**
>
> `username` bezieht sich auf den SSH-Benutzer, der von IDT für den Zugriff auf das zu testende Gerät verwendet wird.

So fügen Sie den Benutzer der sudo-Gruppe hinzu

1. Führen Sie auf dem zu testenden Gerät `sudo usermod -aG sudo` *<username>* aus.

2. Melden Sie sich ab und melden Sie sich dann wieder an, damit die Änderungen wirksam werden.
3. Führen Sie sudo echo test aus, um zu überprüfen, ob der Benutzername erfolgreich hinzugefügt wurde. Wenn Sie nicht zur Eingabe eines Passworts aufgefordert werden, ist Ihr Benutzer korrekt konfiguriert.
4. Öffnen Sie die Datei `/etc/sudoers` und fügen Sie am Ende der Datei die folgende Zeile hinzu:

   ```
   <ssh-username> ALL=(ALL) NOPASSWD: ALL
   ```

Konfigurieren Sie eine benutzerdefinierte Token-Austauschrolle

Sie können eine benutzerdefinierte IAM-Rolle als Token-Austauschrolle verwenden, die das getestete Gerät für die Interaktion mit AWS Ressourcen annimmt. Informationen zum Erstellen einer IAM-Rolle finden Sie unter Erstellen von IAM-Rollen im IAM-Benutzerhandbuch.

Sie müssen die folgenden Anforderungen erfüllen, damit IDT Ihre benutzerdefinierte IAM-Rolle verwenden kann. Wir empfehlen dringend, dieser Rolle nur die minimal erforderlichen Richtlinienaktionen hinzuzufügen.

- Die Konfigurationsdatei userdata.json muss aktualisiert werden, um den `GreengrassV2TokenExchangeRole` Parameter auf zu setzen. `true`
- Die benutzerdefinierte IAM-Rolle muss mit der folgenden Mindestvertrauensrichtlinie konfiguriert werden:

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":[
               "credentials.iot.amazonaws.com",
               "lambda.amazonaws.com",
               "sagemaker.amazonaws.com"
            ]
         },
         "Action":"sts:AssumeRole"
      }
   ]
}
```

- Die benutzerdefinierte IAM-Rolle muss mit der folgenden Mindestberechtigungsrichtlinie konfiguriert werden:

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "iot:DescribeCertificate",
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:PutLogEvents",
            "logs:DescribeLogStreams",
            "iot:Connect",
            "iot:Publish",
            "iot:Subscribe",
            "iot:Receive",
            "iot:ListThingPrincipals",
            "iot:GetThingShadow",
            "iot:UpdateThingShadow",
            "s3:GetBucketLocation",
            "s3:GetObject",
            "s3:PutObject",
            "s3:AbortMultipartUpload",
            "s3:ListMultipartUploadParts"
         ],
         "Resource":"*"
      }
   ]
}
```

- Der Name der benutzerdefinierten IAM-Rolle muss mit der IAM-Rollenressource übereinstimmen, die Sie in den IAM-Berechtigungen für den Testbenutzer angeben. Standardmäßig ermöglicht die Testbenutzerrichtlinie den Zugriff auf IAM-Rollen, deren Rollennamen das `idt-` Präfix enthalten. Wenn Ihr IAM-Rollenname dieses Präfix nicht verwendet, fügen Sie die `arn:aws:iam::*:role/`*`custom-iam-role-name`* Ressource der `roleAliasResources` Anweisung und der `passRoleForResources` Anweisung in Ihrer Testbenutzerrichtlinie hinzu, wie in den folgenden Beispielen gezeigt:

Example **`passRoleForResources`**-Anweisung

```
{
   "Sid":"passRoleForResources",
   "Effect":"Allow",
   "Action":"iam:PassRole",
   "Resource":"arn:aws:iam::*:role/custom-iam-role-name",
   "Condition":{
      "StringEquals":{
         "iam:PassedToService":[
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
         ]
      }
   }
}
```

Example **`roleAliasResources`**-Anweisung

```
{
   "Sid":"roleAliasResources",
   "Effect":"Allow",
   "Action":[
      "iot:CreateRoleAlias",
      "iot:DescribeRoleAlias",
      "iot:DeleteRoleAlias",
      "iot:TagResource",
      "iam:GetRole"
   ],
   "Resource":[
      "arn:aws:iot:*:*:rolealias/idt-*",
      "arn:aws:iam::*:role/custom-iam-role-name"
   ]
}
```

Konfigurieren Ihres Geräts zum Testen optionaler Funktionen

In diesem Abschnitt werden die Geräteanforderungen für die Ausführung von IDT-Tests für optionale Docker- und ML-Funktionen (Machine Learning) beschrieben. Die ML-Funktionen werden nur in IDT

v4.9.3 unterstützt. Sie müssen sicherstellen, dass Ihr Gerät diese Anforderungen nur erfüllt, wenn Sie diese Funktionen testen möchten. Fahren Sie andernfalls mit dem Schritt the section called “IDT-Einstellungen konfigurieren” fort.

Themen

- Anforderungen an die Docker-Qualifikation
- Anforderungen an die ML-Qualifikation
- HSM-Qualifikationsanforderungen

Anforderungen an die Docker-Qualifikation

IDT für AWS IoT Greengrass V2 bietet Docker-Qualifizierungstests, um zu überprüfen, ob Ihre Geräte die von Ihnen AWS bereitgestellte Docker-Anwendungsmanager-Komponente zum Herunterladen von Docker-Container-Images verwenden können, die Sie mit benutzerdefinierten Docker-Container-Komponenten ausführen können. Informationen zum Erstellen benutzerdefinierter Docker-Komponenten finden Sie unter. Führen Sie einen Docker-Container aus

Um Docker-Qualifizierungstests ausführen zu können, müssen Ihre getesteten Geräte die folgenden Anforderungen erfüllen, um die Docker Application Manager-Komponente bereitstellen zu können.

- Docker Engine 1.9.1 oder höher ist auf dem Greengrass-Core-Gerät installiert. Version 20.10 ist die neueste Version, für die verifiziert wurde, dass sie mit der Core-Software funktioniert. AWS IoT Greengrass Sie müssen Docker direkt auf dem Kerngerät installieren, bevor Sie Komponenten bereitstellen, auf denen Docker-Container ausgeführt werden.
- Der Docker-Daemon wurde auf dem Kerngerät gestartet und ausgeführt, bevor Sie diese Komponente bereitstellen.
- Der Systembenutzer, der eine Docker-Container-Komponente ausführt, muss über Root- oder Administratorrechte verfügen, oder Sie müssen Docker so konfigurieren, dass es als Benutzer ohne Root- oder Administratorrechte ausgeführt wird.
  - Auf Linux-Geräten können Sie der Gruppe einen Benutzer hinzufügen, um Befehle ohne Befehle `docker` aufzurufen. `docker` `sudo`
  - Auf Windows-Geräten können Sie der `docker-users` Gruppe einen Benutzer hinzufügen, um `docker` Befehle ohne Administratorrechte aufzurufen.

Linux or Unix

Führen Sie den folgenden Befehl aus`ggc_user`, um der `docker` Gruppe einen Nicht-Root-Benutzer, den Sie zum Ausführen von Docker-Container-Komponenten verwenden, hinzuzufügen.

```
sudo usermod -aG docker ggc_user
```

Weitere Informationen finden Sie unter Docker als Nicht-Root-Benutzer verwalten.

Windows Command Prompt (CMD)

Um der `docker-users` Gruppe den Benutzer`ggc_user`, den Sie zum Ausführen von Docker-Container-Komponenten verwenden, hinzuzufügen, führen Sie den folgenden Befehl als Administrator aus.

```
net localgroup docker-users ggc_user /add
```

Windows PowerShell

Um der `docker-users` Gruppe den Benutzer`ggc_user`, den Sie zum Ausführen von Docker-Container-Komponenten verwenden, hinzuzufügen, führen Sie den folgenden Befehl als Administrator aus.

```
Add-LocalGroupMember -Group docker-users -Member ggc_user
```

Anforderungen an die ML-Qualifikation

> **Note**
>
> Die Funktion für maschinelles Lernen wird nur in IDT v4.9.3 unterstützt.

IDT for AWS IoT Greengrass V2 bietet ML-Qualifizierungstests, um zu überprüfen, ob Ihre Geräte die AWS bereitgestellten Komponenten für maschinelles Lernen verwenden können, um ML-Inferenzen lokal mithilfe der Deep Learning Runtime- oder Lite ML-Frameworks durchzuführen. TensorFlow
Weitere Informationen zur Ausführung von ML-Inferenz auf Greengrass-Geräten finden Sie unter. Durchführen von Machine Learning-Inferenzen

Um ML-Qualifizierungstests durchführen zu können, müssen Ihre getesteten Geräte die folgenden Anforderungen erfüllen, um die Komponenten für maschinelles Lernen bereitstellen zu können.

- Auf Greengrass-Core-Geräten, auf denen Amazon Linux 2 oder Ubuntu 18.04 ausgeführt wird, ist die GNU C Library (Glibc) Version 2.27 oder höher auf dem Gerät installiert.
- Auf ARMv7L-Geräten wie Raspberry Pi sind Abhängigkeiten für OpenCV-Python auf dem Gerät installiert. Führen Sie den folgenden Befehl aus, um die Abhängigkeiten zu installieren.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- Raspberry Pi-Geräte, auf denen Raspberry Pi OS Bullseye ausgeführt wird, müssen die folgenden Anforderungen erfüllen:
  - NumPy 1.22.4 oder höher auf dem Gerät installiert. Raspberry Pi OS Bullseye enthält eine frühere Version von NumPy, sodass Sie den folgenden Befehl ausführen können, um das Gerät zu aktualisieren NumPy .

    ```
    pip3 install --upgrade numpy
    ```

  - Der ältere Kamerastack ist auf dem Gerät aktiviert. Raspberry Pi OS Bullseye enthält einen neuen Kamerastack, der standardmäßig aktiviert und nicht kompatibel ist. Sie müssen also den älteren Kamerastack aktivieren.

    Um den Legacy-Kamerastack zu aktivieren

    1. Führen Sie den folgenden Befehl aus, um das Raspberry Pi-Konfigurationstool zu öffnen.

       ```
       sudo raspi-config
       ```

    2. Wählen Sie Schnittstellenoptionen.
    3. Wählen Sie Legacy-Kamera aus, um den Legacy-Kamerastack zu aktivieren.
    4. Starten Sie den Raspberry Pi neu.

HSM-Qualifikationsanforderungen

AWS IoT Greengrass stellt die PKCS #11 -Anbieterkomponente zur Integration in das PKCS Hardware Security Module (HSM) auf dem Gerät bereit. Das HSM-Setup hängt von Ihrem Gerät und dem HSM-Modul ab, das Sie ausgewählt haben. Solange die erwartete HSM-Konfiguration,

wie in den IDT-Konfigurationseinstellungen dokumentiert, bereitgestellt wird, verfügt IDT über die Informationen, die für die Durchführung dieses optionalen Feature-Qualifizierungstests erforderlich sind.

Konfigurieren Sie die IDT-Einstellungen, um die AWS IoT Greengrass Qualification Suite auszuführen

Bevor Sie Tests ausführen, müssen Sie Einstellungen für AWS Anmeldeinformationen und Geräte auf Ihrem Hostcomputer konfigurieren.

Konfigurieren Sie die AWS Anmeldeinformationen in config.json

Sie müssen Ihre IAM-Benutzeranmeldedaten in der *<device_tester_extract_location>*/`configs/config.json` Datei konfigurieren. Verwenden Sie die Anmeldeinformationen für den IDT for AWS IoT Greengrass V2-Benutzer, der in erstellt wurde. the section called “Erstellen und konfigurieren Sie ein AWS-Konto” Sie können Ihre Anmeldeinformationen auf zwei Arten angeben:

- In einer Datei mit Anmeldeinformationen
- Als Umgebungsvariablen

Konfigurieren Sie AWS Anmeldeinformationen mit einer Anmeldeinformationsdatei

IDT verwendet die gleiche Anmeldeinformationsdatei wie das AWS CLI. Weitere Informationen finden Sie unter Konfigurations- und Anmeldeinformationsdateien.

Der Speicherort der Datei mit den Anmeldeinformationen variiert je nach verwendetem Betriebssystem:

- macOS Linux: `~/.aws/credentials`
- Windows: `C:\Users\`*`UserName`*`\.aws\credentials`

Fügen Sie der `credentials` Datei Ihre AWS Anmeldeinformationen im folgenden Format hinzu:

```
[default]
aws_access_key_id = <your_access_key_id>
aws_secret_access_key = <your_secret_access_key>
```

Um IDT für AWS IoT Greengrass V2 so zu konfigurieren, dass AWS Anmeldeinformationen aus Ihrer `credentials` Datei verwendet werden, bearbeiten Sie Ihre `config.json` Datei wie folgt:

```
{
  "awsRegion": "region",
  "auth": {
    "method": "file",
    "credentials": {
      "profile": "default"
    }
  }
}
```

> **Note**
>
> Wenn Sie das `default` AWS Profil nicht verwenden, ändern Sie unbedingt den Profilnamen in Ihrer `config.json` Datei. Weitere Informationen hierzu finden Sie unter Benannte Profile.

Konfigurieren Sie AWS Anmeldeinformationen mit Umgebungsvariablen

Umgebungsvariablen sind Variablen, die vom Betriebssystem gepflegt und von Systembefehlen verwendet werden. Sie werden nicht gespeichert, wenn Sie die SSH-Sitzung schließen. IDT für AWS IoT Greengrass V2 kann die `AWS_SECRET_ACCESS_KEY` Umgebungsvariablen `AWS_ACCESS_KEY_ID` und zum Speichern Ihrer AWS Anmeldeinformationen verwenden.

Um diese Variablen auf Linux, macOS oder Unix festzulegen, verwenden Sie export:

```
export AWS_ACCESS_KEY_ID=<your_access_key_id>
export AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

In Windows können Sie die Variablen mit set festlegen:

```
set AWS_ACCESS_KEY_ID=<your_access_key_id>
set AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

Um den IDT so zu konfigurieren, dass er die Umgebungsvariablen verwendet, bearbeiten Sie den Abschnitt `auth` in Ihrer Datei `config.json`. Ein Beispiel:

```
{
  "awsRegion": "region",
  "auth": {
```

```
    "method": "environment"
  }
}
```

Konfigurieren von device.json

> **Note**
>
> IDT v4.9.3 unterstützt das Testen der Funktionen`ml`, und`docker`. `streamManagement` IDT v4.9.4 und spätere Versionen unterstützen Tests. `docker` Wenn Sie diese Funktionen nicht testen möchten, setzen Sie den entsprechenden Wert auf. `no`

Zusätzlich zu den AWS Anmeldeinformationen benötigt IDT für AWS IoT Greengrass V2 Informationen über die Geräte, auf denen die Tests ausgeführt werden. Beispielinformationen wären 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:

IDT v4.9.3

```
[
  {
    "id": "<pool-id>",
    "sku": "<sku>",
    "features": [
      {
        "name": "arch",
        "value": "x86_64 | armv6l | armv7l | aarch64"
      },
      {
        "name": "ml",
        "value": "dlr | tensorflowlite | dlr,tensorflowlite | no"
      },
      {
        "name": "docker",
        "value": "yes | no"
      },
      {
        "name": "streamManagement",
```

```
        "value": "yes | no"
      },
      {
        "name": "hsi",
        "value": "hsm | no"
      }
    ],
    "devices": [
      {
        "id": "<device-id>",
        "operatingSystem": "Linux | Windows",
        "connectivity": {
          "protocol": "ssh",
          "ip": "<ip-address>",
          "port": 22,
          "publicKeyPath": "<public-key-path>",
          "auth": {
            "method": "pki | password",
            "credentials": {
              "user": "<user-name>",
              "privKeyPath": "/path/to/private/key",
              "password": "<password>"
            }
          }
        }
      }
    ]
  }
]
```

> **Note**
>
> Geben Sie `privKeyPath` nur an, wenn `method` auf `pki` gesetzt ist.
> Geben Sie `password` nur an, wenn `method` auf `password` gesetzt ist.

Alle Eigenschaften, die Werte enthalten, sind erforderlich, wie hier 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 Gerät im AWS Partner Gerätekatalog anbieten möchten, muss die hier angegebene SKU mit der SKU übereinstimmen, die Sie bei der Angebotserstellung verwenden.

`features`

Ein Array, das die Funktionen enthält, die das Gerät unterstützt. Alle Funktionen sind erforderlich.

`arch`

Die unterstützten Betriebssystemarchitekturen, die durch den Testlauf validiert werden. Gültige Werte für sind:

- `x86_64`
- `armv6l`
- `armv7l`
- `aarch64`

`ml`

Überprüft, ob das Gerät alle erforderlichen technischen Abhängigkeiten erfüllt, um die AWS bereitgestellten Komponenten für maschinelles Lernen (ML) verwenden zu können.

Durch die Aktivierung dieser Funktion wird auch überprüft, ob das Gerät ML-Inferenz mithilfe der Frameworks Deep Learning Runtime und TensorFlow Lite ML durchführen kann.

Gültige Werte sind eine beliebige Kombination aus `dlr` und `tensorflowlite` oder. no

`docker`

Überprüft, ob das Gerät alle erforderlichen technischen Abhängigkeiten erfüllt, um die von AWS-bereitgestellte Docker-Anwendungsmanager () `aws.greengrass.DockerApplicationManager` -Komponente verwenden zu können.

Durch die Aktivierung dieser Funktion wird auch bestätigt, dass das Gerät ein Docker-Container-Image von Amazon ECR herunterladen kann.

Gültige Werte sind eine beliebige Kombination von oder. `yes` `no`

`streamManagement`

Überprüft, ob das Gerät den AWS IoT Greengrass Stream-Manager herunterladen, installieren und ausführen kann.

Gültige Werte sind eine beliebige Kombination von `yes` oderno.

`hsi`

Überprüft, ob das Gerät Verbindungen zu den AWS IoT und AWS IoT Greengrass Diensten mithilfe eines privaten Schlüssels und Zertifikats authentifizieren kann, die in einem Hardware-Sicherheitsmodul (HSM) gespeichert sind. Mit diesem Test wird auch überprüft, ob die vom Hersteller AWS bereitgestellte PKCS #11 -Anbieterkomponente über eine vom Hersteller bereitgestellte PKCS #11 -Bibliothek eine Schnittstelle zum HSM herstellen kann. Weitere Informationen finden Sie unter Integration von Hardware-Sicherheit.

Gültige Werte sind `hsm` oder no.

> **Note**
>
> Das Testen von `hsi` ist nur mit IDT v4.9.3 und späteren Versionen verfügbar.

`devices.id`

Eine benutzerdefinierte eindeutige Kennung für das zu testende Gerät.

`devices.operatingSystem`

Das Betriebssystem des Geräts. Unterstützte Werte sind `Linux` und `Windows`.

`connectivity.protocol`

Das Kommunikationsprotokoll, das für die Kommunikation mit diesem Gerät verwendet wird. Derzeit gilt der einzige unterstützte Wert `ssh` für physische Geräte.

`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.publicKeyPath`

Optional. Der vollständige Pfad zum öffentlichen Schlüssel, der zur Authentifizierung von Verbindungen mit dem zu testenden Gerät verwendet wird.

Wenn Sie das angeben`publicKeyPath`, validiert IDT den öffentlichen Schlüssel des Geräts, wenn es eine SSH-Verbindung zu dem zu testenden Gerät herstellt. Wenn dieser Wert nicht angegeben ist, stellt IDT eine SSH-Verbindung her, validiert aber nicht den öffentlichen Schlüssel des Geräts.

Wir empfehlen dringend, dass Sie den Pfad zum öffentlichen Schlüssel angeben und eine sichere Methode verwenden, um diesen öffentlichen Schlüssel abzurufen. Für standardmäßige SSH-Clients, die auf der Befehlszeile basieren, wird der öffentliche Schlüssel in der Datei bereitgestellt. `known_hosts` Wenn Sie eine separate öffentliche Schlüsseldatei angeben, muss diese Datei dasselbe Format wie die `known_hosts` Datei verwenden, d. h.. *`ip-address key-type public-key`* Wenn es mehrere Einträge mit derselben IP-Adresse gibt, muss der Eintrag für den von IDT verwendeten Schlüsseltyp vor den anderen Einträgen in der Datei stehen.

`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.

IDT v4.9.4

```
[
  {
    "id": "<pool-id>",
    "sku": "<sku>",
    "features": [
      {
        "name": "arch",
        "value": "x86_64 | armv6l | armv7l | aarch64"
      },
      {
        "name": "docker",
```

```
        "value": "yes | no"
      },
      {
        "name": "hsi",
        "value": "hsm | no"
      }
    ],
    "devices": [
      {
        "id": "<device-id>",
        "operatingSystem": "Linux | Windows",
        "connectivity": {
          "protocol": "ssh",
          "ip": "<ip-address>",
          "port": 22,
          "publicKeyPath": "<public-key-path>",
          "auth": {
            "method": "pki | password",
            "credentials": {
              "user": "<user-name>",
              "privKeyPath": "/path/to/private/key",
              "password": "<password>"
            }
          }
        }
      }
    ]
  }
]
```

> **Note**
>
> Geben Sie `privKeyPath` nur an, wenn `method` auf `pki` gesetzt ist.
> Geben Sie `password` nur an, wenn `method` auf `password` gesetzt ist.

Alle Eigenschaften, die Werte enthalten, sind erforderlich, wie hier 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 Gerät im AWS Partner Gerätekatalog anbieten möchten, muss die hier angegebene SKU mit der SKU übereinstimmen, die Sie bei der Angebotserstellung verwenden.

`features`

Ein Array, das die Funktionen enthält, die das Gerät unterstützt. Alle Funktionen sind erforderlich.

`arch`

Die unterstützten Betriebssystemarchitekturen, die durch den Testlauf validiert werden. Gültige Werte für sind:

- `x86_64`
- `armv6l`
- `armv7l`
- `aarch64`

`docker`

Überprüft, ob das Gerät alle erforderlichen technischen Abhängigkeiten erfüllt, um die von ihm AWS bereitgestellte Komponente Docker Application Manager () zu verwenden. `aws.greengrass.DockerApplicationManager`

Durch die Aktivierung dieser Funktion wird auch bestätigt, dass das Gerät ein Docker-Container-Image von Amazon ECR herunterladen kann.

Gültige Werte sind eine beliebige Kombination von oder. `yes` `no`

`hsi`

Überprüft, ob das Gerät Verbindungen zu den AWS IoT und AWS IoT Greengrass Diensten mithilfe eines privaten Schlüssels und Zertifikats authentifizieren kann, die in einem Hardware-Sicherheitsmodul (HSM) gespeichert sind. Mit diesem Test wird auch überprüft, ob die vom Hersteller AWS bereitgestellte PKCS #11 -Anbieterkomponente über eine vom Hersteller bereitgestellte PKCS #11 -Bibliothek eine Schnittstelle zum HSM herstellen kann. Weitere Informationen finden Sie unter Integration von Hardware-Sicherheit.

Gültige Werte sind `hsm` oder `no`.

> **Note**
>
> Das Testen von `hsi` ist nur mit IDT v4.9.3 und späteren Versionen verfügbar.

`devices.id`

Eine benutzerdefinierte eindeutige Kennung für das zu testende Gerät.

`devices.operatingSystem`

Das Betriebssystem des Geräts. Unterstützte Werte sind `Linux` und `Windows`.

`connectivity.protocol`

Das Kommunikationsprotokoll, das für die Kommunikation mit diesem Gerät verwendet wird. Derzeit gilt der einzige unterstützte Wert `ssh` für physische Geräte.

`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.publicKeyPath`

Optional. Der vollständige Pfad zum öffentlichen Schlüssel, der zur Authentifizierung von Verbindungen mit dem zu testenden Gerät verwendet wird.

Wenn Sie das angeben`publicKeyPath`, validiert IDT den öffentlichen Schlüssel des Geräts, wenn es eine SSH-Verbindung zu dem zu testenden Gerät herstellt. Wenn dieser Wert nicht angegeben ist, stellt IDT eine SSH-Verbindung her, validiert aber nicht den öffentlichen Schlüssel des Geräts.

Wir empfehlen dringend, dass Sie den Pfad zum öffentlichen Schlüssel angeben und eine sichere Methode verwenden, um diesen öffentlichen Schlüssel abzurufen. Für standardmäßige SSH-Clients, die auf der Befehlszeile basieren, wird der öffentliche Schlüssel in der Datei bereitgestellt. `known_hosts` Wenn Sie eine separate öffentliche Schlüsseldatei angeben, muss diese Datei dasselbe Format wie die `known_hosts` Datei verwenden, d. h.. *`ip-address key-type public-key`* Wenn es mehrere Einträge mit derselben IP-Adresse gibt, muss der Eintrag für den von IDT verwendeten Schlüsseltyp vor den anderen Einträgen in der Datei stehen.

`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.

Konfigurieren Sie userdata.json

IDT für AWS IoT Greengrass V2 benötigt außerdem zusätzliche Informationen zum Speicherort von Testartefakten und Software. AWS IoT Greengrass

Sie müssen diese Informationen mittels der Vorlage `userdata.json` in *<device_tester_extract_location>*`/configs/userdata.json` angeben:

```
{
    "TempResourcesDirOnDevice": "/path/to/temp/folder",
    "InstallationDirRootOnDevice": "/path/to/installation/folder",
    "GreengrassNucleusZip": "/path/to/aws.greengrass.nucleus.zip",
    "PreInstalled": "yes/no",
    "GreengrassV2TokenExchangeRole": "custom-iam-role-name",
 "hsm": {
        "greengrassPkcsPluginJar": "/path/to/aws.greengrass.crypto.Pkcs11Provider-
latest.jar",
        "pkcs11ProviderLibrary": "/path/to/pkcs11-vendor-library",
        "slotId": "slot-id",
        "slotLabel": "slot-label",
        "slotUserPin": "slot-pin",
        "keyLabel": "key-label",
        "preloadedCertificateArn": "certificate-arn"
        "rootCA": "path/to/root-ca"
    }
}
```

Alle Eigenschaften, die Werte enthalten, sind wie hier beschrieben erforderlich:

`TempResourcesDirOnDevice`

Der vollständige Pfad zu einem temporären Ordner auf dem zu testenden Gerät, in dem Testartefakte gespeichert werden sollen. Stellen Sie sicher, dass keine Sudo-Berechtigungen erforderlich sind, um in dieses Verzeichnis zu schreiben.

> **Note**
>
> IDT löscht den Inhalt dieses Ordners, wenn die Ausführung eines Tests abgeschlossen ist.

`InstallationDirRootOnDevice`

Der vollständige Pfad zu einem Ordner auf dem Gerät, in dem die Installation durchgeführt werden soll. AWS IoT Greengrass Für PreInstalled Greengrass ist dies der Pfad zum Greengrass-Installationsverzeichnis.

Sie müssen die erforderlichen Dateiberechtigungen für diesen Ordner festlegen. Führen Sie den folgenden Befehl für jeden Ordner im Installationspfad aus.

```
sudo chmod 755 folder-name
```

`GreengrassNucleusZip`

Der vollständige Pfad zur Greengrass Nucleus-ZIP-Datei (`greengrass-nucleus-latest.zip`) auf Ihrem Host-Computer. Dieses Feld ist für Tests mit PreInstalled Greengrass nicht erforderlich.

> **Note**
>
> Informationen zu den unterstützten Versionen von Greengrass Nucleus for IDT für AWS IoT Greengrass finden Sie unter. Aktuelle IDT-Version für V2 AWS IoT Greengrass Informationen zum Herunterladen der neuesten Greengrass-Software finden Sie unter AWS IoT Greengrass Software herunterladen.

`PreInstalled`

Diese Funktion ist nur für IDT v4.5.8 und spätere Versionen auf Linux-Geräten verfügbar.

(Optional) Wenn der Wert *ja* ist, geht IDT davon aus, dass der angegebene Pfad das Verzeichnis ist`InstallationDirRootOnDevice`, in dem Greengrass installiert ist.

Weitere Informationen zur Installation von Greengrass auf Ihrem Gerät finden Sie unterInstallieren von AWS IoT Greengrass Core-Software mit automatischer Ressourcenbereitstellung. Wenn Sie die Installation mit manueller Bereitstellung durchführen, schließen Sie den Schritt „Das AWS IoT Ding zu einer neuen oder vorhandenen Dinggruppe hinzufügen“ ein, wenn Sie ein AWS IoT Ding manuell erstellen. IDT geht davon aus, dass das Ding und die Dinggruppe während der Installation erstellt wurden. Stellen Sie sicher, dass diese Werte in der `effectiveConfig.yaml` Datei wiedergegeben werden. IDT sucht nach der Datei `effectiveConfig.yaml` unter`<InstallationDirRootOnDevice>/config/effectiveConfig.yaml`.

Stellen Sie beim Ausführen von Tests mit HSM sicher, dass das `aws.greengrass.crypto.Pkcs11Provider` Feld in aktualisiert ist. `effectiveConfig.yaml`

`GreengrassV2TokenExchangeRole`

(Optional) Die benutzerdefinierte IAM-Rolle, die Sie als Token-Austauschrolle verwenden möchten, von der das zu testende Gerät annimmt, um mit AWS Ressourcen zu interagieren.

> **Note**
>
> IDT verwendet diese benutzerdefinierte IAM-Rolle, anstatt während des Testlaufs die standardmäßige Token-Austauschrolle zu erstellen. Wenn Sie eine benutzerdefinierte Rolle verwenden, können Sie die IAM-Berechtigungen für den Testbenutzer aktualisieren, um die `iamResourcesUpdate` Anweisung auszuschließen, die es dem Benutzer ermöglicht, IAM-Rollen und -Richtlinien zu erstellen und zu löschen.

Weitere Informationen zum Erstellen einer benutzerdefinierten IAM-Rolle als Token-Exchange-Rolle finden Sie unter. Konfigurieren Sie eine benutzerdefinierte Token-Austauschrolle

`hsm`

Diese Funktion ist für IDT v4.5.1 und höher verfügbar.

(Optional) Die Konfigurationsinformationen für Tests mit einem AWS IoT Greengrass Hardware Security Module (HSM). Andernfalls sollte die `hsm`-Eigenschaft weggelassen werden. Weitere Informationen finden Sie unter Integration von Hardware-Sicherheit.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

> **Warning**
>
> Die HSM-Konfiguration kann als sensible Daten betrachtet werden, wenn das Hardware-Sicherheitsmodul von IDT und einem anderen System gemeinsam genutzt wird. In diesem Fall können Sie die Sicherung dieser Konfigurationswerte im Klartext vermeiden, indem Sie sie in einem AWS Parameter SecureString Store-Parameter speichern und IDT so konfigurieren, dass sie während der Testausführung abgerufen werden. Weitere Informationen finden Sie unter ???.

`hsm.greengrassPkcsPluginJar`

Der vollständige Pfad zur PKCS #11 -Anbieterkomponente, die Sie auf den IDT-Hostcomputer herunterladen. AWS IoT Greengrass stellt diese Komponente als JAR-Datei bereit, die Sie herunterladen können, um sie während der Installation als Provisioning-Plugin anzugeben. Sie können die neueste Version der JAR-Datei der Komponente unter der folgenden URL herunterladen: https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/aws.greengrass.crypto.Pkcs11Provider-latest.jar.

`hsm.pkcs11ProviderLibrary`

Der vollständige Pfad zur PKCS #11 -Bibliothek, die vom Hersteller des Hardware-Sicherheitsmoduls (HSM) für die Interaktion mit dem HSM bereitgestellt wird.

`hsm.slotId`

Die Steckplatz-ID, anhand derer der HSM-Steckplatz identifiziert wird, in den Sie den Schlüssel und das Zertifikat laden.

`hsm.slotLabel`

Die Steckplatzbezeichnung, anhand derer der HSM-Steckplatz identifiziert wird, in den Sie den Schlüssel und das Zertifikat laden.

`hsm.slotUserPin`

Die Benutzer-PIN, die IDT verwendet, um die AWS IoT Greengrass Core-Software gegenüber dem HSM zu authentifizieren.

> **Note**
>
> Verwenden Sie aus Sicherheitsgründen nicht dieselbe Benutzer-PIN auf Produktionsgeräten.

`hsm.keyLabel`

Das Label zur Identifizierung des Schlüssels im Hardwaremodul. Sowohl der Schlüssel als auch das Zertifikat müssen dieselbe Schlüsselbezeichnung verwenden.

`hsm.preloadedCertificateArn`

Der Amazon-Ressourcenname (ARN) des hochgeladenen Gerätezertifikats in der AWS IoT Cloud.

Sie müssen dieses Zertifikat zuvor mit dem Schlüssel im HSM generiert, in Ihr HSM importiert und in die AWS IoT Cloud hochgeladen haben. Informationen zum Generieren und Importieren des Zertifikats finden Sie in der Dokumentation zu Ihrem HSM.

Sie müssen das Zertifikat in dasselbe Konto und dieselbe Region hochladen, die Sie in config.json angeben. . Weitere Informationen zum Hochladen Ihres Zertifikats auf AWS IoT finden Sie unter Manuelles Registrieren eines Client-Zertifikats im AWS IoT Entwicklerhandbuch.

`hsm.rootCAPath`

(Optional) Der vollständige Pfad auf dem IDT-Hostcomputer zur Stammzertifizierungsstelle (CA), die Ihr Zertifikat signiert hat. Dies ist erforderlich, wenn das Zertifikat in Ihrem erstellten HSM nicht von der Amazon-Stammzertifizierungsstelle signiert ist.

Rufen Sie die Konfiguration aus dem Parameter Store ab AWS

AWS IoT Device Tester (IDT) enthält eine optionale Funktion zum Abrufen von Konfigurationswerten aus dem AWS Systems Manager Parameter Store. AWS Der Parameter Store ermöglicht die sichere und verschlüsselte Speicherung von Konfigurationen. Nach der Konfiguration kann IDT Parameter aus dem AWS Parameterspeicher abrufen, anstatt Parameter im Klartext in der Datei zu speichern. `userdata.json` Dies ist nützlich für alle sensiblen Daten, die verschlüsselt gespeichert werden sollten, z. B.: Passwörter, Pins und andere geheime Daten.

1. Um diese Funktion nutzen zu können, müssen Sie die bei der Erstellung Ihres IDT-Benutzers verwendeten Berechtigungen aktualisieren, um die GetParameter Aktion mit den Parametern zu ermöglichen, für deren Verwendung IDT konfiguriert ist. Im Folgenden finden Sie ein Beispiel für eine Berechtigungserklärung, die dem IDT-Benutzer hinzugefügt werden kann. Weitere Informationen finden Sie im AWS Systems Manager Benutzerhandbuch.

   ```
   {
          "Sid":"parameterStoreResources",
          "Effect": "Allow",
           "Action": [
               "ssm:GetParameter"
           ],
           "Resource": "arn:aws:ssm:*:*:parameter/IDT*"
   }
   ```

   Die obige Berechtigung ist so konfiguriert, dass alle Parameter abgerufen werden können, deren Name mit, beginntIDT, unter Verwendung des Platzhalterzeichens. * Sie sollten dies an Ihre Bedürfnisse anpassen, damit IDT Zugriff darauf hat, alle konfigurierten Parameter auf der Grundlage der Benennung der von Ihnen verwendeten Parameter abzurufen.

2. Sie müssen Ihre Konfigurationswerte im AWS Paramater Store speichern. Dies kann über die AWS Konsole oder über die AWS CLI erfolgen. AWS Mit Parameter Store können Sie zwischen verschlüsseltem und unverschlüsseltem Speicher wählen. Für die Speicherung sensibler Werte wie Geheimnisse, Passwörter und Pins sollten Sie die verschlüsselte Option verwenden, bei der es sich um einen Parametertyp von SecureString handelt. Um einen Parameter mit der AWS CLI hochzuladen, können Sie den folgenden Befehl verwenden:

   ```
   aws ssm put-parameter --name IDT-example-name --value IDT-example-value --type
    SecureString
   ```

   Mit dem folgenden Befehl können Sie überprüfen, ob ein Parameter gespeichert ist. (Optional) Verwenden Sie das `--with-decryption` Flag, um einen entschlüsselten SecureString Parameter abzurufen.

   ```
   aws ssm get-parameter --name IDT-example-name
   ```

   Bei Verwendung der AWS CLI wird der Parameter in die AWS Region des aktuellen CLI-Benutzers hochgeladen, und IDT ruft Parameter aus der Region ab, in der konfiguriert ist. `config.json` Verwenden Sie Folgendes, um Ihre Region AWS über die CLI zu überprüfen:

```
aws configure get region
```

3. Sobald Sie einen Konfigurationswert in der AWS Cloud haben, können Sie jeden Wert in der IDT-Konfiguration aktualisieren, um ihn aus der AWS Cloud abzurufen. Dazu verwenden Sie einen Platzhalter in Ihrer IDT-Konfiguration des Formulars, `{{AWS.Parameter.parameter_name}}` um den Parameter mit diesem Namen aus dem Parameterspeicher abzurufen. AWS

   Nehmen wir beispielsweise an, Sie möchten den `IDT-example-name` Parameter aus Schritt 2 als HSM-KeyLabel in Ihrer HSM-Konfiguration verwenden. Um dies zu tun, können Sie Ihre `userdata.json` wie folgt aktualisieren:

   ```
   "hsm": {
           "keyLabel": "{{AWS.Parameter.IDT-example-name}}",
           [...]
       }
   ```

   IDT ruft zur Laufzeit den Wert dieses Parameters ab, auf den `IDT-example-value` in Schritt 2 gesetzt wurde. Diese Konfiguration ähnelt der Einstellung, `"keyLabel": "IDT-example-value"` aber stattdessen wird dieser Wert verschlüsselt in der AWS Cloud gespeichert.

Ausführen der AWS IoT Greengrass Qualifizierungssuite

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.

Verwenden Sie den folgenden `run-suite` Befehl, um eine Reihe von Tests auszuführen.

```
devicetester_[linux | mac | win]_x86-64 run-suite  \\
    --suite-id suite-id  \\
    --group-id group-id  \\
    --pool-id your-device-pool \\
    --test-id test-id  \\
    --update-idt y|n  \\
    --userdata userdata.json
```

Alle Optionen sind optional. Sie können beispielsweise weglassen, `pool-id` wenn Sie nur einen Gerätepool haben, bei dem es sich um einen Satz identischer Geräte handelt, die in Ihrer `-device.json`Datei definiert sind. 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 “Test-Suite-Versionen”.

Beispielbefehle zum Ausführen der Qualifizierungssuite

Die folgenden Befehlszeilenbeispiele zeigen Ihnen, wie Sie die Qualifizierungstests für einen Gerätepool ausführen. Weitere Informationen zu `run-suite` und anderen IDT-Befehlen finden Sie unter the section called “IDT-Befehle”.

Verwenden Sie den folgenden Befehl, um alle Testgruppen in einer angegebenen Testsuite auszuführen. Der `list-suites` Befehl listet die Testsuiten auf, die sich im `tests` Ordner befinden.

```
devicetester_[linux | mac | win]_x86-64 run-suite \
    --suite-id GGV2Q_1.0.0 \
    --pool-id <pool-id> \
    --userdata userdata.json
```

Verwenden Sie den folgenden Befehl, um eine bestimmte Testgruppe in einer Testsuite auszuführen. Der `list-groups` Befehl listet die Testgruppen in einer Testsuite auf.

```
devicetester_[linux | mac | win]_x86-64 run-suite \
    --suite-id GGV2Q_1.0.0 \
    --group-id <group-id> \
    --pool-id <pool-id> \
    --userdata userdata.json
```

Verwenden Sie den folgenden Befehl, um einen bestimmten Testfall in einer Testgruppe auszuführen.

```
devicetester_[linux | mac | win]_x86-64 run-suite \
    --group-id <group-id> \
    --test-id <test-id> \
```

```
    --userdata userdata.json
```

Verwenden Sie den folgenden Befehl, um mehrere Testfälle in einer Testgruppe auszuführen.

```
devicetester_[linux | mac | win]_x86-64 run-suite \
    --group-id <group-id> \
    --test-id <test-id1>,<test-id2>
    --userdata userdata.json
```

Verwenden Sie den folgenden Befehl, um alle Testfälle in einer Testgruppe aufzulisten.

```
devicetester_[linux | mac | win]_x86-64 list-test-cases --group-id <group-id>
```

Wir empfehlen Ihnen, die vollständige Qualifikationstestsuite auszuführen, die Testgruppenabhängigkeiten in der richtigen Reihenfolge ausführt. Wenn Sie sich für die Ausführung bestimmter Testgruppen entscheiden, empfehlen wir Ihnen, zuerst die Testgruppe für die Abhängigkeitsprüfung auszuführen, um sicherzustellen, dass alle Greengrass-Abhängigkeiten installiert sind, bevor Sie zugehörige Testgruppen ausführen. Beispielsweise:

- Führen Sie vor Ausführung von Testgruppen für die Core-Qualifizierung `coredependencies` aus.

IDT für AWS IoT Greengrass V2-Befehle

Die IDT-Befehle befinden sich im *<device-tester-extract-location>*`/bin`-Verzeichnis. Um eine Testsuite auszuführen, geben Sie den Befehl im folgenden Format an:

`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`. Die Testgruppe, nach der gesucht werden soll. Diese Option ist erforderlich und muss eine einzelne Gruppe angeben.

`run-suite`

Führt eine Reihe von Tests in einem Pool von Geräten aus. Im Folgenden finden Sie einige unterstützte Optionen:

- `suite-id`. Die auszuführende Testsuite-Version. Wenn nicht angegeben, verwendet IDT die neueste Version im `tests`-Ordner.
- `group-id`. Die auszuführenden Testgruppen als kommagetrennte Liste. Wenn nicht angegeben, führt IDT alle entsprechenden Testgruppen in der Testsuite aus, abhängig von den konfigurierten Einstellungen in `device.json`. IDT führt keine Testgruppen aus, die das Gerät basierend auf Ihren konfigurierten Einstellungen nicht unterstützt, selbst wenn diese Testgruppen in der `group-id` Liste angegeben sind.
- `test-id`. Die auszuführenden Testfälle als kommagetrennte Liste. Wenn angegeben, muss `group-id` eine einzelne Gruppe angeben.
- `pool-id`. Der zu testende Gerätepool. Sie müssen einen Pool angeben, wenn mehrere Gerätepools in der `device.json`-Datei definiert sind.
- `stop-on-first-failure`. Konfiguriert IDT so, dass es beim ersten Fehler nicht mehr ausgeführt wird. Verwenden Sie diese Option mit , `group-id` wenn Sie die angegebenen Testgruppen debuggen möchten. Verwenden Sie diese Option nicht, wenn Sie eine vollständige Testsuite ausführen, um einen Qualifizierungsbericht zu generieren.
- `update-idt`. Legt die Antwort für die Aufforderung zum Aktualisieren von IDT fest. Die `Y` Antwort stoppt die Testausführung, wenn IDT feststellt, dass es eine neuere Version gibt. Die `N` Antwort setzt die Testausführung fort.
- `userdata`. Der vollständige Pfad zu der `userdata.json` Datei, die Informationen zu Testartefaktpfaden enthält. Diese Option ist für den `run-suite` Befehl erforderlich. Die `userdata.json` Datei muss sich im Verzeichnis *`devicetester_extract_location`* / devicetester_ggv2_*`[win|mac|linux]`*/configs/ befinden.

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.

Informationen zur Behebung von Fehlern finden Sie unterFehlerbehebung IDT fürAWS IoT GreengrassV2.

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>`/. In beiden Berichten werden die Ergebnisse der Ausführung der Qualifizierungstestsuite erfasst.

Dies`awsiotdevicetester_report.xml` ist der Qualifizierungstestbericht, den Sie einreichenAWS, um Ihr Gerät imAWS Partner Gerätekatalog aufzulisten. 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 auf der Grundlage der Gerätefunktionen wie lokaler Ressourcenzugriff, Shadow und MQTT getestet wurden.

Der Bericht `GGV2Q_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.

Interpretieren derAWS IoT Device Tester Ergebnisse

Der Bericht im Abschnitt `awsiotdevicetester_report.xml` oder `awsiotdevicetester_report.xml` listet die Tests und die Ergebnisse auf, die ausgeführt wurden.

Das erste XML-Tag`<testsuites>` enthält die Zusammenfassung des Testlaufs. 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`

Die Zeitspanne in Sekunden, die die Qualifikationssuite ausgeführt wurde.

`tests`

Die Anzahl der Tests, die durchgeführt wurden.

`failures`

Die Anzahl der ausgeführten Tests, die den Test nicht bestanden haben

`errors`

Die Anzahl der Tests, die IDT nicht ausführen konnte.

`disabled`

Ignoriere dieses Attribut. Sie wird nicht verwendet.

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.

```
<name="aws-iot-greengrass-v2-core" value="supported" type="required"></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 Partner Gerätekatalog auflisten möchten, können Sie diesen Bericht als Qualifikationsnachweis verwenden.

Falls bei Tests Fehler auftreten, können Sie den fehlgeschlagenen Test identifizieren, indem Sie die XML-Tags von `<testsuites>` überprüfen. Die XML-Tags von `<testsuite>` im `<testsuites>`-Tag zeigen die Ergebniszusammenfassung eines Tests für eine Testgruppe. 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. In jedem`<testsuite>` XML-Tag gibt es`<testcase>` Tags für jeden Test, der für eine Testgruppe ausgeführt wurde. 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`

Die Häufigkeit, mit der 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. 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 Logs aus Testläufen in*<devicetester-extract-location>*/`results/`*<execution-id>*`/logs`. Es werden zwei Protokollgruppen generiert:

`test_manager.log`

Mit der Test Manager-Komponente von generierte ProtokolleAWS IoT Device Tester (z. B. Protokolle zur Konfiguration, Testsequenzierung und Berichtsgenerierung).

<test-case-id>`.log (for example, lambdaDeploymentTest.log)`

Protokolle des Testfalls innerhalb der Testgruppe, einschließlich der Protokolle des zu testenden Geräts. Ab IDT v4.2.0 gruppiert IDT die Testprotokolle für jeden Testfall in einem separaten *<test-case-id >* -Ordner innerhalb des*<devicetester-extract-location>*/`results/`*<execution-id>*`/logs/`*<test-group-id>/* Verzeichnisses.

Verwenden Sie IDT, um Ihre eigenen Testsuiten zu entwickeln und auszuführen

Ab IDT v4.0.1 kombiniert IDT für AWS IoT Greengrass V2 ein standardisiertes Konfigurations-Setup und 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 für die Geräteverifizierung zur Verfügung stellen.

Verwenden Sie IDT, um benutzerdefinierte Testsuiten wie folgt zu entwickeln und auszuführen:

So entwickeln Sie benutzerdefinierte Testsuiten

- Erstellen Sie Testsuiten mit benutzerdefinierter Testlogik für das Greengrass-Gerät, das Sie testen möchten.
- Stellen Sie IDT Ihre benutzerdefinierten Testsuiten bereit, um Runner zu testen. Fügen Sie Informationen zu bestimmten Einstellungskonfigurationen für Ihre Testsuiten hinzu.

So führen Sie benutzerdefinierte Testsuiten aus

- Richten Sie das Gerät ein, das Sie testen möchten.
- Implementieren Sie die Einstellungskonfigurationen gemäß den Anforderungen der Testsuiten, die Sie verwenden möchten.
- Verwenden Sie IDT, um Ihre benutzerdefinierten Testsuiten 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

Laden Sie die neueste Version von IDT herunter und extrahieren Sie die Software an einem Speicherort (*<device-tester-extract-location>*) auf Ihrem Dateisystem, an dem Sie über Lese-/Schreibberechtigungen verfügen.

> **Note**
>
> IDT unterstützt nicht die Ausführung durch mehrere Benutzer von einem gemeinsam genutzten Speicherort aus, z. B. einem NFS-Verzeichnis oder einem freigegebenen Windows-Netzwerkordner. Es wird empfohlen, das IDT-Paket auf ein lokales Laufwerk zu extrahieren und die IDT-Binärdatei auf Ihrer lokalen Workstation auszuführen.
> Windows hat eine Pfadlängenbegrenzung von 260 Zeichen. Wenn Sie Windows verwenden, extrahieren Sie IDT in ein Stammverzeichnis wie `C:\` oder `D:\`, um Ihre Pfade unter der Grenze von 260 Zeichen zu halten.

Workflow zur Erstellung einer Testsuite

Testsuiten bestehen aus drei Arten von Dateien:

- Konfigurationsdateien, die IDT mit Informationen zum Ausführen der Testsuite bereitstellen.
- 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 Konfigurationsdateien für Ihre Testsuite.
2. Erstellen Sie ausführbare Testfalldateien, die die Testlogik für Ihre Testsuite enthalten.
3. Überprüfen und dokumentieren Sie die Konfigurationsinformationen, die für Test Runner erforderlich sind, um die Testsuite auszuführen.
4. Stellen Sie sicher, dass IDT Ihre Testsuite ausführen und wie erwartet Testergebnisse liefern kann.

Um schnell eine benutzerdefinierte Beispielsuite zu erstellen und auszuführen, folgen Sie den Anweisungen unter Tutorial: Erstellen und Ausführen der IDT-Beispieltestsuite.

Informationen zum Erstellen einer benutzerdefinierten Testsuite in Python finden Sie unter Tutorial: Entwickeln einer einfachen IDT-Testsuite.

Tutorial: Erstellen und Ausführen der IDT-Beispieltestsuite

Der AWS IoT Device Tester Download enthält den Quellcode für eine Beispiel-Testsuite. Sie können dieses Tutorial abschließen, um die Beispiel-Testsuite zu erstellen und auszuführen, um zu verstehen, wie Sie IDT für verwenden könnenAWS IoT Greengrass, um benutzerdefinierte Testsuiten auszuführen.

In diesem Tutorial führen Sie die folgenden Schritte aus:

1. Erstellen der Beispiel-Testsuite
2. Verwenden Sie IDT, um die Beispiel-Testsuite auszuführen

Voraussetzungen

Zum Durcharbeiten dieses Tutorials ist Folgendes erforderlich:

- Hostcomputer-Anforderungen
  - Neueste Version von AWS IoT Device Tester

- Python 3.7 oder höher

  Führen Sie den folgenden Befehl aus, um die Version von Python zu überprüfen, die auf Ihrem Computer installiert ist:

  ```
  python3 --version
  ```

  Wenn unter Windows mit diesem Befehl ein Fehler zurückgegeben wird, verwenden Sie `python --version` stattdessen . Wenn die zurückgegebene Versionsnummer 3.7 oder höher ist, führen Sie den folgenden Befehl in einem Powershell-Terminal aus, um `python3` als Alias für Ihren `python` Befehl festzulegen.

  ```
  Set-Alias -Name "python3" -Value "python"
  ```

  Wenn keine Versionsinformationen zurückgegeben werden oder die Versionsnummer kleiner als 3.7 ist, folgen Sie den Anweisungen unter Python herunterladen, um Python 3.7+ zu installieren. Weitere Informationen finden Sie in der Python-Dokumentation.

- urllib3

  Führen Sie den folgenden Befehl aus, um zu überprüfen, ob korrekt installiert `urllib3` ist:

  ```
  python3 -c 'import urllib3'
  ```

  Wenn nicht installiert `urllib3` ist, führen Sie den folgenden Befehl aus, um es zu installieren:

  ```
  python3 -m pip install urllib3
  ```

- Anforderungen an Speichergeräte

  - Ein Gerät mit einem Linux-Betriebssystem und einer Netzwerkverbindung zu demselben Netzwerk wie Ihr Hostcomputer.

    Wir empfehlen Ihnen, einen Raspberry Pi mit Raspberry Pi OS zu verwenden. Stellen Sie sicher, dass Sie SSH auf Ihrem Raspberry Pi einrichten, um eine Remote-Verbindung herzustellen.

Konfigurieren von Geräteinformationen für IDT

Konfigurieren Sie Ihre Geräteinformationen für IDT, um den Test auszuführen. Sie müssen die `device.json` Vorlage im *<device-tester-extract-location>*`/configs` Ordner mit den folgenden Informationen aktualisieren.

```
[
  {
    "id": "pool",
    "sku": "N/A",
    "devices": [
      {
        "id": "<device-id>",
        "connectivity": {
          "protocol": "ssh",
          "ip": "<ip-address>",
          "port": "<port>",
          "auth": {
            "method": "pki | password",
            "credentials": {
              "user": "<user-name>",
              "privKeyPath": "/path/to/private/key",
              "password": "<password>"
            }
          }
        }
      }
    ]
  }
]
```

Geben Sie im -`devices`Objekt die folgenden Informationen an:

`id`

Eine benutzerdefinierte eindeutige Kennung für Ihr Gerät.

`connectivity.ip`

Die IP-Adresse Ihres Geräts.

`connectivity.port`

Optional. Die Portnummer, die für SSH-Verbindungen zu Ihrem Gerät verwendet werden soll.

`connectivity.auth`

Authentifizierungsinformationen für die Verbindung.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.auth.method`

Die Authentifizierungsmethode, die für den Zugriff über ein bestimmtes Verbindungsprotokoll auf ein Gerät verwendet wird.

Unterstützte Werte sind:

- `pki`
- `password`

`connectivity.auth.credentials`

Die für die Authentifizierung verwendeten Anmeldeinformationen.

`connectivity.auth.credentials.user`

Der Benutzername, der für die Anmeldung bei Ihrem Gerät verwendet wird.

`connectivity.auth.credentials.privKeyPath`

Der vollständige Pfad zu dem privaten Schlüssel, der für die Anmeldung bei Ihrem Gerät verwendet wurde.

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.

Erstellen der Beispiel-Testsuite

Der *<device-tester-extract-location>*`/samples/python` Ordner enthält Beispielkonfigurationsdateien, Quellcode und das IDT-Client-SDK, das Sie mithilfe der bereitgestellten Build-Skripts in eine Testsuite kombinieren können. Die folgende Verzeichnisstruktur zeigt den Speicherort dieser Beispieldateien:

```
<device-tester-extract-location>
### ...
### tests
### samples
#   ### ...
#   ### python
#       ### configuration
#       ### src
#       ### build-scripts
#           ### build.sh
#           ### build.ps1
### sdks
    ### ...
    ### python
        ### idt_client
```

Um die Testsuite zu erstellen, führen Sie die folgenden Befehle auf Ihrem Host-Computer aus:

Windows

```
cd <device-tester-extract-location>/samples/python/build-scripts
./build.ps1
```

Linux, macOS, or UNIX

```
cd <device-tester-extract-location>/samples/python/build-scripts
./build.sh
```

Dadurch wird die Beispiel-Testsuite im `IDTSampleSuitePython_1.0.0` Ordner im *<device-tester-extract-location>*`/tests` Ordner erstellt. Sehen Sie sich die Dateien im `IDTSampleSuitePython_1.0.0` Ordner an, um zu verstehen, wie die Beispiel-Testsuite strukturiert ist, und um verschiedene Beispiele für ausführbare Testfälle und JSON-Dateien für die Testkonfiguration zu sehen.

> **Note**
>
> Die Beispiel-Testsuite enthält Python-Quellcode. Fügen Sie keine sensiblen Informationen in Ihren Testsuite-Code ein.

Nächster Schritt: Verwenden Sie IDT, um die von Ihnen erstellte Beispiel-Testsuite auszuführen.

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 an die Konsole. Wenn die Ausführung des Tests abgeschlossen ist, sehen Sie die folgenden Informationen:

```
========== Test Summary ==========
Execution Time:         5s
Tests Completed:        4
Tests Passed:           4
Tests Failed:           0
Tests Skipped:          0
---------------------------------
Test Groups:
    sample_group:       PASSED
---------------------------------
Path to IoT Device Tester Report: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/logs
Path to Aggregated JUnit Report: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/IDTSampleSuitePython_Report.xml
```

Fehlerbehebung

Verwenden Sie die folgenden Informationen, um Probleme beim Abschließen des Tutorials zu 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 des Testlaufs helfen kann. Stellen Sie sicher, dass Sie alle Voraussetzungen für dieses Tutorial erfüllen.

Es kann keine Verbindung zum getesteten Gerät hergestellt werden

Überprüfen Sie Folgendes:

- Ihre -`device.json`Datei enthält die richtigen IP-Adresse, Port und Authentifizierungsinformationen.
- Sie können von Ihrem Host-Computer aus eine Verbindung zu Ihrem Gerät über SSH herstellen.

Tutorial: Entwickeln einer einfachen IDT-Testsuite

Eine Testsuite kombiniert Folgendes:

- Testen von ausführbaren Dateien, die die Testlogik enthalten
- Konfigurationsdateien, die die Testsuite beschreiben

Dieses Tutorial zeigt Ihnen, wie Sie IDT für verwenden, AWS IoT Greengrass um eine Python-Testsuite zu entwickeln, die einen einzelnen Testfall enthält. In diesem Tutorial führen Sie die folgenden Schritte aus:

1. Erstellen eines Testsuite-Verzeichnisses
2. Erstellen von Konfigurationsdateien
3. Erstellen der ausführbaren Testfall-Datei
4. Ausführen der Testsuite

Voraussetzungen

Zum Durcharbeiten dieses Tutorials ist Folgendes erforderlich:

- Hostcomputer-Anforderungen
  - Neueste Version von AWS IoT Device Tester

- Python 3.7 oder höher

  Führen Sie den folgenden Befehl aus, um die auf Ihrem Computer installierte Version von Python zu überprüfen:

  ```
  python3 --version
  ```

  Wenn unter Windows bei Verwendung dieses Befehls ein Fehler zurückgegeben wird, verwenden Sie `python --version` stattdessen . Wenn die zurückgegebene Versionsnummer 3.7 oder höher ist, führen Sie den folgenden Befehl in einem Powershell-Terminal aus, um `python3` als Alias für Ihren `python` Befehl festzulegen.

  ```
  Set-Alias -Name "python3" -Value "python"
  ```

  Wenn keine Versionsinformationen zurückgegeben werden oder die Versionsnummer kleiner als 3.7 ist, folgen Sie den Anweisungen unter Python herunterladen, um Python 3.7+ zu installieren. Weitere Informationen finden Sie in der Python-Dokumentation.

- urllib3

  Führen Sie den folgenden Befehl aus, um zu überprüfen, ob korrekt installiert `urllib3` ist:

  ```
  python3 -c 'import urllib3'
  ```

  Wenn nicht installiert `urllib3` ist, führen Sie den folgenden Befehl aus, um es zu installieren:

  ```
  python3 -m pip install urllib3
  ```

- Anforderungen an Speichergeräte

  - Ein Gerät mit einem Linux-Betriebssystem und einer Netzwerkverbindung zu demselben Netzwerk wie Ihr Hostcomputer.

    Wir empfehlen Ihnen, einen Raspberry Pi mit Raspberry Pi OS zu verwenden. Stellen Sie sicher, dass Sie SSH auf Ihrem Raspberry Pi einrichten, um eine Remote-Verbindung herzustellen.

Erstellen eines Testsuite-Verzeichnisses

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 die folgende Verzeichnisstruktur in diesem Ordner:

```
MyTestSuite_1.0.0
### suite
    ### myTestGroup
        ### myTestCase
```

Erstellen von Konfigurationsdateien

Ihre Testsuite muss die folgenden erforderlichen Konfigurationsdateien enthalten:

Erforderliche Konfigurationsdateien

`suite.json`

Enthält Informationen über die Testsuite. Siehe Suite.json konfigurieren.

`group.json`

Enthält Informationen zu einer Testgruppe. Sie müssen für jede Testgruppe in Ihrer Testsuite eine `group.json` Datei erstellen. Siehe Konfigurieren von group.json.

`test.json`

Enthält Informationen zu einem Testfall. Sie müssen für jeden Testfall in Ihrer Testsuite eine `test.json` Datei erstellen. Siehe Konfigurieren von test.json.

1. Erstellen Sie im `MyTestSuite_1.0.0/suite` Ordner eine `suite.json` Datei mit der folgenden Struktur:

   ```
   {
       "id": "MyTestSuite_1.0.0",
       "title": "My Test Suite",
       "details": "This is my test suite.",
       "userDataRequired": false
   }
   ```

2. Erstellen Sie im `MyTestSuite_1.0.0/myTestGroup` Ordner eine `group.json` Datei mit der folgenden Struktur:

```
{
    "id": "MyTestGroup",
    "title": "My Test Group",
    "details": "This is my test group.",
    "optional": false
}
```

3. Erstellen Sie im `MyTestSuite_1.0.0/myTestGroup/myTestCase` Ordner eine `test.json` Datei mit der folgenden Struktur:

```
{
    "id": "MyTestCase",
    "title": "My Test Case",
    "details": "This is my test case.",
    "execution": {
        "timeout": 300000,
        "linux": {
            "cmd": "python3",
            "args": [
                "myTestCase.py"
            ]
        },
        "mac": {
            "cmd": "python3",
            "args": [
                "myTestCase.py"
            ]
        },
        "win": {
            "cmd": "python3",
            "args": [
                "myTestCase.py"
            ]
        }
    }
}
```

Die Verzeichnisstruktur für Ihren `MyTestSuite_1.0.0` Ordner sollte jetzt wie folgt aussehen:

```
MyTestSuite_1.0.0
### suite
```

```
    ### suite.json
    ### myTestGroup
        ### group.json
        ### myTestCase
            ### test.json
```

Abrufen des IDT-Client-SDK

Sie verwenden das IDT-Client-SDK, um es IDT zu ermöglichen, mit dem getesteten Gerät zu interagieren und Testergebnisse zu melden. Für dieses Tutorial verwenden Sie die Python-Version des SDK.

Kopieren Sie den `idt_client` Ordner aus dem *<device-tester-extract-location>*`/sdks/python/` Ordner in Ihren `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase` Ordner.

Führen Sie den folgenden Befehl aus, um zu überprüfen, ob das SDK erfolgreich kopiert wurde.

```
cd MyTestSuite_1.0.0/suite/myTestGroup/myTestCase
python3 -c 'import idt_client'
```

Erstellen der ausführbaren Testfall-Datei

Testfall-ausführbare Dateien enthalten die Testlogik, die Sie ausführen möchten. Eine Testsuite kann mehrere ausführbare Testfälle enthalten. Für dieses Tutorial erstellen Sie nur eine ausführbare Testfalldatei.

1. Erstellen Sie die Testsuite-Datei.

   Erstellen Sie im `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase` Ordner eine `myTestCase.py` Datei mit dem folgenden Inhalt:

   ```
   from idt_client import *

   def main():
       # Use the client SDK to communicate with IDT
       client = Client()

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

2. Verwenden Sie Client-SDK-Funktionen, um Ihrer `myTestCase.py` Datei die folgende Testlogik hinzuzufügen:

a. Führen Sie einen SSH-Befehl auf dem zu testenden Gerät 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()
```

Konfigurieren von Geräteinformationen für IDT

Konfigurieren Sie Ihre Geräteinformationen für IDT, um den Test auszuführen. Sie müssen die `device.json` Vorlage im *<device-tester-extract-location>*`/configs` Ordner mit den folgenden Informationen aktualisieren.

```
[
  {
    "id": "pool",
    "sku": "N/A",
    "devices": [
      {
        "id": "<device-id>",
        "connectivity": {
          "protocol": "ssh",
          "ip": "<ip-address>",
          "port": "<port>",
          "auth": {
            "method": "pki | password",
            "credentials": {
              "user": "<user-name>",
              "privKeyPath": "/path/to/private/key",
              "password": "<password>"
            }
          }
        }
      }
    ]
  }
]
```

Geben Sie im `devices` Objekt die folgenden Informationen an:

`id`

Eine benutzerdefinierte eindeutige Kennung für Ihr Gerät.

`connectivity.ip`

Die IP-Adresse Ihres Geräts.

`connectivity.port`

Optional. Die Portnummer, die für SSH-Verbindungen zu Ihrem Gerät verwendet werden soll.

`connectivity.auth`

Authentifizierungsinformationen für die Verbindung.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.auth.method`

Die Authentifizierungsmethode, die für den Zugriff über ein bestimmtes Verbindungsprotokoll auf ein Gerät verwendet wird.

Unterstützte Werte sind:

- `pki`
- `password`

`connectivity.auth.credentials`

Die für die Authentifizierung verwendeten Anmeldeinformationen.

`connectivity.auth.credentials.user`

Der Benutzername, der für die Anmeldung bei Ihrem Gerät verwendet wird.

`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.

Ausführen der Testsuite

Nachdem Sie Ihre Testsuite erstellt haben, möchten Sie sicherstellen, dass sie wie erwartet funktioniert. Führen Sie dazu die folgenden Schritte aus, um die Testsuite mit Ihrem vorhandenen Gerätepool auszuführen.

1. Kopieren Sie Ihren `MyTestSuite_1.0.0` Ordner in *<device-tester-extract-location>*`/tests`.
2. Führen Sie die folgenden Befehle aus:

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id MyTestSuite
```

IDT führt Ihre Testsuite aus und streamt die Ergebnisse an die Konsole. Wenn die Ausführung des Tests abgeschlossen ist, sehen Sie die folgenden Informationen:

```
time="2020-10-19T09:24:47-07:00" level=info msg=Using pool: pool
time="2020-10-19T09:24:47-07:00" level=info msg=Using test suite "MyTestSuite_1.0.0"
 for execution
time="2020-10-19T09:24:47-07:00" level=info msg=b'hello world\n'
 suiteId=MyTestSuite groupId=myTestGroup testCaseId=myTestCase deviceId=my-device
 executionId=9a52f362-1227-11eb-86c9-8c8590419f30
time="2020-10-19T09:24:47-07:00" level=info msg=All tests finished.
 executionId=9a52f362-1227-11eb-86c9-8c8590419f30
time="2020-10-19T09:24:48-07:00" level=info msg=

========== Test Summary ==========
Execution Time:         1s
Tests Completed:        1
Tests Passed:           1
Tests Failed:           0
Tests Skipped:          0
----------------------------------
Test Groups:
    myTestGroup:        PASSED
----------------------------------
```

```
Path to IoT Device Tester Report: /path/to/devicetester/
results/9a52f362-1227-11eb-86c9-8c8590419f30/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/devicetester/
results/9a52f362-1227-11eb-86c9-8c8590419f30/logs
Path to Aggregated JUnit Report: /path/to/devicetester/
results/9a52f362-1227-11eb-86c9-8c8590419f30/MyTestSuite_Report.xml
```

Fehlerbehebung

Verwenden Sie die folgenden Informationen, um Probleme beim Abschließen des Tutorials zu 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 des Testlaufs helfen kann. Bevor Sie die Fehlerprotokolle überprüfen, überprüfen Sie Folgendes:

- Das IDT-Client-SDK befindet sich im richtigen Ordner, wie in diesem Schritt beschrieben.
- Sie erfüllen alle Voraussetzungen für dieses Tutorial.

Es kann keine Verbindung zum getesteten Gerät hergestellt werden

Überprüfen Sie Folgendes:

- Ihre -`device.json`Datei enthält die richtigen IP-Adresse, Port und Authentifizierungsinformationen.
- Sie können von Ihrem Hostcomputer aus eine Verbindung zu Ihrem Gerät über SSH herstellen.

Erstellen von IDT-Testsuite-Konfigurationsdateien

In diesem Abschnitt werden die Formate beschrieben, in denen Sie Konfigurationsdateien erstellen, die Sie beim Schreiben einer benutzerdefinierten Testsuite angeben.

Erforderliche Konfigurationsdateien

`suite.json`

Enthält Informationen zur Testsuite. Siehe Suite.json konfigurieren.

`group.json`

Enthält Informationen zu einer Testgruppe. Sie müssen für jede Testgruppe in Ihrer Testsuite eine `group.json` Datei erstellen. Siehe Konfigurieren von group.json.

`test.json`

Enthält Informationen zu einem Testfall. Sie müssen für jeden Testfall in Ihrer Testsuite eine `test.json` Datei erstellen. Siehe Konfigurieren von test.json.

Optionale Konfigurationsdateien

`test_orchestrator.yaml` oder `state_machine.json`

Definiert, wie Tests ausgeführt werden, wenn IDT die Testsuite ausführt. SSe Konfigurieren von test_orchestrator.yaml.

> **Note**
>
> Ab IDT v4.5.1 verwenden Sie die `-test_orchestrator.yaml`Datei, um den Test-Workflow zu definieren. In früheren Versionen von IDT verwenden Sie die `-state_machine.json`Datei. Weitere Informationen zum Zustandsautomaten finden Sie unter Konfigurieren Sie den IDT-Statuscomputer.

`userdata_schema.json`

Definiert das Schema für die `userdata.json` Datei, die Test-Runner in ihre Einstellungskonfiguration aufnehmen können. Die `userdata.json` Datei wird für alle zusätzlichen Konfigurationsinformationen verwendet, die zum Ausführen des Tests erforderlich sind, aber nicht in der `device.json` Datei vorhanden sind. Siehe Konfigurieren von userdata_schema.json.

Konfigurationsdateien werden *`<custom-test-suite-folder>`* wie hier gezeigt in Ihrem abgelegt.

```
<custom-test-suite-folder>
### suite
    ### suite.json
    ### test_orchestrator.yaml
```

```
    ### userdata_schema.json
    ### <test-group-folder>
        ### group.json
        ### <test-case-folder>
            ### test.json
```

Suite.json konfigurieren

Die `suite.json` Datei legt Umgebungsvariablen fest und bestimmt, ob Benutzerdaten zum Ausführen der Testsuite erforderlich sind. Verwenden Sie die folgende Vorlage, um Ihre -*<custom-test-suite-folder>*`/suite/suite.json`Datei zu konfigurieren:

```
{
    "id": "<suite-name>_<suite-version>",
    "title": "<suite-title>",
    "details": "<suite-details>",
    "userDataRequired": true | false,
    "environmentVariables": [
        {
            "key": "<name>",
            "value": "<value>",
        },
        ...
        {
            "key": "<name>",
            "value": "<value>",
        }
    ]
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`id`

Eine eindeutige benutzerdefinierte ID für die Testsuite. Der Wert von `id` muss mit dem Namen des Testsuite-Ordners übereinstimmen, in dem sich die `suite.json` Datei befindet. Der Suite-Name und die Suite-Version müssen außerdem die folgenden Anforderungen erfüllen:

- *<suite-name>* darf keine Unterstriche enthalten.
- *<suite-version>* wird als bezeichnet*x.x.x*, wobei eine Zahl x ist.

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 Test Runner angezeigt.

`details`

Eine kurze Beschreibung des Zwecks der Testsuite.

`userDataRequired`

Definiert, ob Test-Runner benutzerdefinierte Informationen in eine `userdata.json` Datei aufnehmen müssen. Wenn Sie diesen Wert auf festlegen`true`, müssen Sie auch die -`userdata_schema.json`Datei in Ihren Testsuite-Ordner aufnehmen.

`environmentVariables`

Optional. Ein Array von Umgebungsvariablen, die für diese Testsuite festgelegt werden sollen.

`environmentVariables.key`

Der Name der Umgebungsvariable.

`environmentVariables.value`

Der Wert der Umgebungsvariable.

Konfigurieren von group.json

Die `group.json` Datei definiert, ob eine Testgruppe erforderlich oder optional ist. Verwenden Sie die folgende Vorlage, um Ihre -*<custom-test-suite-folder>*`/suite/`*<test-group>*`/group.json`Datei zu konfigurieren:

```
{
    "id": "<group-id>",
    "title": "<group-title>",
    "details": "<group-details>",
    "optional": true | false,
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`id`

Eine eindeutige benutzerdefinierte ID für die Testgruppe. Der Wert von `id` muss mit dem Namen des Testgruppenordners übereinstimmen, in dem sich die `group.json` Datei befindet, und darf keine Unterstriche () enthalten_. Die ID wird in IDT-generierten Testberichten verwendet.

`title`

Ein beschreibender Name für die Testgruppe. Der Name wird in der IDT-CLI für Test Runner angezeigt.

`details`

Eine kurze Beschreibung des Zwecks der Testgruppe.

`optional`

Optional. Legen Sie den Wert auf fest`true`, um diese Testgruppe als optionale Gruppe anzuzeigen, nachdem IDT die Ausführung der erforderlichen Tests abgeschlossen hat. Der Standardwert ist `false`.

Konfigurieren von test.json

Die `test.json` Datei bestimmt die ausführbaren Testfall-Dateien und die Umgebungsvariablen, die von einem Testfall verwendet werden. Weitere Informationen zum Erstellen von ausführbaren Testfalldateien finden Sie unter Ausführbare IDT-Testfalldateien erstellen.

Verwenden Sie die folgende Vorlage, um Ihre -*<custom-test-suite-folder>*/suite/*<test-group>*/*<test-case>*/test.jsonDatei zu konfigurieren:

```
{
    "id": "<test-id>",
    "title": "<test-title>",
    "details": "<test-details>",
    "requireDUT": true | false,
    "requiredResources": [
        {
            "name": "<resource-name>",
            "features": [
                {
                    "name": "<feature-name>",
                    "version": "<feature-version>",
                    "jobSlots": <job-slots>
                }
```

```
            ]
        }
    ],
    "execution": {
        "timeout": <timeout>,
        "mac": {
            "cmd": "/path/to/executable",
            "args": [
                "<argument>"
            ],
        },
        "linux": {
            "cmd": "/path/to/executable",
            "args": [
                "<argument>"
            ],
        },
        "win": {
            "cmd": "/path/to/executable",
            "args": [
                "<argument>"
            ]
        }
    },
    "environmentVariables": [
        {
            "key": "<name>",
            "value": "<value>",
        }
    ]
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`id`

Eine eindeutige benutzerdefinierte ID für den Testfall. Der Wert von `id` muss mit dem Namen des Testfallordners übereinstimmen, in dem sich die `test.json` Datei befindet, und darf keine Unterstriche () enthalten_. Die ID wird in IDT-generierten Testberichten verwendet.

`title`

Ein beschreibender Name für den Testfall. Der Name wird in der IDT-CLI für Test Runner angezeigt.

`details`

Eine kurze Beschreibung des Zwecks des Testfalls.

`requireDUT`

Optional. Setzen Sie auf , `true` wenn ein Gerät erforderlich ist, um diesen Test auszuführen, andernfalls auf `false`. Der Standardwert ist `true`. Test Runner konfigurieren die Geräte, die sie zum Ausführen des Tests in ihrer -`device.json`Datei verwenden werden.

`requiredResources`

Optional. Ein Array, das Informationen über Ressourcengeräte bereitstellt, die für die Ausführung dieses Tests erforderlich sind.

`requiredResources.name`

Der eindeutige Name, der dem Ressourcengerät gegeben werden soll, wenn dieser Test ausgeführt wird.

`requiredResources.features`

Ein Array von benutzerdefinierten Funktionen für Ressourcengeräte.

`requiredResources.features.name`

Der Name des Features. Die Gerätefunktion, für die Sie dieses Gerät verwenden möchten. Dieser Name wird mit dem Feature-Namen abgeglichen, der vom Test-Runner in der `resource.json` Datei bereitgestellt wird.

`requiredResources.features.version`

Optional. Die Version der Funktion. Dieser Wert wird mit der Feature-Version abgeglichen, die vom Test-Runner in der `resource.json` Datei bereitgestellt wird. Wenn keine Version angegeben wird, wird die Funktion nicht überprüft. Wenn für das Feature keine Versionsnummer erforderlich ist, lassen Sie dieses Feld leer.

`requiredResources.features.jobSlots`

Optional. Die Anzahl der gleichzeitigen Tests, die dieses Feature unterstützen kann. Der Standardwert ist `1`. Wenn Sie möchten, dass IDT unterschiedliche Geräte für einzelne Funktionen verwendet, empfehlen wir Ihnen, diesen Wert auf festzulegen`1`.

`execution.timeout`

Die Zeit (in Millisekunden), die IDT auf den Abschluss des Tests wartet. Weitere Informationen zum Festlegen dieses Werts finden Sie unter Ausführbare IDT-Testfalldateien erstellen.

`execution.os`

Die ausführbaren Testfall-Dateien, 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 Testfall-Datei, die Sie für das angegebene Betriebssystem ausführen möchten. Dieser Speicherort muss sich im Systempfad befinden.

`execution.os.args`

Optional. Die Argumente, die für die Ausführung der ausführbaren Testfall-Datei angegeben werden sollen.

`environmentVariables`

Optional. Ein Array von Umgebungsvariablen, die für diesen Testfall festgelegt wurden.

`environmentVariables.key`

Der Name der Umgebungsvariable.

`environmentVariables.value`

Der Wert der Umgebungsvariable.

> **Note**
>
> Wenn Sie dieselbe Umgebungsvariable in der `test.json` Datei und in der `suite.json` Datei angeben, hat der Wert in der `test.json` Datei Vorrang.

Konfigurieren von test_orchestrator.yaml

Ein Testorchestrator ist ein Konstrukt, das den Testsuite-Ausführungsfluss steuert. Sie bestimmt den Startstatus einer Testsuite, verwaltet Statusübergänge basierend auf benutzerdefinierten Regeln und wechselt weiterhin durch diese Zustände, bis sie den Endstatus erreicht.

Wenn Ihre Testsuite keinen benutzerdefinierten Testorchestrator enthält, generiert IDT einen Testorchestrator für Sie.

Der Standard-Testorchestrator führt die folgenden Funktionen aus:

- Bietet Test Runnern die Möglichkeit, bestimmte Testgruppen anstelle der gesamten Testsuite auszuwählen und auszuführen.
- Wenn bestimmte Testgruppen nicht ausgewählt sind, führt jede Testgruppe in der Testsuite nach dem Zufallsprinzip aus.
- Generiert Berichte und druckt eine Konsolenübersicht, die die Testergebnisse für jede Testgruppe und jeden Testfall anzeigt.

Weitere Informationen darüber, wie der IDT-Testorchestrator funktioniert, finden Sie unter Konfigurieren Sie den IDT-Test-Orchestrator.

Konfigurieren von userdata_schema.json

Die Datei `userdata_schema.json` bestimmt das Schema, in dem Test Runner Benutzerdaten bereitstellen. Benutzerdaten sind erforderlich, wenn Ihre Testsuite Informationen erfordert, die nicht in der `device.json` Datei vorhanden sind. Ihre Tests benötigen beispielsweise möglicherweise Wi-Fi-Netzwerkanmeldeinformationen, bestimmte offene Ports oder Zertifikate, die ein Benutzer bereitstellen muss. Diese Informationen können IDT als Eingabeparameter mit dem Namen zur Verfügung gestellt werden`userdata`. Dabei handelt es sich um den Wert, für den es sich um eine `userdata.json` Datei handelt, die Benutzer in ihrem *`<device-tester-extract-location>`*`/config` Ordner erstellen. Das Format der `userdata.json` Datei basiert auf der `userdata_schema.json` Datei, die Sie in die Testsuite aufnehmen.

So geben Sie an, dass Test Runner eine -`userdata.json`Datei bereitstellen müssen:

1. Setzen Sie in der `suite.json` Datei `userDataRequired` auf `true`.
2. *`<custom-test-suite-folder>`*Erstellen Sie in Ihrem eine -`userdata_schema.json`Datei.
3. Bearbeiten Sie die `userdata_schema.json` Datei , um ein gültiges JSON-Schema für IETF Entwurf v4 zu erstellen.

Wenn IDT Ihre Testsuite ausführt, liest es das Schema automatisch und verwendet es, um die vom Test-Runner bereitgestellte `userdata.json` Datei zu validieren. Falls gültig, ist der Inhalt der `userdata.json` Datei sowohl im IDT-Kontext als auch im Testorchestrator-Kontext verfügbar.

Konfigurieren Sie den IDT-Test-Orchestrator

Ab IDT v4.5.1 enthält IDT ein neuesTest-OrchestratorKomponente. Der Testorchestrator ist eine IDT-Komponente, die den Ausführungsablauf der Testsuite steuert und den Testbericht generiert,

nachdem IDT alle Tests ausgeführt hat. Der Testorchestrator bestimmt die Testauswahl und die Reihenfolge, in der Tests ausgeführt werden, basierend auf benutzerdefinierten Regeln.

Wenn Ihre Testsuite keinen benutzerdefinierten Testorchestrator enthält, generiert IDT einen Testorchestrator für Sie.

Der Standard-Testorchestrator 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 Testorchestrator ersetzt den IDT-Testorchestrator. Wir empfehlen dringend, dass Sie den Testorchestrator verwenden, um Ihre Test-Suiten anstelle des IDT-Testorchestrators zu entwickeln. Der Test-Orchestrator bietet die folgenden verbesserten Funktionen:

- Verwendet ein deklaratives Format im Vergleich zu dem zwingenden Format, das der IDT-Statuscomputer verwendet. Dies ermöglicht es Ihnenspezifizierenwelche Tests Sie ausführen möchten undwenndu willst sie laufen lassenaus.
- Verwaltet die spezifische Gruppenbehandlung, Berichtsgenerierung, Fehlerbehandlung und Ergebnisverfolgungdamit Sie nicht benötigt werdenum diese Aktionen manuell zu verwalten.
- Verwendet das YAML-Format, das standardmäßig Kommentare unterstützt.
- Benötigt80 Prozentweniger Speicherplatz als der Test-Orchestrator, um denselben Workflow zu definieren.
- Fügt eine Validierung vor dem Test hinzu, um sicherzustellen, dass Ihre Workflow-Definition keine falschen Test-IDs oder zirkuläre Abhängigkeiten enthält

Test-Orchestrator-Format

Sie können die folgende Vorlage verwenden, um Ihre eigene zu konfigurieren*<custom-test-suite-folder>*`/suite/test_orchestrator.yaml`file:

```
Aliases:
```

```
  string: context-expression

ConditionalTests:
  - Condition: context-expression
    Tests:
      - test-descriptor

Order:
  - - group-descriptor
    - group-descriptor

Features:
  - Name: feature-name
    Value: support-description
    Condition: context-expression
    Tests:
        - test-descriptor
    OneOfTests:
        - test-descriptor
    IsRequired: boolean
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`Aliases`

Optional. Benutzerdefinierte Zeichenfolgen, die Kontextausdrücken zugeordnet sind. Aliase ermöglichen es Ihnen, Anzeigenamen zu generierenidentifizieren Sie Kontextausdrücke in Ihrer Test-Orchestrator-Konfiguration. Dies ist besonders nützlich, wenn Sie komplexe Kontextausdrücke oder -ausdrücke erstellen, die Sie an mehreren Stellen verwenden.

Sie können Kontextausdrücke verwenden, um Kontextabfragen zu speichern, mit denen Sie auf Daten aus anderen IDT-Konfigurationen zugreifen können. Weitere Informationen finden Sie unter Zugriff auf Daten im Kontext.

Example Beispiel

```
Aliases:
    FizzChosen: "'{{$pool.features[?(@.name == 'Fizz')].value[0]}}' == 'yes'"
    BuzzChosen: "'{{$pool.features[?(@.name == 'Buzz')].value[0]}}' == 'yes'"
    FizzBuzzChosen: "'{{$aliases.FizzChosen}}' && '{{$aliases.BuzzChosen}}'"
```

`ConditionalTests`

Optional. Eine Liste der Bedingungen und die entsprechenden Testfälle, die ausgeführt werden, wenn jede Bedingung erfüllt ist. Jede Bedingung kann mehrere Testfälle haben; Sie können jedoch einen bestimmten Testfall nur einer Bedingung zuweisen.

Standardmäßig führt IDT jeden Testfall aus, der keiner Bedingung in dieser Liste zugewiesen ist. Wenn Sie diesen Abschnitt nicht angeben, führt IDT alle Testgruppen in der Testsuite aus.

Jeder Artikel im`ConditionalTests`Die Liste enthält die folgenden Parameter:

`Condition`

Ein Kontextausdruck, der zu einer ausgewertet wirdBooleschWert. Wenn der ausgewertete Wert true ist, führt IDT die Testfälle aus, die im`Tests`-Parameter.

`Tests`

Die Liste der Testdeskriptoren.

Jeder Testdeskriptor verwendet die Testgruppen-ID und eine oder mehrere Testfall-IDs, um die einzelnen Tests zu identifizieren, die von einer bestimmten Testgruppe aus ausgeführt werden sollen. Der Testdeskriptor verwendet das folgende Format:

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

Example Beispiel

Im folgenden Beispiel werden generische Kontextausdrücke verwendet, die Sie als definieren können`Aliases`aus.

```
ConditionalTests:
    - Condition: "{{$aliases.Condition1}}"
      Tests:
          - GroupId: A
          - GroupId: B
    - Condition: "{{$aliases.Condition2}}"
      Tests:
          - GroupId: D
    - Condition: "{{$aliases.Condition1}} || {{$aliases.Condition2}}"
      Tests:
```

```
            - GroupId: C
```

Basierend auf den definierten Bedingungen wählt IDT Testgruppen wie folgt aus:

- Wenn`Condition1`stimmt, IDT führt die Tests in den Testgruppen A, B und C.
- Wenn`Condition2`stimmt, IDT führt die Tests in den Testgruppen C und D durch.

`Order`

Optional. Die Reihenfolge, in der Tests durchgeführt werden sollen. Sie geben die Testreihenfolge auf Testgruppenebene an. Wenn Sie diesen Abschnitt nicht angeben, führt IDT alle anwendbaren Testgruppen in einer zufälligen Reihenfolge aus. Der Wert von`Order`ist eine Liste von Gruppendeskriptorlisten. Jede Testgruppe, in der Sie nicht auflisten`Order`, kann parallel zu jeder anderen aufgelisteten Testgruppe ausgeführt werden.

Jede Gruppendeskriptorliste enthält einen von mehreren Gruppendeskriptoren und gibt die Reihenfolge an, in der die Gruppen ausgeführt werden sollen, die in jedem Deskriptor angegeben sind. Sie können die folgenden Formate verwenden, um einzelne Gruppendeskriptoren zu definieren:

- *`group-id`*— Die Gruppen-ID einer vorhandenen Testgruppe.
- [*`group-id`*, *`group-id`*]— Liste der Testgruppen, die in beliebiger Reihenfolge relativ zueinander ausgeführt werden können.
- "*"—Platzhalterzeichen. Dies entspricht der Liste aller Testgruppen, die nicht bereits in der aktuellen Gruppendeskriptorliste angegeben sind.

Der Wert für`Order`Sie müssen auch die folgenden Anforderungen erfüllen:

- Testgruppen-IDs, die Sie in einem Gruppendeskriptor angeben, müssen in Ihrer Testsuite vorhanden sein.
- Jede Gruppendeskriptor-Liste muss mindestens eine Testgruppe umfassen.
- Jede Gruppendeskriptorliste muss eindeutige Gruppen-IDs enthalten. Sie können eine Testgruppen-ID nicht innerhalb einzelner Gruppendeskriptoren wiederholen.
- Eine Gruppendeskriptor-Liste kann höchstens über einen Platzhalterdeskriptor verfügen. Der Platzhaltergruppen-Deskriptor muss das erste oder letzte Element in der Liste sein.

Example Beispiele

Für eine Testsuite, die die Testgruppen A, B, C, D und E enthält, zeigt die folgende Beispielliste verschiedene Möglichkeiten, um anzugeben, dass IDT zuerst Testgruppe A ausführen, dann

die Testgruppe B ausführen und dann die Testgruppen C, D und E in beliebiger Reihenfolge ausführen soll.

- ```
  Order:
      - - A
        - B
        - [C, D, E]
  ```
- ```
  Order:
      - - A
        - B
        - "*"
  ```
- ```
  Order:
      - - A
        - B

      - - B
        - C

      - - B
        - D

      - - B
        - E
  ```

`Features`

Optional. Die Liste der Produktfunktionen, die IDT dem`awsiotdevicetester_report.xml`file. Wenn Sie diesen Abschnitt nicht angeben, fügt IDT dem Bericht keine Produktfunktionen hinzu.

Ein Produktmerkmal sind benutzerdefinierte Informationen zu bestimmten Kriterien, die ein Gerät möglicherweise erfüllen kann. Beispielsweise kann die MQTT-Produktfunktion angeben, dass das Gerät MQTT-Nachrichten ordnungsgemäß veröffentlicht. In :`awsiotdevicetester_report.xml`werden Produkteigenschaften als`supported`,`not-supported`oder ein benutzerdefinierter benutzerdefinierter Wert, basierend darauf, ob bestimmte Tests bestanden wurden.

Jeder Artikel im`Features`Die Liste enthält die folgenden Parameter:

`Name`

Der Name der Funktion.
  ```

`Value`

Optional. Der benutzerdefinierte Wert, den Sie im Bericht anstelle von verwenden möchten`supported`aus. Wenn dieser Wert nicht angegeben wird, legt basierend IDT den Feature-Wert auf`supported`oder`not-supported`basierend auf Testergebnissen. Wenn Sie dasselbe Feature mit verschiedenen Bedingungen testen, können Sie einen benutzerdefinierten Wert für jede Instanz dieses Features im`Features`list, und IDT verkettet die Feature-Werte für unterstützte Bedingungen. Weitere Informationen finden Sie unter

`Condition`

Ein Kontextausdruck, der zu einer ausgewertet wirdBooleschWert. Wenn der ausgewertete Wert true ist, fügt IDT die Funktion dem Testbericht hinzu, nachdem die Testsuite ausgeführt wurde. Wenn der ausgewertete Wert false ist, ist der Test nicht im Bericht enthalten.

`Tests`

Optional. Die Liste der Testdeskriptoren. Alle Tests, die in dieser Liste angegeben sind, müssen bestehen, damit das Feature unterstützt wird.

Jeder Testdeskriptor in dieser Liste verwendet die Testgruppen-ID und eine oder mehrere Testfall-IDs, um die einzelnen Tests zu identifizieren, die von einer bestimmten Testgruppe aus ausgeführt werden sollen. Der Testdeskriptor verwendet das folgende Format:

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

Sie müssen entweder angeben`Tests`oder`OneOfTests`für jedes Feature im`Features`-Liste.

`OneOfTests`

Optional. Die Liste der Testdeskriptoren. Mindestens einer der in dieser Liste angegebenen Tests muss bestehen, damit das Feature unterstützt wird.

Jeder Testdeskriptor in dieser Liste verwendet die Testgruppen-ID und eine oder mehrere Testfall-IDs, um die einzelnen Tests zu identifizieren, die von einer bestimmten Testgruppe aus ausgeführt werden sollen. Der Testdeskriptor verwendet das folgende Format:

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

Sie müssen entweder angeben`Tests`oder`OneOfTests`für jedes Feature im`Features`-Liste.

`IsRequired`

Der boolesche Wert, der definiert, ob das Feature im Testbericht erforderlich ist. Der Standardwert ist `false`.

Example

Testen von Orchestrierung

Der Test Orchestrator-Kontext ist ein schreibgeschütztes JSON-Dokument, das Daten enthält, die dem Testorchestrator während der Ausführung zur Verfügung stehen. Der Test Orchestrator-Kontext ist nur vom Testorchestrator 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 Test Orchestrierkontext verwendet das folgende Format:

```
{
    "pool": {
        <device-json-pool-element>
    },
    "userData": {
        <userdata-json-content>
    },
    "config": {
        <config-json-content>
    }
}
```

`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 im`config.json`file.

Sie können den Kontext mit der JsonPath-Notation abfragen. Die Syntax für JsonPath-Abfragen in Statusdefinitionen lautet`{{query}}`aus. Wenn Sie auf Daten aus dem Testorchestrator-Kontext zugreifen, stellen Sie sicher, dass jeder Wert zu einer Zeichenfolge, einer Zahl oder einemBooleschaus.

Weitere Informationen zur Verwendung von JSONPath-Notation für den Zugriff auf Daten aus dem Kontext finden Sie unterVerwenden Sie den IDT-Kontextaus.

Konfigurieren Sie den IDT-Statuscomputer

> **Important**
>
> Ab IDT v4.5.1 ist dieser Zustandscomputer veraltet. Es wird dringend empfohlen, den neuen Test-Orchestrator zu verwenden. Weitere Informationen finden Sie unter Konfigurieren Sie den IDT-Test-Orchestrator.

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 entwederSucceedoderFailaus.

Format des Zustandsautomaten

Sie können die folgende Vorlage verwenden, um Ihre eigene zu konfigurieren*<custom-test-suite-folder>*/suite/state_machine.jsonfile:

```
{
  "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 nächsten Zustands, in dem IDT mit der Ausführung der Testsuite beginnt. Der Wert von`StartAt`muss auf einen der in der`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.

DieStatesObjekt muss dasSucceedundFail-Staaten. 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 von einer in der Testsuite definierten Testgruppe aus.

```
{
    "Type": "RunTask",
    "Next": "<state-name>",
    "TestGroup": "<group-id>",
    "TestCases": [
        "<test-id>"
    ],
    "ResultVar": "<result-name>"
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`Next`

Der Name des 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 beide`TestGroup`und`TestCases`angegeben sind, führt IDT die angegebenen Testfälle aus der Testgruppe aus.
- Wann`TestCases`sind angegeben aber`TestGroup`ist nicht angegeben, IDT führt die angegebenen Testfälle aus.
- Wann`TestGroup`ist angegeben, aber`TestCases`Ist nicht angegeben, führt IDT alle Testfälle innerhalb der angegebenen Testgruppe aus.
- Wenn keines`TestGroup`oder`TestCases`angegeben 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 einschließen`RunTask`und`Choice`Bundesstaaten in deinem`state_machine.json`file. 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 “Die IDB CLI-Befehle”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 haben`TestGroup`aus. IDT legt den Wert der Variablen fest, in der Sie definieren`ResultVar`zu`true`oder`false`basierend 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 wird verwendet`RunTask`status, 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 die`RunTaskError`Fehler bei der-Ausführung. Wenn der Status auf einen Ausführungsfehler stößt, wird auch die`hasExecutionError`Variable im Zustandsmaschine-Kontext zu`true`aus.

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 definierten Ausdrücke`Choices`kann ausgewertet werden`true`aus.

`FallthroughOnError`

Optional. Gibt das Verhalten an, wenn der Status bei der Auswertung von Ausdrücken auf einen Fehler stößt. Stellen Sie auf`true`wenn Sie einen Ausdruck überspringen möchten, wenn die Auswertung zu einem Fehler führt. Wenn keine Ausdrücke übereinstimmen, wechselt der

Zustandsautomat in den`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`, wechselt der Zustandsautomat 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 des Zustandsautomatenaus.

`Choices.Next`

Der Name des nächsten Zustands, wenn der in definierte Ausdruck`Choices.Expression`wertet nach`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`blockieren, 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, dass Sie festlegen`FallthroughOnError`zu`true`, und abhängig von Ihrem Anwendungsfall führen Sie eine der folgenden Maßnahmen durch:

- 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`-Staaten. 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 übermitteln, 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 unterBeispiel für einen Zustandsautomaten: Führen Sie zwei Testgruppen parallel ausaus.

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 des Features

`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 den`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. Stellen Sie auf`false`um diese Funktion als optionales Feature im Bericht zu markieren. Der Standardwert ist `true`.

`ExecutionMethods`

Optional. Ein Array von Ausführungsmethoden, die mit der`protocol`Der in der`device.json`file. Wenn dieser Wert angegeben wird, müssen Testläufer eine angeben`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 denAddProductFeaturesstate, müssen Sie den Wert vonResultVarimRunTaskGeben Sie einen der folgenden Werte an:

- Wenn Sie einzelne Testfall-IDs angegeben haben, setzen SieResultVarzu*group-id_test-id*_passedaus.
- Wenn Sie keine einzelnen Testfall-IDs angegeben haben, setzen SieResultVarzu*group-id*_passedaus.

DieAddProductFeaturesStatusprüfungen für Testergebnisse in der folgenden Weise:

- Wenn Sie keine Testfall-IDs angegeben haben, wird das Ergebnis für jede Testgruppe aus dem Wert des*group-id*_passedvariabel im Zustandsmaschinen-Kontext.
- Wenn Sie Testfall-IDs angegeben haben, wird das Ergebnis für jeden der Tests aus dem Wert des*group-id_test-id*_passedvariabel im Zustandsmaschinen-Kontext.

Fehlerbehandlung

Wenn eine in diesem Status angegebene Gruppen-ID keine gültige Gruppen-ID ist, führt dieser Status zuAddProductFeaturesErrorFehler bei der-Ausführung. Wenn der Status auf einen Ausführungsfehler stößt, wird auch diehasExecutionErrorsVariable im Zustandsmaschine-Kontext zutrueaus.

Bericht

DieReportDer Zustand generiert den*suite-name*_Report.xmlundawsiotdevicetester_report.xmlDateien. 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 in die 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`-Staaten.

```
{
    "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 des 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 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`-Staaten.
- 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 in die Konsole.

Sie können die folgenden Methoden verwenden, um Ausführungsfehler zu behandeln:

- Hinzufügen einer`Catch`Blockin der Statusdefinition.
- Überprüfen Sie den Wert des`hasExecutionErrors`WertIm Kontext des Zustandsautomaten.

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`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 zu`true`im Zustandsautomatenkontext. 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 Beispielkonfigurationen für Zustandsautomaten.

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 vorhanden sein muss`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, die Funktion 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, die Funktion 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 jedoch 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"
        }
    }
}
```

Ausführbare IDT-Testfalldateien erstellen

Sie können auf folgende Weise ausführbare Testfalldateien erstellen und in einem Testsuite-Ordner ablegen:

- Für Testsuiten, die Argumente oder Umgebungsvariablen aus den`test.json` Dateien verwenden, um zu bestimmen, welche Tests ausgeführt werden sollen, können Sie 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 auf der Grundlage bestimmter Befehle ausführen möchten, erstellen Sie für jeden Testfall in der Testsuite eine ausführbare Testfalldatei.

Als Testautor können Sie bestimmen, welcher Ansatz für Ihren Anwendungsfall geeignet ist, und Ihre ausführbare Testfalldatei entsprechend strukturieren. Stellen Sie sicher, dass Sie in jeder`test.json` Datei den richtigen Pfad zur ausführbaren Testfalldatei angeben und dass die angegebene ausführbare Datei korrekt ausgeführt wird.

Wenn alle Geräte für die Ausführung eines Testfalls bereit sind, liest IDT die folgenden Dateien:

- Das`test.json` für den ausgewählten Testfall bestimmt die zu startenden Prozesse und die zu setzenden Umgebungsvariablen.
- Die`suite.json` für die Testsuite bestimmt die zu setzenden Umgebungsvariablen.

IDT startet den erforderlichen ausführbaren Testprozess auf der Grundlage der in der`test.json` Datei angegebenen Befehle und Argumente und übergibt die erforderlichen Umgebungsvariablen an den Prozess.

Verwenden Sie das IDT Client SDK

Mit den IDT-Client-SDKs können Sie das Schreiben von Testlogik in Ihre Testdatei mit API-Befehlen vereinfachen, die Sie verwenden können, um mit IDT und Ihren zu testenden Geräten zu interagieren. IDT stellt derzeit die folgenden SDKs bereit:

- IDT
- SDK for Go
- SDK for Java

Diese SDKs befinden sich in dem`<device-tester-extract-location>`/`sdks` Ordner. Wenn Sie eine neue ausführbare Testfalldatei erstellen, müssen Sie das SDK, das Sie verwenden möchten, in den Ordner kopieren, der die ausführbare Testfalldatei enthält, und das SDK in Ihrem Code referenzieren. Dieser Abschnitt enthält eine kurze Beschreibung der verfügbaren API-Befehle, die Sie in Ihren ausführbaren Testfalldateien verwenden können.

In diesem Abschnitt

Interaktion mit dem Gerät

Mit den folgenden Befehlen können Sie mit dem zu testenden Gerät kommunizieren, ohne zusätzliche Funktionen für die Geräteinteraktion und das Konnektivitätsmanagement implementieren zu müssen.

`ExecuteOnDevice`

Ermöglicht Testsuiten das Ausführen von Shell-Befehlen auf einem Gerät, das SSH- oder Docker-Shell-Verbindungen unterstützt.

`CopyToDevice`

Ermöglicht Testsuiten, eine lokale Datei vom Host-Computer, auf dem IDT ausgeführt wird, an einen bestimmten Ort auf einem Gerät zu kopieren, das SSH- oder Docker-Shell-Verbindungen unterstützt.

`ReadFromDevice`

Ermöglicht Testsuiten das Lesen von der seriellen Schnittstelle von Geräten, die UART-Verbindungen unterstützen.

> **Note**
>
> Da IDT keine direkten Verbindungen zu Geräten verwaltet, die mithilfe von Gerätezugriffsinformationen aus dem Kontext hergestellt werden, empfehlen wir, diese API-Befehle für Geräteinteraktionen in Ihren ausführbaren Testfalldateien zu verwenden. Wenn diese Befehle jedoch nicht Ihren Testfallanforderungen entsprechen, können Sie die

Gerätezugriffsinformationen aus dem IDT-Kontext abrufen und damit eine direkte Verbindung zum Gerät aus der Testsuite herstellen.
Um eine direkte Verbindung herzustellen, rufen Sie die Informationen in den Felderndevice.connectivity und in denresource.devices.connectivity Feldern für Ihr zu testendes Gerät bzw. für Ressourcengeräte ab. Weitere Informationen zur Verwendung des IDTVerwenden Sie den IDT-Kontext

IDT-Interaktion

Die folgenden Befehle ermöglichen es Ihren Testsuiten, mit IDT zu kommunizieren.

`PollForNotifications`

Ermöglicht Testsuiten, nach Benachrichtigungen von IDT zu suchen.

`GetContextValue` und `GetContextString`

Ermöglicht Testsuiten das Abrufen von Werten aus dem IDT-Kontext. Weitere Informationen finden Sie unter Verwenden Sie den IDT-Kontext.

`SendResult`

Ermöglicht Testsuiten, Testfallergebnisse an IDT zu melden. Dieser Befehl muss am Ende jedes Testfalls in einer Testsuite aufgerufen werden.

Interaktion mit dem Gastgeber

Mit dem folgenden Befehl können Ihre Testsuiten mit dem Host-Computer kommunizieren.

`PollForNotifications`

Ermöglicht Testsuiten, nach Benachrichtigungen von IDT zu suchen.

`GetContextValue` und `GetContextString`

Ermöglicht Testsuiten das Abrufen von Werten aus dem IDT-Kontext. Weitere Informationen finden Sie unter Verwenden Sie den IDT-Kontext.

`ExecuteOnHost`

Ermöglicht Testsuiten die Ausführung von Befehlen auf dem lokalen Computer und ermöglicht IDT die Verwaltung des Lebenszyklus der ausführbaren Testfalldatei.

Die IDB CLI-Befehle

Der`run-suite` Befehl IDT CLI bietet verschiedene Optionen, mit denen Test Runner die Testausführung anpassen kann. Damit Testläufer diese Optionen verwenden können, um Ihre benutzerdefinierte Testsuite auszuführen, implementieren Sie die Unterstützung für die IDT-CLI. Wenn Sie die Unterstützung nicht implementieren, können Testläufer weiterhin Tests ausführen, aber einige CLI-Optionen funktionieren nicht richtig. Um ein optimales Kundenerlebnis zu bieten, empfehlen wir, die Unterstützung für die folgenden Argumente für den`run-suite` Befehl in der IDT-CLI zu implementieren:

`timeout-multiplier`

Gibt einen Wert größer als 1,0 an, der bei der Ausführung 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 Testrunner dieses Argument in seinem`run-suite` Befehl angibt, berechnet IDT damit den Wert der Umgebungsvariablen IDT_TEST_TIMEOUT und legt das`config.timeoutMultiplier` Feld im IDT-Kontext fest. Um dieses Argument zu untermauern, müssen Sie die folgenen Schritte ausführen:

- Anstatt den Timeout-Wert direkt aus der`test.json` Datei zu verwenden, lesen Sie die Umgebungsvariable IDT_TEST_TIMEOUT, um den korrekt berechneten Timeout-Wert zu erhalten.
- Rufen Sie den`config.timeoutMultiplier` Wert aus dem IDT-Kontext ab und wenden Sie ihn auf lange laufende Timeouts an.

Weitere Hinweise zum vorzeitigen Beenden aufgrund von Timeout-Ereignissen finden Sie unterGeben Sie das Ausgangsverhalten an.

`stop-on-first-failure`

Gibt an, dass IDT die Ausführung aller Tests beenden soll, wenn ein Fehler auftritt.

Wenn ein Testrunner dieses Argument in seinem`run-suite` Befehl angibt, stoppt IDT die Ausführung von Tests, sobald ein Fehler auftritt. Wenn Testfälle jedoch parallel laufen, kann dies zu unerwarteten Ergebnissen führen. Um die Unterstützung zu implementieren, stellen Sie sicher, dass Ihre Testlogik alle laufenden Testfälle anweist, anzuhalten, temporäre Ressourcen zu bereinigen und IDT ein Testergebnis zu melden, wenn IDT auf dieses Ereignis stößt. Weitere Informationen zum vorzeitigen Beenden finden Sie unterGeben Sie das Ausgangsverhalten an.

`group-id` und `test-id`

Legt fest, dass IDT nur die ausgewählten Testgruppen oder Testfälle ausführen soll.

Testläufer können diese Argumente zusammen mit ihrem`run-suite` Befehl verwenden, um das folgende Testausführungsverhalten anzugeben:

- 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 Testorchestrator für Ihre Testsuite einen bestimmten Satz von`RunTaskChoice` AND-Zuständen in Ihrem Testorchestrator enthalten. Wenn Sie keine benutzerdefinierte State-Machine verwenden, enthält der standardmäßige IDT-Testorchestrator die erforderlichen Zustände für Sie, und Sie müssen keine zusätzlichen Maßnahmen ergreifen. Wenn Sie jedoch einen benutzerdefinierten Testorchestrator verwenden, verwenden Sie ihnBeispiel für einen Zustandsautomaten: Ausführen von vom Benutzer ausgewählten Testgruppen als Beispiel, um die erforderlichen Zustände in Ihrem Testorchestrator hinzuzufügen.

Weitere Informationen zu EB CLI-Befehlen finden Sie unterDebuggen und Ausführen benutzerdefinierter Testsuiten.

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 Konsolennachrichten finden Sie unterNachrichtenformat.

Wenn das IDT die Ausführung der Testsuite abgeschlossen hat, sind diese Informationen auch in der`test_manager.log` Datei im*<devicetester-extract-location>*/`results/`*<execution-id>*`/logs` Ordner verfügbar.

Sie können jeden Testfall so konfigurieren, dass die Protokolle seines Testlaufs, einschließlich der Protokolle des zu testenden Geräts, in die*<group-id>*_*<test-id>* Datei im*<device-tester-extract-location>*`/results/`*execution-id*`/logs` Ordner geschrieben werden. Rufen Sie dazu den Pfad zur Protokolldatei mit der`testData.logFilePath` Abfrage aus dem IDT-Kontext ab, erstellen Sie eine Datei in diesem Pfad und schreiben Sie den gewünschten Inhalt hinein. IDT aktualisiert den Pfad automatisch auf der Grundlage des laufenden Testfalls. Wenn Sie sich dafür entscheiden, die Protokolldatei für einen Testfall nicht zu erstellen, wird keine Datei für diesen Testfall generiert.

Sie können Ihre ausführbare Textdatei auch einrichten, um nach Bedarf zusätzliche Protokolldateien im*<device-tester-extract-location>*/logs Ordner zu erstellen. Wir empfehlen, dass Sie eindeutige Präfixe für Protokolldateinamen angeben, damit Ihre Dateien nicht überschrieben werden.

Ergebnisse an IDT melden

IDT schreibt Testergebnisse in dieawsiotdevicetester_report.xml und die*suite-name*_report.xml Dateien. Diese Berichtsdateien befinden sich unter*<device-tester-extract-location>*/results/*<execution-id>*/. In beiden Berichten werden die Ergebnisse der Ausführung der Testsuite erfasst. Weitere Informationen zu den Schemas, die IDT für diese Berichte verwendet, finden Sie unterÜberprüfen Sie IDT-Testergebnisse und -protokolle

Um den Inhalt der*suite-name*_report.xml Datei aufzufüllen, müssen Sie denSendResult Befehl verwenden, um die Testergebnisse an IDT zu melden, bevor die Testausführung abgeschlossen ist. Wenn IDT die Ergebnisse eines Tests nicht finden kann, gibt es einen Fehler für den Testfall aus. Der folgende Python-Auszug zeigt die Befehle zum Senden eines Testergebnisses an IDT:

```
request-variable = SendResultRequest(TestResult(result))
client.send_result(request-variable)
```

Wenn Sie keine Ergebnisse über die API melden, sucht IDT im Ordner „Testartefacts“ nach Testergebnissen. Der Pfad zu diesem Ordner wird in dertestData.testArtifactsPath Datei im IDT-Kontext gespeichert. In diesem Ordner verwendet IDT die erste alphabetisch sortierte XML-Datei, die es als Testergebnis findet.

Wenn Ihre Testlogik JUnit-XML-Ergebnisse erzeugt, können Sie die Testergebnisse in eine XML-Datei im Ordner artefacts schreiben, um die Ergebnisse direkt an IDT weiterzuleiten, anstatt die Ergebnisse zu analysieren und sie dann über die API an IDT zu senden.

Wenn Sie diese Methode verwenden, stellen Sie sicher, dass Ihre Testlogik die Testergebnisse korrekt zusammenfasst, und formatieren Sie Ihre Ergebnisdatei im gleichen Format wie die*suite-name*_report.xml Datei. IDT führt keine Validierung der von Ihnen bereitgestellten Daten durch, mit den folgenden Ausnahmen:

- IDT ignoriert alle Eigenschaften destestsuites Tags. Stattdessen werden die Tag-Eigenschaften anhand anderer gemeldeter Testgruppenergebnisse berechnet.
- Darin muss mindestens eintestsuite Tag vorhanden seintestsuites.

Da IDT für alle Testfälle denselben Artefaktordner verwendet und keine Ergebnisdateien zwischen den Testläufen löscht, kann diese Methode auch zu fehlerhaften Berichten führen, wenn IDT die falsche Datei liest. Wir empfehlen, dass Sie für alle Testfälle denselben Namen für die generierte XML-Ergebnisdatei verwenden, um die Ergebnisse für jeden Testfall zu überschreiben und sicherzustellen, dass IDT die richtigen Ergebnisse zur Verwendung zur Verfügung hat. Sie können zwar einen gemischten Ansatz für die Berichterstattung in Ihrer Testsuite verwenden, d. h. für einige Testfälle eine XML-Ergebnisdatei verwenden und für andere die Ergebnisse über die API senden, aber wir empfehlen diesen Ansatz nicht.

Geben Sie das Ausgangsverhalten 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 anzuzeigen, dass ein Testfall nicht ausgeführt wurde oder ob die ausführbare Testfalldatei keine Ergebnisse an IDT übermitteln konnte. Wenn IDT einen Exit-Code ungleich Null empfängt, markiert dies, dass im Testfall ein Fehler aufgetreten ist, der die Ausführung verhindert hat.

IDT kann in den folgenden Ereignissen anfordern oder erwarten, dass die Ausführung eines Testfalls beendet wird, bevor er abgeschlossen ist. Verwenden Sie diese Informationen, um Ihre ausführbare Testfalldatei so zu konfigurieren, dass jedes dieser Ereignisse im Testfall erkannt wird:

Timeout (Zeitüberschreitung)

Tritt auf, wenn ein Testfall länger als der in der`test.json` Datei angegebene Timeout-Wert ausgeführt wird. Wenn der Testrunner das`timeout-multiplier` Argument verwendet hat, um einen Timeout-Multiplikator anzugeben, berechnet IDT den Timeout-Wert mit dem Multiplikator.

Verwenden Sie die Umgebungsvariable IDT_TEST_TIMEOUT, um dieses Ereignis zu erkennen. Wenn ein Test-Runner einen Test startet, setzt IDT den Wert der Umgebungsvariablen IDT_TEST_TIMEOUT auf den berechneten Timeout-Wert (in Sekunden) und übergibt die Variable an die ausführbare Testfalldatei. Sie können den Variablenwert lesen, um einen geeigneten Timer einzustellen.

Unterbrechen

Tritt auf, wenn der Testrunner IDT unterbricht. Zum Beispiel durch DrückenCtrl+C.

Da Terminals Signale an alle untergeordneten Prozesse weitergeben, 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` booleschen Werts in der`PollForNotifications` API-Antwort zu überprüfen. Wenn IDT ein Interruptsignal empfängt, setzt es den Wert des`CancellationRequested` booleschen Werts auf`true`.

Stoppen Sie beim ersten Ausfall

Tritt auf, wenn ein Testfall, der parallel zum aktuellen Testfall ausgeführt wird, fehlschlägt und der Testrunner das`stop-on-first-failure` Argument verwendet hat, um anzugeben, dass IDT anhalten soll, wenn ein Fehler auftritt.

Um dieses Ereignis zu erkennen, können Sie die API regelmäßig abfragen, um den Wert des`CancellationRequested` booleschen Werts in der`PollForNotifications` API-Antwort zu überprüfen. Wenn IDT auf einen Fehler stößt und so konfiguriert ist, dass es beim ersten Fehler stoppt, setzt es den Wert des`CancellationRequested` booleschen Werts auf`true`.

Wenn eines dieser Ereignisse eintritt, wartet IDT 5 Minuten, bis die Ausführung aller aktuell laufenden Testfälle abgeschlossen ist. Wenn nicht alle laufenden Testfälle innerhalb von 5 Minuten abgeschlossen sind, zwingt IDT jeden seiner Prozesse zum Stoppen. Wenn IDT vor dem Ende der Prozesse keine Testergebnisse erhalten hat, markiert es die Testfälle als abgelaufen. Als bewährte Methode sollten Sie sicherstellen, dass Ihre Testfälle die folgenden Aktionen ausführen, wenn sie auf eines der Ereignisse stoßen:

1. Beenden Sie die Ausführung der normalen Testlogik.
2. Bereinigen Sie alle temporären Ressourcen, wie z. B. Testartefakte, auf dem zu testenden Gerät.
3. Melden Sie IDT ein Testergebnis, z. B. einen Testfehler oder einen Fehler.
4. Aussteigen.

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. Beispielsweise wird 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.

Kontext-Schema

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 entsprichtdevicesArray definiert imresource.jsonfile. EACHdevices-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 demAWSCloud. Diese Informationen werden von derconfig.jsonfile.

`testData.logFilePath`

Der Pfad zu der Protokolldatei, in die der Testfall Protokollmeldungen schreibt. Die Test-Suite erstellt diese Datei, falls sie nicht vorhanden ist.

`userData`

Informationen, die der Testläufer imuserdata.jsonOrdneraus.

Zugriff auf Daten im Kontext

Sie können den Kontext mithilfe der JsonPath-Notation aus Ihren JSON-Dateien und aus Ihrer ausführbaren Textdatei mit dem`GetContextValue`und`GetContextString`APIs. 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 :`test_orchestrator.yaml`verwenden Sie`{{query}}`aus.

  Wenn Sie den veralteten Zustandscomputer verwenden, dann wird `instate_machine.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.

Operator	Description
`[? (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 Standard-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 zu verwenden, 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 auf dieRE2-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. 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 verwenden`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-Standard`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 in`test.json`und`suite.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 Test Runner

Um benutzerdefinierte Testsuiten auszuführen, müssen Test Runner ihre Einstellungen basierend auf der Testsuite konfigurieren, die sie ausführen möchten. Die Einstellungen werden basierend auf Konfigurationsdateivorlagen im *<device-tester-extract-location>*`/configs/` Ordner angegeben. Falls erforderlich, müssen Test Runner auch AWS Anmeldeinformationen einrichten, die IDT für die Verbindung mit der AWS Cloud verwendet.

Als Test-Writer müssen Sie diese Dateien konfigurieren, um Ihre Testsuite zu debuggen. Sie müssen Anweisungen zum Testen von Runnern bereitstellen, damit diese die folgenden Einstellungen nach Bedarf konfigurieren können, um Ihre Testsuiten auszuführen.

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).

Test Runner können diese Informationen mithilfe der folgenden `device.json` Vorlagendatei bereitstellen, die sich im *<device-tester-extract-location>*`/configs/` Ordner befindet.

```
[
    {
        "id": "<pool-id>",
```

```
        "sku": "<pool-sku>",
        "features": [
            {
                "name": "<feature-name>",
                "value": "<feature-value>",
                "configs": [
                    {
                        "name": "<config-name>",
                        "value": "<config-value>"
                    }
                ],
            }
        ],
        "devices": [
            {
                "id": "<device-id>",
                "connectivity": {
                    "protocol": "ssh | uart | docker",
                    // ssh
                    "ip": "<ip-address>",
                    "port": <port-number>,
                    "auth": {
                        "method": "pki | password",
                        "credentials": {
                            "user": "<user-name>",
                            // pki
                            "privKeyPath": "/path/to/private/key",

                            // password
                            "password": "<password>",
                        }
                    },

                    // uart
                    "serialPort": "<serial-port>",

                    // docker
                    "containerId": "<container-id>",
                    "containerUser": "<container-user-name>",
                }
            }
        ]
    }
```

```
]
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`id`

Eine benutzerdefinierte alphanumerische ID, die eine Sammlung von Geräten, den sogenannten Gerätepool, eindeutig identifiziert. Geräte, die zu einem Pool gehören, müssen über identische Hardware verfügen. Bei der Ausführung einer Reihe von Tests werden Geräte im Pool verwendet, um die Workload zu parallelisieren. Mehrere Geräte werden verwendet, um verschiedene Tests auszuführen.

`sku`

Ein alphanumerischer Wert, durch den das zu testende Gerät eindeutig identifiziert wird. Die SKU wird verwendet, um qualifizierte Geräte zu verfolgen.

> **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 Test-Runnern Informationen zu den Feature-Namen und -Werten bereitstellen, die in die `device.json` Datei aufgenommen werden sollen. Wenn Sie beispielsweise ein Gerät testen möchten, das als MQTT-Server für andere Geräte fungiert, können Sie Ihre Testlogik so konfigurieren, dass bestimmte unterstützte Stufen für ein Feature namens validiert werden`MQTT_QOS`. Test Runner geben diesen Feature-Namen an und setzen den Feature-Wert auf die von ihrem Gerät unterstützten QOS-Level. Sie können die bereitgestellten Informationen aus dem IDT-Kontext mit der `devicePool.features` Abfrage oder aus dem Testorchestrator-Kontext mit der `pool.features` Abfrage abrufen.

`features.name`

Der Name des Features.

`features.value`

Die unterstützten Feature-Werte.

`features.configs`

Konfigurationseinstellungen für die Funktion, falls erforderlich.

`features.config.name`

Der Name der Konfigurationseinstellung.

`features.config.value`

Die unterstützten Einstellungswerte.

`devices`

Ein Array von Geräten im Pool, die getestet werden sollen. Mindestens ein Gerät ist erforderlich.

`devices.id`

Eine benutzerdefinierte eindeutige Kennung für das zu testende Gerät.

`connectivity.protocol`

Das Kommunikationsprotokoll, das für die Kommunikation mit diesem Gerät verwendet wird. Jedes Gerät in einem Pool muss dasselbe Protokoll verwenden.

Derzeit werden nur die Werte `ssh` und `uart` für physische Geräte und `docker` für Docker-Container unterstützt.

`connectivity.ip`

Die IP-Adresse des zu testenden Geräts.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.port`

Optional. Die Portnummer, die für SSH-Verbindungen verwendet werden soll.

Der Standardwert ist 22.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.auth`

Authentifizierungsinformationen für die Verbindung.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.auth.method`

Die Authentifizierungsmethode, die für den Zugriff über ein bestimmtes Verbindungsprotokoll auf ein Gerät verwendet wird.

Unterstützte Werte sind:

- `pki`
- `password`

`connectivity.auth.credentials`

Die für die Authentifizierung verwendeten Anmeldeinformationen.

`connectivity.auth.credentials.password`

Das Passwort für die Anmeldung am Gerät wird überprüft.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `password` festgelegt ist.

`connectivity.auth.credentials.privKeyPath`

Der vollständige Pfad zum privaten Schlüssel, der für die Anmeldung bei dem zu testenden Gerät verwendet wird.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `pki` festgelegt ist.

`connectivity.auth.credentials.user`

Der Benutzername für die Anmeldung bei dem zu testenden Gerät.

`connectivity.serialPort`

Optional. Die serielle Schnittstelle, mit der das Gerät verbunden 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 `ssh` festgelegt ist.

`connectivity.containerUser`

Optional. Der Name des Benutzers, der innerhalb des Containers verwendet werden soll. Der Standardwert ist der im Dockerfile bereitgestellte Benutzer.

Der Standardwert ist 22.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

> **Note**
>
> Um zu überprüfen, ob Test Runner die falsche Geräteverbindung für einen Test konfigurieren, können Sie `pool.Devices[0].Connectivity.Protocol` aus dem Testorchestrator-Kontext abrufen und ihn mit dem erwarteten Wert in einem `-Choice`Zustand vergleichen. Wenn ein falsches Protokoll verwendet wird, drucken Sie eine Nachricht mit dem `LogMessage` Status und wechseln Sie in den `Fail` Status . Alternativ können Sie Fehlerbehandlungscode verwenden, um einen Testfehler für falsche Gerätetypen zu melden.

(Optional) Konfigurieren von userdata.json

Die `userdata.json` Datei enthält alle zusätzlichen Informationen, die von einer Testsuite benötigt werden, aber nicht in der `device.json` Datei angegeben sind. Das Format dieser Datei hängt von der `userdata_scheme.json` Datei ab, die in der Testsuite definiert ist. Wenn Sie ein Test-Writer sind, stellen Sie sicher, dass Sie diese Informationen Benutzern bereitstellen, die die von Ihnen geschriebenen Testsuiten ausführen.

(Optional) Konfigurieren von resource.json

Die `resource.json` Datei enthält Informationen zu allen Geräten, die als Ressourcengeräte verwendet werden. Ressourcengeräte sind Geräte, die erforderlich sind, um bestimmte Funktionen eines getesteten Geräts zu testen. 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 Test-Writer verwenden Sie die Datei test.json, um die Features des Ressourcengeräts zu definieren, die für einen Test erforderlich sind. Test-Runner verwenden dann die `-resource.json`Datei, um einen Pool von Ressourcengeräten bereitzustellen, die über die erforderlichen Funktionen verfügen. Stellen Sie sicher, dass Sie diese Informationen Benutzern bereitstellen, die die von Ihnen geschriebenen Testsuiten ausführen.

Test Runner können diese Informationen mithilfe der folgenden `resource.json` Vorlagendatei bereitstellen, die sich im *`<device-tester-extract-location>`*`/configs/` Ordner befindet.

```
[
    {
        "id": "<pool-id>",
        "features": [
            {
                "name": "<feature-name>",
                "version": "<feature-version>",
                "jobSlots": <job-slots>
            }
        ],
        "devices": [
            {
                "id": "<device-id>",
                "connectivity": {
                    "protocol": "ssh | uart | docker",
                    // ssh
                    "ip": "<ip-address>",
                    "port": <port-number>,
                    "auth": {
                        "method": "pki | password",
                        "credentials": {
                            "user": "<user-name>",
                            // pki
                            "privKeyPath": "/path/to/private/key",

                            // password
                            "password": "<password>",
                        }
                    },

                    // uart
                    "serialPort": "<serial-port>",

                    // docker
                    "containerId": "<container-id>",
                    "containerUser": "<container-user-name>",
                }
            }
        ]
    }
]
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`id`

Eine benutzerdefinierte alphanumerische ID, die eine Sammlung von Geräten, den sogenannten Gerätepool, eindeutig identifiziert. Geräte, die zu einem Pool gehören, müssen über identische Hardware verfügen. Bei der Ausführung einer Reihe von Tests werden Geräte im Pool verwendet, um die Workload zu parallelisieren. Mehrere Geräte werden verwendet, um verschiedene Tests auszuführen.

`features`

Optional. Ein Array, das die Funktionen enthält, die das Gerät unterstützt. Die in diesem Feld erforderlichen Informationen sind in den test.json-Dateien in der Testsuite definiert und bestimmen, welche Tests ausgeführt werden sollen und wie diese Tests ausgeführt werden sollen. Wenn die Testsuite keine Funktionen benötigt, ist dieses Feld nicht erforderlich.

`features.name`

Der Name des Features.

`features.version`

Die Feature-Version.

`features.jobSlots`

Einstellung, um anzugeben, wie viele Tests das Gerät gleichzeitig verwenden können. Der Standardwert ist 1.

`devices`

Ein Array von Geräten im Pool, die getestet werden sollen. Mindestens ein Gerät ist erforderlich.

`devices.id`

Eine benutzerdefinierte eindeutige Kennung für das zu testende Gerät.

`connectivity.protocol`

Das Kommunikationsprotokoll, das für die Kommunikation mit diesem Gerät verwendet wird. Jedes Gerät in einem Pool muss dasselbe Protokoll verwenden.

Derzeit werden nur die Werte `ssh` und `uart` für physische Geräte und `docker` für Docker-Container unterstützt.

`connectivity.ip`

Die IP-Adresse des zu testenden Geräts.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.port`

Optional. Die Portnummer, die für SSH-Verbindungen verwendet werden soll.

Der Standardwert ist 22.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.auth`

Authentifizierungsinformationen für die Verbindung.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.auth.method`

Die Authentifizierungsmethode, die für den Zugriff über ein bestimmtes Verbindungsprotokoll auf ein Gerät verwendet wird.

Unterstützte Werte sind:

- `pki`
- `password`

`connectivity.auth.credentials`

Die für die Authentifizierung verwendeten Anmeldeinformationen.

`connectivity.auth.credentials.password`

Das Passwort für die Anmeldung am Gerät wird überprüft.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `password` festgelegt ist.

`connectivity.auth.credentials.privKeyPath`

Der vollständige Pfad zum privaten Schlüssel, der für die Anmeldung bei dem zu testenden Gerät verwendet wird.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `pki` festgelegt ist.

`connectivity.auth.credentials.user`

Der Benutzername für die Anmeldung bei dem zu testenden Gerät.

`connectivity.serialPort`

Optional. Die serielle Schnittstelle, mit der das Gerät verbunden 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 `ssh` festgelegt ist.

`connectivity.containerUser`

Optional. Der Name des Benutzers, der innerhalb des Containers verwendet werden soll. Der Standardwert ist der im Dockerfile bereitgestellte Benutzer.

Der Standardwert ist 22.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

(Optional) Konfigurieren von config.json

Die `config.json` Datei enthält Konfigurationsinformationen für IDT. In der Regel müssen Test-Runner diese Datei nicht ändern, es sei denn, sie geben ihre AWS Benutzeranmeldeinformationen für IDT und optional für eine -AWSRegion an. Wenn AWS Anmeldeinformationen mit den erforderlichen Berechtigungen bereitgestellt werden, werden Nutzungsmetriken AWS IoT Device Tester erfasst und an übermitteltAWS. Dies ist eine Opt-In-Funktion und wird zur Verbesserung der IDT-Funktionalität verwendet. Weitere Informationen finden Sie unter IDT-Nutzungsmetriken.

Test Runner können ihre AWS Anmeldeinformationen auf eine der folgenden Arten konfigurieren:

- Anmeldeinformationsdatei

  IDT verwendet die gleiche Anmeldeinformationsdatei wie das AWS CLI. Weitere Informationen finden Sie unter Konfigurations- und Anmeldeinformationsdateien.

  Der Speicherort der Datei mit den Anmeldeinformationen variiert je nach verwendetem Betriebssystem:

  - macOS Linux: `~/.aws/credentials`
  - Windows: `C:\Users\UserName\.aws\credentials`
- Umgebungsvariablen

Umgebungsvariablen sind Variablen, die vom Betriebssystem gepflegt und von Systembefehlen verwendet werden. Variablen, die während einer SSH-Sitzung definiert wurden, sind nicht verfügbar, nachdem diese Sitzung geschlossen wurde. IDT kann die `AWS_SECRET_ACCESS_KEY` Umgebungsvariablen `AWS_ACCESS_KEY_ID` und verwenden, um AWS Anmeldeinformationen zu speichern

Um diese Variablen auf Linux, macOS oder Unix festzulegen, verwenden Sie export:

```
export AWS_ACCESS_KEY_ID=<your_access_key_id>
export AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

In Windows können Sie die Variablen mit set festlegen:

```
set AWS_ACCESS_KEY_ID=<your_access_key_id>
set AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

Um AWS Anmeldeinformationen für IDT zu konfigurieren, bearbeiten Test-Runner den `-auth`Abschnitt in der `config.json` Datei im *<device-tester-extract-location>*`/configs/` Ordner .

```
{
    "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>* definiert.

`log.location`

Der Pfad zum Protokollordner im *<device-tester-extract-location>*.

`configFiles.root`

Der Pfad zu dem Ordner, der die Konfigurationsdateien enthält.

`configFiles.device`

Der Pfad zur `device.json` Datei.

`testPath`

Der Pfad zu dem Ordner, der Testsuiten enthält.

`reportPath`

Der Pfad zu dem Ordner, der die Testergebnisse enthält, nachdem IDT eine Testsuite ausgeführt hat.

`awsRegion`

Optional. Die AWS Region, die Testsuiten verwenden werden. Wenn nicht festgelegt, verwendet Testsuiten die Standardregion, die in jeder Testsuite angegeben ist.

`auth.method`

Die Methoden-IDT verwendet , um AWS Anmeldeinformationen abzurufen. Unterstützte Werte sind `file` das Abrufen von Anmeldeinformationen aus einer Anmeldeinformationsdatei und `environment` das Abrufen von Anmeldeinformationen mithilfe von Umgebungsvariablen.

`auth.credentials.profile`

Das Anmeldeinformationsprofil, das aus der Anmeldeinformationsdatei verwendet werden soll. Diese Eigenschaft gilt nur, wenn `auth.method` auf `file` festgelegt ist.

Debuggen und Ausführen benutzerdefinierter Testsuiten

Nachdem die erforderliche Konfiguration festgelegt wurde, kann IDT Ihre Testsuite ausführen. Die Laufzeit der vollständigen Testsuite hängt von der Hardware und der Zusammensetzung der Testsuite ab. Als Referenz dauert es etwa 30 Minuten, bis die vollständige AWS IoT Greengrass Qualifikationstestsuite auf einem Raspberry Pi 3B abgeschlossen ist.

Während Sie Ihre Testsuite schreiben, können Sie IDT verwenden, um die Testsuite im Debug-Modus auszuführen, um Ihren Code zu überprüfen, bevor Sie ihn ausführen, oder ihn für Test Runner bereitstellen.

IDT im Debug-Modus ausführen

Da Testsuiten von IDT abhängen, um mit Geräten zu interagieren, den Kontext bereitzustellen und Ergebnisse zu erhalten, können Sie Ihre Testsuiten nicht einfach ohne IDT-Interaktion in einer IDE debuggen. Dazu stellt die IDT-CLI den `debug-test-suite` Befehl bereit, mit dem Sie IDT im Debug-Modus ausführen können. Führen Sie den folgenden Befehl aus, um die verfügbaren Optionen für 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 oder führt den Testorchestrator aus. Stattdessen interagiert es mit Ihrer IDE, um auf Anfragen zu antworten, die von der Testsuite gestellt werden, die in der IDE ausgeführt wird, und druckt die Protokolle in der Konsole. IDT führt keine Zeitüberschreitung durch und wartet auf das Beenden, bis es manuell unterbrochen wird. Im Debug-Modus führt IDT auch den Testorchestrator 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`.
- Argumente zur Auswahl von Ressourcengeräten. IDT liest diese Informationen nicht aus `test.json`.

Führen Sie die folgenden Schritte aus, um Ihre Testsuiten zu debuggen:

1. Erstellen Sie die Einstellungskonfigurationsdateien, die zum Ausführen der Testsuite erforderlich sind. Wenn Ihre Testsuite beispielsweise die `device.json`, und erfordert`user data.json`, stellen Sie sicher`resource.json`, dass Sie alle nach Bedarf konfigurieren.
2. Führen Sie den folgenden Befehl aus, um IDT in den Debug-Modus zu versetzen, und wählen Sie alle Geräte aus, 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 die - `run` oder -`debug`Konfiguration, um Folgendes zu tun:
   a. Legen Sie die Werte der IDT-generierten Umgebungsvariablen fest.
   b. Legen Sie den Wert aller Umgebungsvariablen oder Argumente fest, die Sie in Ihrer -`test.json` und -`suite.json`Datei angegeben haben.
   c. Legen Sie nach Bedarf Breakpoints 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 Debug-Modus keine Zeitüberschreitung durch.
5. Nachdem Sie das Debuggen abgeschlossen haben, unterbrechen Sie IDT, um den Debugging-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 die unterstützten Produkte für Ihre Version von IDT, in diesem Fall AWS IoT Greengrass Versionen und Versionen der AWS IoT Greengrass Qualifizierungstestsuite auf, die für die aktuelle IDT-Version verfügbar sind.

`list-test-cases`

Listet die Testfälle in einer bestimmten Testgruppe auf. Die folgende Option wird unterstützt:

- `group-id`. Die Testgruppe, nach der gesucht werden soll. Diese Option ist erforderlich und muss eine einzelne Gruppe angeben.

`run-suite`

Führt eine Reihe von Tests in einem Pool von Geräten aus. Im Folgenden sind einige häufig verwendete Optionen aufgeführt:

- `suite-id`. Die auszuführende Testsuite-Version. Wenn nicht angegeben, verwendet IDT die neueste Version im `tests`-Ordner.
- `group-id`. Die auszuführenden Testgruppen als kommagetrennte Liste. Bei fehlender Angabe führt IDT alle Testgruppen in der Testsuite aus.
- `test-id`. Die auszuführenden Testfälle als kommagetrennte Liste. Wenn angegeben, muss `group-id` eine einzelne Gruppe angeben.
- `pool-id`. Der zu testende Gerätepool. Test Runner müssen einen Pool angeben, wenn sie mehrere Gerätepools in Ihrer `device.json` Datei definiert haben.
- `timeout-multiplier`. Konfiguriert IDT, um das in der `-test.json`Datei angegebene Timeout für die Testausführung mit einem benutzerdefinierten Multiplikator zu ändern.
- `stop-on-first-failure`. Konfiguriert IDT so, dass die Ausführung beim ersten Fehler gestoppt wird. Diese Option sollte mit `group-id` verwendet werden, um die angegebenen Testgruppen zu debuggen.
- `userdata`. Legt die Datei fest, die Benutzerdaten enthält, die zum Ausführen der Testsuite erforderlich sind. Dies ist nur erforderlich, wenn in der `-suite.json`Datei für die Testsuite auf „true“ gesetzt `userdataRequired` ist.

Weitere Informationen zu `run-suite`-Optionen erhalten Sie mit der `help`-Option:

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

`debug-test-suite`

Führen Sie die Testsuite im Debug-Modus aus. Weitere Informationen finden Sie unter IDT im Debug-Modus ausführen.

Überprüfen Sie IDT-Testergebnisse und -protokolle

In diesem Abschnitt wird das Format beschrieben, in dem IDT Konsolenprotokolle und Testberichte generiert.

Nachrichtenformat

AWS IoT Device Testerverwendet 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`

Die 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 Test-Orchestrator. 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 IoT Device TesterBerichts-Schema

`awsiotdevicetester_report.xml`list ein signierter 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.
- Die GerätesKU- und der 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 beidenawsiotdevicetester_report.xmlodersuite-name_report.xmllistet 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 zum Ausführen 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, AWS IoT Device Tester sammelt und sendet Nutzungsmetriken 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 AWS CLI IDT-Befehle, die zum Ausführen von Tests verwendet werden
- Die Testsuiten, die ausgeführt werden
- 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.json` Datei nicht für den Zugriff darauf.

Konfigurieren Sie Ihre AWS Anmeldeinformationen

Wenn Sie noch keine haben AWS-Konto, müssen Sie eine erstellen. Wenn Sie bereits eine haben 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 einer AWS-Konto

In diesem Schritt erstellen und konfigurieren Sie eine AWS-Konto. Wenn Sie bereits über ein AWS-Konto verfügen, fahren Sie direkt mit the section called “Schritt 2: Konfigurieren von Berechtigungen für IDT” fort.

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.

Wählen Sie zum Erstellen eines Administratorbenutzers eine der folgenden Optionen aus.

Wählen Sie eine Möglichkeit zur Verwaltung Ihres Administrators aus.	Bis	Von	Sie können auch
Im IAM Identity Center (Empfohlen)	Verwendung von kurzfristigen Anmeldeinformationen für den Zugriff auf AWS. Dies steht im Einklang mit den bewährten Methoden für die Sicherheit. Weitere Informationen zu bewährten Methoden finden Sie unter Bewährte Methoden für die Sicherheit in IAM im IAM-Benutzerhandbuch.	Beachtung der Anweisungen unter Erste Schritte im AWS IAM Identity Center - Benutzerhandbuch.	Konfigurieren Sie den programmatischen Zugriff, indem Sie den AWS IAM Identity Center im AWS Command Line Interface Benutzerhandbuch AWS CLI zu verwendenden konfigurieren.
In IAM (Nicht empfohlen)	Verwendung von langfristigen Anmeldeinformationen für den Zugriff auf AWS.	Beachtung der Anweisungen unter Erstellen Ihres ersten IAM-Administratorbenutzers und Ihrer ersten Benutzergruppe im IAM-Benutzerhandbuch.	Programmgesteuerten Zugriff unter Verwendung der Informationen unter Verwalten der Zugriffsschlüssel für IAM-Benutzer im IAM-Benutzerhandbuch konfigurieren.

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 Richtlinie prüfen.

   d. Geben Sie unter Name **`IDTUsageMetricsIAMPermissions`** ein. Überprüfen Sie unter Summary (Zusammenfassung) die von Ihrer Richtlinie gewährten Berechtigungen.

e. 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 Set permissions (Berechtigungen einrichten) die Option Attach existing policies to user directly (Vorhandene Richtlinien dem Benutzer direkt anfügen) 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.

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 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.

PowerShell

```
aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-
document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot-device-tester:SendMetrics"
            ],
            "Resource": "*"
        }
    ]
}'
```

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 die folgenden Befehle aus, um Umgebungsvariablen zu verwenden.

      Linux or Unix

      ```
      export AWS_ACCESS_KEY_ID=access-key
      export AWS_SECRET_ACCESS_KEY=secret-access-key
      ```

      Windows Command Prompt (CMD)

      ```
      set AWS_ACCESS_KEY_ID=access-key
      set AWS_SECRET_ACCESS_KEY=secret-access-key
      ```

      PowerShell

      ```
      $env:AWS_ACCESS_KEY_ID="access-key"
      $env:AWS_SECRET_ACCESS_KEY="secret-access-key"
      ```

   b. Um die Datei mit den Anmeldeinformationen zu verwenden, fügen Sie der `~/.aws/credentials` Datei die folgenden Informationen hinzu.

      ```
      [profile-name]
      aws_access_key_id=access-key
      aws_secret_access_key=secret-access-key
      ```

2. Konfigurieren Sie den `auth` Abschnitt der `config.json` Datei. Weitere Informationen finden Sie unter (Optional) Konfigurieren von config.json.

Fehlerbehebung IDT fürAWS IoT GreengrassV2

IDT fürAWS IoT GreengrassV2 schreibt Fehler je nach Art der Fehler an verschiedene Speicherorte. IDT schreibt Fehler in die Konsole, in Protokolldateien und in Testberichte.

Wo kann man nach Fehlern suchen

Allgemeine Fehler werden auf der Konsole angezeigt, während der Test ausgeführt wird, und eine Zusammenfassung der fehlgeschlagenen Tests wird angezeigt, wenn alle Tests abgeschlossen sind.`awsiotdevicetester_report.xml`enthält eine Zusammenfassung aller Fehler, die zum Fehlschlagen eines Tests geführt haben. IDT speichert die Protokolldateien für jeden Testlauf in einem Verzeichnis mit einer UUID für die Testausführung, die während des Testlaufs auf der Konsole angezeigt wird.

Das IDT-Testprotokollverzeichnis ist*<device-tester-extract-location>*/`results/`*<execution-id>*`/logs/`. Dieses Verzeichnis enthält die folgenden Dateien, die in der Tabelle angezeigt werden. Dies ist beim Debuggen nützlich.

Datei	Beschreibung
`test_manager.log`	Die Protokolle, die während der Ausführun g des Tests auf die Konsole geschrieben wurden. Die Zusammenfassung der Ergebniss e am Ende dieser Datei enthält eine Liste der fehlgeschlagenen Tests. Die Warn- und Fehlerprotokolle in dieser Datei können Ihnen Informationen über den/die Fehler bereitstellen.
test-group-id /*test-case-id* /*test-name* `.log`	Detaillierte Protokolle für den spezifischen Test in einer Testgruppe. Bei Tests, bei denen Greengrass-Komponenten eingesetzt werden, wird die Testfall-Protokolldatei aufgerufe `ngreengrass-test-run.log` .
test-group-id /*test-case-id* / `greengrass.log`	Detaillierte Protokolle fürAWS IoT Greengras sKernsoftware. IDT kopiert diese Datei von dem zu testenden Gerät, wenn es Tests ausführt, bei denen Folgendes installiert wirdAWS IoT GreengrassKernsoftware auf dem Gerät. Weitere Informationen zu den Meldungen in dieser Protokolldatei finden Sie

Datei	Beschreibung
	unterProblembehebung AWS IoT Greengrass V2.
test-group-id /*test-case-id*/*component-name* .log	Detaillierte Protokolle für Greengrass-Komponenten, die während Testläufen eingesetzt werden. IDT kopiert Komponenten-Protokolldateien vom zu testenden Gerät, wenn Tests ausgeführt werden, bei denen bestimmte Komponenten bereitgestellt werden. Der Name jeder Komponenten-Protokolldatei entspricht dem Namen der bereitgestellten Komponente. Weitere Hinweise zu den Meldungen in dieser Protokolldatei finden Sie unterProblembehebung AWS IoT Greengrass V2.

IDT auflösen fürAWS IoT GreengrassV2-Fehler

Bevor Sie IDT ausführen fürAWS IoT Greengrass, richten Sie die richtigen Konfigurationsdateien ein. Wenn Sie Parsing- und Konfigurationsfehler erhalten, besteht Ihr erster Schritt darin, eine für Ihre Umgebung geeignete Konfigurationsvorlage zu finden und zu verwenden.

Wenn weiterhin Probleme auftreten, beachten Sie den folgenden Debugging-Vorgang.

Themen

- Fehler bei der Alias-Auflösung
- Konfliktfehler
- Fehler aufgrund eines nicht startenden Tests
- Das Docker-Qualifikationsbild enthält Fehler
- Anmeldeinformationen konnten nicht gelesen werden
- Guide-Fehler mit PreInstalled Grünes Gras
- Ungültige Signaturausnahme
- Qualifizierungsfehler beim maschinellen Lernen
- Fehlgeschlagene Bereitstellungen im Open Test Framework (OTF)

- Parsing-Fehler
- Fehler bei abgelehnter Berechtigung
- Fehler beim Generieren des Qualifizierungsberichts
- Fehler aufgrund fehlender erforderlicher Parameter
- Sicherheitsausnahme auf macOS
- SSH-Verbindungsfehler
- Qualifizierungsfehler im Stream Manager
- Timeout-Fehler
- Fehler bei der Versionsprüfung

Fehler bei der Alias-Auflösung

Wenn Sie benutzerdefinierte Testsuiten ausführen, wird möglicherweise der folgende Fehler in der Konsole und im`test_manager.log`.

```
Couldn't resolve placeholders: couldn't do a json lookup: index out of range
```

Dieser Fehler kann auftreten, wenn die im IDT Test Orchestrator konfigurierten Aliase nicht korrekt aufgelöst werden oder wenn die aufgelösten Werte nicht in den Konfigurationsdateien vorhanden sind. Um diesen Fehler zu beheben, stellen Sie sicher, dass`device.json`und`userdata.json` enthalten die richtigen Informationen, die für Ihre Testsuite erforderlich sind. Für Informationen über die Konfiguration, die erforderlich ist fürAWS IoT GreengrassQualifikation finden Sie unterKonfigurieren Sie die IDT-Einstellungen, um die AWS IoT Greengrass Qualification Suite auszuführen.

Konfliktfehler

Möglicherweise wird der folgende Fehler angezeigt, wenn Sie denAWS IoT GreengrassQualification Suite gleichzeitig auf mehr als einem Gerät.

```
ConflictException: Component [com.example.IDTHelloWorld : 1.0.0] for account [account-
id] already exists with state: [DEPLOYABLE] { RespMetadata: { StatusCode: 409,
 RequestID: “id” }, Message_: “Component [com.example.IDTHelloWorld : 1.0.0] for
 account [account-id] already exists with state: [DEPLOYABLE]” }
```

Die gleichzeitige Testausführung wird noch nicht unterstützt fürAWS IoT GreengrassQualifizierungssuite. Führen Sie die Qualification Suite nacheinander für jedes Gerät aus.

Fehler aufgrund eines nicht startenden Tests

Möglicherweise treten Fehler auf, die auf Fehler hinweisen, die beim Versuch, den Test zu starten, aufgetreten sind. Es gibt mehrere mögliche Ursachen. Gehen Sie daher wie folgt vor:

- Stellen Sie sicher, dass der Poolname in Ihrem Ausführungsbefehl tatsächlich existiert. IDT referenziert den Poolnamen direkt von Ihrem`device.json`datei.
- Stellen Sie sicher, dass die Geräte in Ihrem Pool über die richtigen Konfigurationsparameter verfügen.

Das Docker-Qualifikationsbild enthält Fehler

Die Qualifizierungstests für Docker Application Manager verwenden den`amazon/amazon-ec2-metadata-mock`Container-Image in Amazon ECR, um das zu testende Gerät zu qualifizieren.

Möglicherweise wird die folgende Fehlermeldung angezeigt, wenn das Image bereits in einem Docker-Container auf dem zu testenden Gerät vorhanden ist.

```
The Docker image amazon/amazon-ec2-metadata-mock:version already exists on the device.
```

Wenn Sie dieses Image zuvor heruntergeladen und ausgeführt haben`amazon/amazon-ec2-metadata-mock`Container auf Ihrem Gerät, stellen Sie sicher, dass Sie dieses Image von dem zu testenden Gerät entfernen, bevor Sie die Qualifikationstests ausführen.

Anmeldeinformationen konnten nicht gelesen werden

Beim Testen von Windows-Geräten stoßen Sie möglicherweise auf`Failed to read credential`Fehler in der`greengrass.log`Datei, wenn der Benutzer, mit dem Sie eine Verbindung zu dem zu testenden Gerät herstellen, nicht im Anmeldeinformationsmanager auf diesem Gerät eingerichtet ist.

Um diesen Fehler zu beheben, konfigurieren Sie den Benutzer und das Passwort für den IDT-Benutzer im Anmeldeinformationsmanager auf dem zu testenden Gerät.

Weitere Informationen finden Sie unter Konfigurieren Sie Benutzeranmeldeinformationen für Windows-Geräte.

Guide-Fehler mit PreInstalled Grünes Gras

Beim Ausführen von IDT mit PreInstalled Greengrass, wenn Sie auf einen Fehler stoßen von`Guice`oder`ErrorInCustomProvider`, überprüfen Sie, ob die Datei`userdata.json`hat die`InstalledDirRootOnDevice`auf den Greengrass-Installationsordner eingestellt. IDT sucht nach der Datei`effectiveConfig.yaml`unter`<InstallationDirRootOnDevice>/config/effectiveConfig.yaml`.

Weitere Informationen finden Sie unter Konfigurieren Sie Benutzeranmeldeinformationen für Windows-Geräte.

Ungültige Signaturausnahme

Wenn Sie Lambda-Qualifizierungstests ausführen, stoßen Sie möglicherweise auf`invalidsignatureexception`Fehler, wenn auf Ihrem IDT-Hostcomputer Probleme mit dem Netzwerkzugriff auftreten. Setzen Sie Ihren Router zurück und führen Sie die Tests erneut durch.

Qualifizierungsfehler beim maschinellen Lernen

Wenn Sie Qualifizierungstests für maschinelles Lernen (ML) durchführen, kann es zu Qualifikationsfehlern kommen, wenn Ihr Gerät die Anforderungen nicht erfüllt Anforderungen zur Bereitstellung der AWS-bereitgestellte ML-Komponenten. Gehen Sie wie folgt vor, um Fehler bei der ML-Qualifizierung zu beheben:

- Suchen Sie in den Komponentenprotokollen nach Fehlerdetails für die Komponenten, die während des Testlaufs bereitgestellt wurden. Die Komponentenprotokolle befinden sich im *<device-tester-extract-location>*`/results/`*<execution-id>*`/logs/`*<test-group-id>*Verzeichnis.
- Füge das hinzu`-Dgg.persist=installed.software`Argument zum`test.json`Datei für den fehlgeschlagenen Testfall. Die`test.json`Die Datei befindet sich im *<device-tester-extract-location>*`/tests/GGV2Q_`*version* `directory.`

Fehlgeschlagene Bereitstellungen im Open Test Framework (OTF)

Wenn die Bereitstellung durch OTF-Tests nicht abgeschlossen werden kann, liegt dies wahrscheinlich an den für den übergeordneten Ordner von festgelegten Berechtigungen`TempResourcesDirOnDevice`und`InstallationDirRootOnDevice`. Führen Sie den folgenden Befehl aus, um die Berechtigungen für diesen Ordner korrekt festzulegen. Ersetzen *folder-name* durch den Namen des übergeordneten Ordners.

```
sudo chmod 755 folder-name
```

Parsing-Fehler

Tippfehler in einer JSON-Konfiguration können zu Analysefehlern führen. In den meisten Fällen sind die Ursache des Problems ausgelassene Klammern, Kommas oder Anführungszeichen in Ihrer JSON-Datei. IDT führt eine JSON-Validierung durch und druckt Debugging-Informationen. Gedruckt werden die Zeile, in der der Fehler aufgetreten ist, sowie Zeilennummer und Spaltennummer des Syntaxfehlers. Diese Informationen sollten ausreichen, um Ihnen bei der Behebung des Fehlers zu helfen. Wenn Sie den Fehler jedoch immer noch nicht finden können, können Sie die Validierung manuell in Ihrer IDE, einem Texteditor wie Atom oder Sublime oder über ein Online-Tool wie JsonLint durchführen.

Fehler 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.

Fehler beim Generieren des Qualifizierungsberichts

IDT unterstützt die vier neuesten*major*.*minor*Versionen vonAWS IoT GreengrassV2 Qualification Suite (GGV2Q) zur Erstellung von Qualifikationsberichten, die Sie einreichen könnenAWS Partner Networkum Ihre Geräte in die aufzunehmenAWS PartnerGerätekatalog. Frühere Versionen der Qualification Suite generieren keine Qualifikationsberichte.

Wenn Sie Fragen zu den Support-Richtlinien haben, wenden Sie sich anAWS Support.

Fehler aufgrund fehlender erforderlicher Parameter

Wenn IDT neue Funktionen hinzufügt, kann dies zu Änderungen an den Konfigurationsdateien führen. 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 validiert auch Ihre JSON-Konfigurationsdateischemas, um sicherzustellen, dass Sie die neueste unterstützte Version verwenden.

Sicherheitsausnahme auf macOS

Wenn Sie IDT auf einem macOS-Hostcomputer ausführen, wird die Ausführung von IDT blockiert. Um IDT auszuführen, gewähren Sie den ausführbaren Dateien eine Sicherheitsausnahme, die Teil der IDT-Laufzeitfunktionalität ist. Wenn die Warnmeldung auf Ihrem Host-Computer angezeigt wird, gehen Sie für jede der entsprechenden ausführbaren Dateien wie folgt vor:

Um ausführbaren IDT-Dateien eine Sicherheitsausnahme zu gewähren

1. Öffnen Sie auf dem macOS-Computer im Apple-MenüSystemeinstellungen.
2. WähleSicherheit und Datenschutz, dann auf derAllgemeinWählen Sie auf der Registerkarte das Schlosssymbol, um Änderungen an den Sicherheitseinstellungen vorzunehmen.
3. Im Falle einer Blockierung`devicetester_mac_x86-64`, suche nach der Nachricht`"devicetester_mac_x86-64" was blocked from use because it is not from an identified developer.`und wähleTrotzdem zulassen.
4. Setzen Sie die IDT-Tests fort, bis Sie alle beteiligten ausführbaren Dateien abgeschlossen haben.

SSH-Verbindungsfehler

Wenn IDT keine Verbindung zu einem zu testenden Gerät herstellen kann, werden Verbindungsfehler protokolliert`/results/<execution-id>/logs/<test-case-id>.log`. SSH-Nachrichten werden oben in dieser Protokolldatei angezeigt, da die Verbindung zu einem zu testenden Gerät eine der ersten Operationen ist, die IDT ausführt.

Die meisten Windows-Konfigurationen verwenden die PuTTY-Terminalanwendung, um eine Verbindung zu Linux-Hosts herzustellen. Für diese Anwendung müssen Sie standardmäßige private PEM-Schlüsseldateien in ein proprietäres Windows-Format namens PPK konvertieren. Wenn Sie SSH in Ihrem konfigurierende`device.json`Datei, verwenden Sie PEM-Dateien. Wenn Sie eine PPK-Datei verwenden, kann IDT keine SSH-Verbindung mit dem herstellenAWS IoT GreengrassGerät und kann keine Tests ausführen.

Wenn Sie ab IDT v4.4.0 SFTP auf Ihrem zu testenden Gerät nicht aktiviert haben, wird möglicherweise der folgende Fehler in der Protokolldatei angezeigt.

```
SSH connection failed with EOF
```

Um diesen Fehler zu beheben, aktivieren Sie SFTP auf Ihrem Gerät.

Qualifizierungsfehler im Stream Manager

Wenn Sie Stream Manager-Qualifizierungstests ausführen, wird möglicherweise der folgende Fehler in der`com.aws.StreamManagerExport.log`datei.

```
Failed to upload data to S3
```

Dieser Fehler kann auftreten, wenn der Stream-Manager denAWSAnmeldeinformationen in der`~/root/.aws/credentials`Datei auf Ihrem Gerät, anstatt die Umgebungsanmeldedaten zu verwenden, die IDT auf das zu testende Gerät exportiert. Um dieses Problem zu vermeiden, löschen Sie die`credentials`Datei auf Ihrem Gerät und führen Sie den Qualifikationstest erneut aus.

Timeout-Fehler

Sie können das Timeout für jeden Test erhöhen, indem Sie einen Timeout-Multiplikator angeben, der auf den Standardwert des Timeouts jedes 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`. Beispiele:

```
./devicetester_linux run-suite --suite-id GGV2Q_1.0.0 --pool-id DevicePool1 --timeout-multiplier 2.5
```

Führen Sie `run-suite --help` aus, um weitere Informationen zu erhalten.

Einige Timeout-Fehler treten auf, wenn IDT-Testfälle aufgrund von Konfigurationsproblemen nicht abgeschlossen werden können. Sie können diese Fehler nicht beheben, indem Sie den Timeout-Multiplikator erhöhen. Verwenden Sie die Protokolle des Testlaufs, um die zugrunde liegenden Konfigurationsprobleme zu beheben.

- Wenn die MQTT- oder Lambda-Komponentenprotokolle Folgendes enthalten`Access denied`Fehler, Ihr Greengrass-Installationsordner hat möglicherweise nicht die richtigen Dateiberechtigungen. Führen Sie den folgenden Befehl für jeden Ordner im Installationspfad aus, den Sie in Ihrem`userdata.json`datei.

  ```
  sudo chmod 755 folder-name
  ```

- Wenn die Greengrass-Protokolle darauf hinweisen, dass die Greengrass-CLI-Bereitstellung nicht abgeschlossen ist, gehen Sie wie folgt vor:
  - Stellen Sie sicher, dass`bash`ist auf dem zu testenden Gerät installiert.
  - Wenn dein`userdata.json`Datei enthält die`GreengrassCliVersion`Konfigurationsparameter, entferne ihn. Dieser Parameter ist in IDT v4.1.0 und späteren Versionen veraltet. Weitere Informationen finden Sie unter Konfigurieren Sie userdata.json.
- Wenn der Lambda-Bereitstellungstest mit der Fehlermeldung „Validating Lambda publish: time out“ fehlschlug und Sie eine Fehlermeldung in der Testprotokolldatei erhalten (`idt-gg2-lambda-function-idt-<resource-id>.log`) das heißt`Error: Could not find or load main class com.amazonaws.greengrass.runtime.LambdaRuntime.`, mach das Folgende:
  - Überprüfen Sie, wofür der Ordner verwendet wurde`InstallationDirRootOnDevice`in der`userdata.json`datei.
  - Stellen Sie sicher, dass die richtigen Benutzerberechtigungen auf Ihrem Gerät eingerichtet sind. Weitere Einzelheiten finden Sie unterKonfigurieren Sie Benutzerberechtigungen auf Ihrem Gerät.

Fehler bei der Versionsprüfung

IDT gibt den folgenden Fehler aus, wennAWSDie Benutzeranmeldedaten für den IDT-Benutzer verfügen nicht über die erforderlichen IAM-Berechtigungen.

```
Failed to check version compatibility
```

DieAWSBenutzer, der nicht über die erforderlichen IAM-Berechtigungen verfügt.

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

AWS IoT Device Tester für AWS IoT Greengrass ist ein Testautomatisierungstool, das zur Validierung und Qualifizierung Ihrer AWS IoT Greengrass Geräte für die Aufnahme in den AWS Partner Gerätekatalog verwendet wird. Wir empfehlen Ihnen, die neueste Version von AWS IoT Greengrass und zu verwenden, AWS IoT Device Tester um Ihre Geräte zu testen oder zu qualifizieren.

Für jede unterstützte Version von AWS IoT Device Tester ist mindestens eine Version von verfügbar AWS IoT Greengrass. Unterstützte Versionen von AWS IoT Greengrass finden Sie unter Greengrass-Kernversionen. Informationen zu unterstützten Versionen von AWS IoT Device Tester finden Sie unter Unterstützte Versionen von AWS IoT Device Tester für AWS IoT Greengrass V2.

Sie können auch eine der unterstützten Versionen von AWS IoT Greengrass und verwenden AWS IoT Device Tester , um Ihre Geräte zu testen oder zu qualifizieren. Obwohl Sie weiterhin nicht unterstützte Versionen von verwenden können AWS IoT Device Tester, erhalten diese Versionen keine Fehlerbehebungen oder Updates. Wenn Sie Fragen zur Support-Richtlinie haben, wenden Sie sich an AWS Support.

Greengrass-basierte IoT-Lösungen

Everyware von Eurotech GreenEdge befindet sich in einer Vorschauversion für AWS IoT Greengrass und kann sich ändern. Diese Lösung wird vom AWS nicht unterstützt. Bei Problemen mit diesem Gerät müssen Sie sich an Eurotech wenden.

AWS IoT Greengrass bietet Lösungen von -Partnern, um Ihre Erfahrung bei der Installation von Greengrass zu optimieren. Im Folgenden finden Sie eine Lösung, die in Zusammenarbeit mit Eurotech angeboten AWS wurde. Diese Lösung wird mit vorinstallierter AWS IoT Greengrass Core-Edge-Laufzeit und zusätzlichen Funktionen geliefert.

Eurotech

AWS hat in Zusammenarbeit mit Eurotech eine IoT-Lösung für Kunden angeboten, die nach einem Gerät suchen, auf dem die -AWS IoT GreengrassCore-Software vorinstalliert ist. Everyware von Eurotech GreenEdge ist eine IoT-Edge-Software, die von vorkonfiguriert und vorkonfiguriert istAWS. Diese Lösung führt die Funktionen von Greengrass und dem Eurotech Everyware Software Framework (ESF) zusammen, um Kunden umfangreiche, universelle Konnektivität über Protokolladapter zu bieten, z. B.: Bol, OPC-UA Client/Server, S7, TwinCAT ,J1939, DNP3 Master/Outstation und mehr. Mit dieser Lösung können Sie auch Daten an die senden AWS Cloud und eine Verbindung zu allen nordgebundenen AWS Services (wie AWS IoT Core, AWS IoT SiteWise, , AWS IoT Analytics Amazon S3 und Amazon Kinesis Video Streams) herstellen. In Kombination mit Everyware Cloud, der Geräteverwaltungslösung von Eurotech, führt diese Lösung einen neuen Zero-Speed-Bereitstellungsservice ein, der das Onboarding und die Massenbereitstellung von Geräten vereinfacht.

Weitere Informationen zu Eurotech finden Sie unter Eurotech.

Problembehebung AWS IoT Greengrass V2

Verwenden Sie die Informationen und Lösungen zur Fehlerbehebung in diesem Abschnitt, um Probleme mit AWS IoT Greengrass Version 2 zu lösen.

Themen

Protokolle der AWS IoT Greengrass Kernsoftware und der Komponenten anzeigen

Die AWS IoT Greengrass Core-Software schreibt Protokolle in das lokale Dateisystem, mit denen Sie Echtzeitinformationen über das Kerngerät abrufen können. Sie können Core-Geräte auch so konfigurieren, dass sie Protokolle in CloudWatch Logs schreiben, sodass Sie Probleme bei Kerngeräten per Fernzugriff beheben können. Mithilfe dieser Protokolle können Sie Probleme mit Komponenten, Bereitstellungen und Kerngeräten identifizieren. Weitere Informationen finden Sie unter Überwachen von AWS IoT Greengrass Protokollen.

AWS IoT Greengrass Kernprobleme mit der Software

Beheben Sie Probleme mit der AWS IoT Greengrass Kernsoftware.

Themen

- com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: CKR_OPERATION_NOT_INITIALIZED
- Greengrass core device stuck on nucleus v2.12.3

Das Kerngerät konnte nicht eingerichtet werden

Wenn das AWS IoT Greengrass Core-Softwareinstallationsprogramm fehlschlägt und Sie kein Kerngerät einrichten können, müssen Sie die Software möglicherweise deinstallieren und es erneut versuchen. Weitere Informationen finden Sie unter Deinstallieren der AWS IoT Greengrass -Core-Software.

Die AWS IoT Greengrass Core-Software konnte nicht als Systemdienst gestartet werden

Wenn die AWS IoT Greengrass Core-Software nicht gestartet werden kann, überprüfen Sie die Systemdienstprotokolle, um das Problem zu identifizieren. Ein häufiges Problem besteht darin, dass Java in der Umgebungsvariablen PATH (Linux) oder der Systemvariablen PATH (Windows) nicht verfügbar ist.

Nucleus kann nicht als Systemdienst eingerichtet werden

Dieser Fehler tritt möglicherweise auf, wenn das Installationsprogramm der AWS IoT Greengrass Core-Software nicht AWS IoT Greengrass als Systemdienst eingerichtet werden kann. Auf Linux-Geräten tritt dieser Fehler normalerweise auf, wenn das Core-Gerät nicht über das Systemd-Init-System verfügt. Das Installationsprogramm kann die AWS IoT Greengrass Core-Software erfolgreich einrichten, auch wenn der Systemdienst nicht eingerichtet werden kann.

Führen Sie eine der folgenden Aktionen aus:

- Konfigurieren Sie die AWS IoT Greengrass Core-Software und führen Sie sie als Systemdienst aus. Sie müssen die Software als Systemdienst konfigurieren, um alle Funktionen von nutzen zu können AWS IoT Greengrass. Sie können systemd installieren oder ein anderes Init-System verwenden. Weitere Informationen finden Sie unter Den Greengrass Nucleus als Systemdienst konfigurieren.
- Führen Sie die AWS IoT Greengrass Core-Software ohne Systemdienst aus. Sie können die Software mit einem Loader-Skript ausführen, das der Installer im Greengrass-Stammordner einrichtet. Weitere Informationen finden Sie unter Ausführen der AWS IoT Greengrass Core-Software ohne Systemservice.

Es konnte keine Verbindung hergestellt werden AWS IoT Core

Dieser Fehler tritt möglicherweise auf, wenn die AWS IoT Greengrass Core-Software beispielsweise keine Verbindung AWS IoT Core zum Abrufen von Bereitstellungsaufträgen herstellen kann. Gehen Sie wie folgt vor:

- Vergewissern Sie sich, dass Ihr Core-Gerät eine Verbindung zum Internet herstellen kann und AWS IoT Core. Weitere Informationen zum AWS IoT Core Endpunkt, mit dem Ihr Gerät eine Verbindung herstellt, finden Sie unterKonfigurieren Sie die AWS IoT Greengrass Core-Software.
- Vergewissern Sie sich, dass das Gerät AWS IoT Ihres Kerngeräts ein Zertifikat verwendet`iot:Connect`, das die, `iot:Publish``iot:Receive`, und `iot:Subscribe` - Berechtigungen zulässt.
- Wenn Ihr Hauptgerät einen Netzwerk-Proxy verwendet, überprüfen Sie, ob Ihr Kerngerät eine Geräterolle hat und ob seine Rolle die`iot:Connect`, `iot:Publish``iot:Receive`, und `iot:Subscribe` Berechtigungen zulässt.

Fehler: Nicht genügend Arbeitsspeicher

Dieser Fehler tritt normalerweise auf, wenn Ihr Gerät nicht über ausreichend Speicher verfügt, um ein Objekt im Java-Heap zuzuweisen. Auf Geräten mit begrenztem Speicher müssen Sie möglicherweise eine maximale Heap-Größe angeben, um die Speicherzuweisung zu steuern. Weitere Informationen finden Sie unter Steuern Sie die Speicherzuweisung mit JVM-Optionen.

Greengrass CLI kann nicht installiert werden

Möglicherweise wird die folgende Konsolenmeldung angezeigt, wenn Sie das `--deploy-dev-tools` Argument in Ihrem Installationsbefehl für AWS IoT Greengrass Core verwenden.

```
Thing group exists, it could have existing deployment and devices, hence NOT creating
 deployment for Greengrass first party dev tools, please manually create a deployment
 if you wish to
```

Dies tritt auf, wenn die Greengrass-CLI-Komponente nicht installiert ist, weil Ihr Kerngerät Mitglied einer Dinggruppe ist, für die bereits eine Bereitstellung vorhanden ist. Wenn Sie diese Meldung sehen, können Sie die Greengrass-CLI-Komponente (`aws.greengrass.Cli`) manuell auf dem Gerät bereitstellen, um die Greengrass-CLI zu installieren. Weitere Informationen finden Sie unter Installieren Sie die Greengrass-CLI.

User root is not allowed to execute

Dieser Fehler wird möglicherweise angezeigt, wenn der Benutzer, der die AWS IoT Greengrass Core-Software ausführt, (normalerweise`root`) nicht berechtigt ist, die Software `sudo` mit einem Benutzer und einer Gruppe zu verwenden. Für den `ggc_user` Standardsystembenutzer sieht dieser Fehler wie folgt aus:

```
Sorry, user root is not allowed to execute <command> as ggc_user:ggc_group.
```

Vergewissern Sie sich, dass Ihre `/etc/sudoers` Datei dem Benutzer die Erlaubnis gibt, `sudo` sie unter anderen Gruppen auszuführen. Die Zugriffsrechte für den Benutzer `/etc/sudoers` sollten wie im folgenden Beispiel aussehen.

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

com.aws.greengrass.lifecyclemanager.GenericExternalService: Could not determine user/group to run with

Dieser Fehler tritt möglicherweise auf, wenn das Kerngerät versucht, eine Komponente auszuführen, und der Greengrass-Nucleus keinen Standardsystembenutzer für die Ausführung von Komponenten angibt.

Um dieses Problem zu beheben, konfigurieren Sie den Greengrass-Nucleus so, dass er den Standardsystembenutzer angibt, der die Komponenten ausführt. Weitere Informationen finden Sie unter Konfigurieren Sie den Benutzer, der die Komponenten ausführt und Konfigurieren Sie den Standardkomponentenbenutzer.

Failed to map segment from shared object: operation not permitted

Dieser Fehler tritt möglicherweise auf, wenn die AWS IoT Greengrass Core-Software nicht gestartet werden kann, weil der `/tmp` Ordner mit `noexec` entsprechenden Berechtigungen bereitgestellt wurde. Die AWS Common Runtime (CRT) -Bibliothek verwendet den `/tmp` Ordner standardmäßig.

Führen Sie eine der folgenden Aktionen aus:

- Führen Sie den folgenden Befehl aus, um den `/tmp` Ordner mit den `exec` entsprechenden Berechtigungen erneut bereitzustellen, und versuchen Sie es erneut.

```
sudo mount -o remount,exec /tmp
```

- Wenn Sie Greengrass Nucleus v2.5.0 oder höher ausführen, können Sie eine JVM-Option festlegen, um den Ordner zu ändern, den die CRT-Bibliothek verwendet. AWS Sie können den `jvmOptions` Parameter in der Greengrass Nucleus-Komponentenkonfiguration in einer Bereitstellung oder bei der Installation der AWS IoT Greengrass Core-Software angeben. Ersetzen Sie */path/to/use* durch den Pfad zu einem Ordner, den die CRT-Bibliothek verwenden kann. AWS

```
{
  "jvmOptions": "-Daws.crt.lib.dir=\"/path/to/use\""
}
```

Der Windows-Dienst konnte nicht eingerichtet werden

Dieser Fehler wird möglicherweise angezeigt, wenn Sie die AWS IoT Greengrass Core-Software auf einem Microsoft Windows 2016-Gerät installieren. Die AWS IoT Greengrass Core-Software wird unter Windows 2016 nicht unterstützt. Eine Liste der unterstützten Betriebssysteme finden Sie unterUnterstützte Plattformen.

Wenn Sie Windows 2016 verwenden müssen, können Sie Folgendes tun:

1. Entpacken Sie das heruntergeladene AWS IoT Greengrass Core-Installationsarchiv
2. Öffnen Sie im `Greengrass` Verzeichnis die `bin/greengrass.xml.template` Datei.
3. Fügen Sie das `<autoRefresh>` Tag am Ende der Datei direkt vor dem `</service>` Tag hinzu.

```
  </log>
  <autoRefresh>false</autoRefresh>
</service>
```

com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager

Dieser Fehler wird möglicherweise angezeigt, wenn Sie die AWS IoT Greengrass Core-Software ohne eine Root-Zertifizierungsstellendatei (CA) installieren.

```
2022-06-05T10:00:39.556Z [INFO] (main) com.aws.greengrass.lifecyclemanager.Kernel:
 service-loaded. {serviceName=DeploymentService}
2022-06-05T10:00:39.943Z [WARN] (main)
 com.aws.greengrass.componentmanager.ClientConfigurationUtils: configure-greengrass-
mutual-auth. Error during configure greengrass client mutual auth. {}
com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager
```

Vergewissern Sie sich, dass Sie eine gültige Root-CA-Datei mit dem `rootCaPath` Parameter in der Konfigurationsdatei angeben, die Sie dem Installationsprogramm zur Verfügung stellen. Weitere Informationen finden Sie unter Installieren Sie die AWS IoT Greengrass Core-Software..

com.aws.greengrass.deployment.IotJobsHelper: No connection available during subscribing to Iot Jobs descriptions topic. Will retry in sometime

Diese Warnmeldung wird möglicherweise angezeigt, wenn das Core-Gerät keine Verbindung herstellen kann, AWS IoT Core um Benachrichtigungen über Bereitstellungsaufträge zu abonnieren. Gehen Sie wie folgt vor:

- Vergewissern Sie sich, dass das Hauptgerät mit dem Internet verbunden ist und den von Ihnen konfigurierten AWS IoT Datenendpunkt erreichen kann. Weitere Informationen zu Endpunkten, die von Kerngeräten verwendet werden, finden Sie unterZulassen von Gerätedatenverkehr über einen Proxy oder eine Firewall.
- Überprüfen Sie die Greengrass-Protokolle auf andere Fehler, die andere Hauptursachen aufdecken.

software.amazon.awssdk.services.iam.model.IamException: The security token included in the request is invalid

Dieser Fehler tritt möglicherweise auf, wenn Sie die AWS IoT Greengrass Core-Software mit automatischer Bereitstellung installieren und das Installationsprogramm ein ungültiges AWS Sitzungstoken verwendet. Gehen Sie wie folgt vor:

- Wenn Sie temporäre Sicherheitsanmeldeinformationen verwenden, überprüfen Sie, ob das Sitzungstoken korrekt ist und ob Sie das vollständige Sitzungstoken kopieren und einfügen.
- Wenn Sie langfristige Sicherheitsanmeldedaten verwenden, stellen Sie sicher, dass das Gerät nicht über ein Sitzungstoken aus einer Zeit verfügt, in der Sie zuvor temporäre Anmeldeinformationen verwendet haben. Gehen Sie wie folgt vor:

1. Führen Sie den folgenden Befehl aus, um die Umgebungsvariable für das Sitzungstoken zu deaktivieren.

   Linux or Unix

   ```
   unset AWS_SESSION_TOKEN
   ```

   Windows Command Prompt (CMD)

   ```
   set AWS_SESSION_TOKEN=
   ```

   PowerShell

   ```
   Remove-Item Env:\AWS_SESSION_TOKEN
   ```

2. Überprüfen Sie, ob die AWS Anmeldeinformationsdatei,`~/.aws/credentials`, ein Sitzungstoken, `aws_session_token` enthält. Wenn ja, entfernen Sie diese Zeile aus der Datei.

   ```
   aws_session_token = AQoEXAMPLEH4aoAH0gNCAPyJxz4BlCFFxWNE1OPTgk5TthT
   +FvwqnKwRcOIfrRh3c/LTo6UDdyJwOOvEVPvLXCrrrUtdnniCEXAMPLE/
   IvU1dYUg2RVAJBanLiHb4IgRmpRV3zrkuWJOgQs8IZZaIv2BXIa2R4Olgk
   ```

Sie können die AWS IoT Greengrass Core-Software auch ohne Angabe von AWS Anmeldeinformationen installieren. Weitere Informationen finden Sie unter Installieren Sie die AWS IoT Greengrass Core-Software mit manueller Ressourcenbereitstellung oder Installieren Sie die AWS IoT Greengrass Core-Software mit AWS IoT Flottenbereitstellung.

software.amazon.awssdk.services.iot.model.IotException: User: <user> is not authorized to perform: iot:GetPolicy

Dieser Fehler tritt möglicherweise auf, wenn Sie die AWS IoT Greengrass Core-Software mit automatischer Bereitstellung installieren und das Installationsprogramm AWS Anmeldeinformationen verwendet, für die nicht die erforderlichen Berechtigungen erforderlich sind. Weitere Informationen zu den erforderlichen Berechtigungen finden Sie unterMinimale IAM-Richtlinie für das Installationsprogramm zur Bereitstellung von Ressourcen.

Überprüfen Sie die Berechtigungen für die IAM-Identität der Anmeldeinformationen und gewähren Sie der IAM-Identität alle erforderlichen Berechtigungen, die fehlen.

Error: com.aws.greengrass.shadowmanager.sync.model.FullShadowSyncRequest: Could not execute cloud shadow get request

Dieser Fehler wird möglicherweise angezeigt, wenn Sie die Shadow-Manager-Komponente verwenden, um Geräteschatten mit zu synchronisieren. AWS IoT Core Der HTTP-Statuscode 403 gibt an, dass dieser Fehler aufgetreten ist, weil die AWS IoT Richtlinie des Kerngeräts keine Anrufberechtigung gewährt`GetThingShadow`.

```
com.aws.greengrass.shadowmanager.sync.model.FullShadowSyncRequest: Could not execute
 cloud shadow get request. {thing name=MyGreengrassCore, shadow name=MyShadow}
2021-07-14T21:09:02.456Z [ERROR] (pool-2-thread-109)
 com.aws.greengrass.shadowmanager.sync.SyncHandler: sync. Skipping sync request. {thing
 name=MyGreengrassCore, shadow name=MyShadow}
com.aws.greengrass.shadowmanager.exception.SkipSyncRequestException:
 software.amazon.awssdk.services.iotdataplane.model.IotDataPlaneException:
 null (Service: IotDataPlane, Status Code: 403, Request ID:
 f6e713ba-1b01-414c-7b78-5beb3f3ad8f6, Extended Request ID: null)
```

Um lokale Shadows mit zu synchronisieren AWS IoT Core, muss die AWS IoT Richtlinie des Kerngeräts die folgenden Berechtigungen gewähren:

- `iot:GetThingShadow`
- `iot:UpdateThingShadow`
- `iot:DeleteThingShadow`

Überprüfen Sie die AWS IoT Richtlinien des Core-Geräts und fügen Sie alle erforderlichen Berechtigungen hinzu, die noch fehlen. Weitere Informationen finden Sie hier:

- AWS IoT Core Richtlinienaktionen im AWS IoT Entwicklerhandbuch
- Aktualisieren der AWS IoT Richtlinie eines Core-Geräts

Operation aws.greengrass#<operation> is not supported by Greengrass

Dieser Fehler tritt möglicherweise auf, wenn Sie einen IPC-Vorgang (Interprocess Communication) in einer benutzerdefinierten AWS Greengrass-Komponente verwenden und die erforderliche, bereitgestellte Komponente nicht auf dem Kerngerät installiert ist.

Um dieses Problem zu beheben, fügen Sie die erforderliche Komponente als Abhängigkeit zu Ihrem Komponentenrezept hinzu, sodass die AWS IoT Greengrass Core-Software die erforderliche Komponente bei der Bereitstellung Ihrer Komponente installiert hat.

- Geheime Werte abrufen — `aws.greengrass.SecretManager`
- Interagiere mit lokalen Schatten — `aws.greengrass.ShadowManager`
- Lokale Bereitstellungen und Komponenten verwalten — `aws.greengrass.Cli` v2.6.0 oder höher
- Authentifizieren und autorisieren Sie Client-Geräte — v2.2.0 oder höher `aws.greengrass.clientdevices.Auth`

java.io.FileNotFoundException: <stream-manager-store-root-dir>/stream_manager_metadata_store (Permission denied)

Dieser Fehler wird möglicherweise in der Stream Manager-Protokolldatei (`aws.greengrass.StreamManager.log`) angezeigt, wenn Sie den Stream Manager so konfigurieren, dass er einen Stammordner verwendet, der nicht existiert oder nicht über die richtigen Berechtigungen verfügt. Weitere Informationen zur Konfiguration dieses Ordners finden Sie unter Stream Manager-Konfiguration.

com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: Private key or certificate with label <label> does not exist

Dieser Fehler tritt auf, wenn die PKCS #11 -Anbieterkomponente den privaten Schlüssel oder das Zertifikat, das Sie bei der Konfiguration der AWS IoT Greengrass Core-Software für die Verwendung eines Hardware-Sicherheitsmoduls (HSM) angeben, nicht finden oder laden kann. Gehen Sie wie folgt vor:

- Überprüfen Sie anhand des Steckplatzes, der Benutzer-PIN und der Objektbezeichnung, für die Sie die AWS IoT Greengrass Core-Software konfiguriert haben, ob der private Schlüssel und das Zertifikat im HSM gespeichert sind.
- Vergewissern Sie sich, dass der private Schlüssel und das Zertifikat dieselbe Objektbezeichnung im HSM verwenden.
- Wenn Ihr HSM Objekt-IDs unterstützt, überprüfen Sie, ob der private Schlüssel und das Zertifikat dieselbe Objekt-ID im HSM verwenden.

In der Dokumentation zu Ihrem HSM erfahren Sie, wie Sie Details zu den Sicherheitstoken im HSM abfragen können. Wenn Sie den Steckplatz, die Objektbezeichnung oder die Objekt-ID für ein Sicherheitstoken ändern müssen, lesen Sie in der Dokumentation zu Ihrem HSM nach, wie Sie das tun können.

software.amazon.awssdk.services.secretsmanager.model.SecretsManagerExcep User: <user> is not authorized to perform: secretsmanager:GetSecretValue on resource: <arn>

Dieser Fehler kann auftreten, wenn Sie die Secret Manager-Komponente verwenden, um ein AWS Secrets Manager Geheimnis bereitzustellen. Wenn die Token-Exchange-IAM-Rolle des Kerngeräts keine Erlaubnis zum Abrufen des Geheimnisses erteilt, schlägt die Bereitstellung fehl und die Greengrass-Protokolle enthalten diesen Fehler.

Um ein Core-Gerät zum Herunterladen eines Secrets zu autorisieren

1. Fügen Sie die `secretsmanager:GetSecretValue` Berechtigung zur Token-Austauschrolle des Kerngeräts hinzu. Das folgende Beispiel für eine Richtlinienanweisung erteilt die Erlaubnis, den Wert eines Geheimnisses abzurufen.

   ```
   {
       "Effect": "Allow",
       "Action": [
           "secretsmanager:GetSecretValue"
       ],
       "Resource": [
           "arn:aws:secretsmanager:us-west-2:123456789012:secret:MyGreengrassSecret-
   abcdef"
       ]
   }
   ```

   Weitere Informationen finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.
2. Wenden Sie die Bereitstellung erneut auf das Kerngerät an. Führen Sie eine der folgenden Aktionen aus:
   - Überarbeiten Sie die Bereitstellung ohne Änderungen. Das Kerngerät versucht erneut, den geheimen Schlüssel herunterzuladen, wenn es die überarbeitete Bereitstellung erhält. Weitere Informationen finden Sie unter Überarbeiten von Bereitstellungen.

- Starten Sie die AWS IoT Greengrass Core-Software neu, um die Bereitstellung erneut zu versuchen. Weitere Informationen finden Sie unter Ausführen der AWS IoT Greengrass - Core-Software.

Die Bereitstellung ist erfolgreich, wenn Secret Manager das Secret erfolgreich herunterlädt.

software.amazon.awssdk.services.secretsmanager.model.SecretsManagerExcep Access to KMS is not allowed

Dieser Fehler kann auftreten, wenn Sie die Secret Manager-Komponente verwenden, um ein AWS Secrets Manager Geheimnis bereitzustellen, das mit einem AWS Key Management Service Schlüssel verschlüsselt ist. Wenn die Token-Exchange-IAM-Rolle des Kerngeräts keine Erlaubnis zur Entschlüsselung des Geheimnisses erteilt, schlägt die Bereitstellung fehl und die Greengrass-Protokolle enthalten diesen Fehler.

Um das Problem zu beheben, fügen Sie die `kms:Decrypt` Berechtigung zur Token-Austauschrolle des Kerngeräts hinzu. Weitere Informationen finden Sie hier:

- Geheime Verschlüsselung und Entschlüsselung im AWS Secrets Manager Benutzerhandbuch
- Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS

java.lang.NoClassDefFoundError: com/aws/greengrass/security/CryptoKeySpi

Dieser Fehler wird möglicherweise angezeigt, wenn Sie versuchen, die AWS IoT Greengrass Core-Software mit Hardwaresicherheit zu installieren, und Sie eine frühere Greengrass Nucleus-Version verwenden, die die Hardware-Sicherheitsintegration nicht unterstützt. Um die Hardware-Sicherheitsintegration verwenden zu können, müssen Sie Greengrass Nucleus v2.5.3 oder höher verwenden.

com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: CKR_OPERATION_NOT_INITIALIZED

Dieser Fehler wird möglicherweise angezeigt, wenn Sie die TPM2-Bibliothek verwenden, wenn Sie AWS IoT Greengrass Core als Systemdienst ausführen.

Dieser Fehler weist darauf hin, dass Sie eine Umgebungsvariable hinzufügen müssen, die den Speicherort des PKCS #11 -Speichers in der AWS IoT Greengrass Core-Systemd-Servicedatei angibt.

Weitere Informationen finden Sie im Abschnitt „Anforderungen“ der PKCS #11-Anbieter Komponentendokumentation.

Greengrass core device stuck on nucleus v2.12.3

Wenn Ihr Greengrass Core-Gerät Ihre Bereitstellung von Nucleus Version 2.12.3 nicht überarbeitet, müssen Sie die `Greengrass.jar` Datei möglicherweise herunterladen und durch Greengrass Nucleus Version 2.12.2 ersetzen. Gehen Sie wie folgt vor:

1. Führen Sie auf Ihrem Greengrass Core-Gerät den folgenden Befehl aus, um die Greengrass Core-Software zu beenden.

   Linux or Unix

   ```
   sudo systemctl stop greengrass
   ```

   Windows Command Prompt (CMD)

   ```
   sc stop "greengrass"
   ```

   PowerShell

   ```
   Stop-Service -Name "greengrass"
   ```

2. Laden Sie die AWS IoT Greengrass Software auf Ihrem Core-Gerät in eine Datei mit dem Namen herunter. `greengrass-2.12.2.zip`

   Linux or Unix

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-2.12.2.zip >
    greengrass-2.12.2.zip
   ```

Windows Command Prompt (CMD)

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-2.12.2.zip >
 greengrass-2.12.2.zip
```

PowerShell

```
iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-2.12.2.zip -
OutFile greengrass-2.12.2.zip
```

3. Entpacken Sie die AWS IoT Greengrass Core-Software in einen Ordner auf Ihrem Gerät. *GreengrassInstaller*Ersetzen Sie es durch den Ordner, den Sie verwenden möchten.

   Linux or Unix

   ```
   unzip greengrass-2.12.2.zip -d GreengrassInstaller && rm greengrass-2.12.2.zip
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir GreengrassInstaller && tar -xf greengrass-2.12.2.zip -
   C GreengrassInstaller && del greengrass-2.12.2.zip
   ```

   PowerShell

   ```
   Expand-Archive -Path greengrass-2.12.2.zip -DestinationPath .\
   \GreengrassInstaller
   rm greengrass-2.12.2.zip
   ```

4. Führen Sie den folgenden Befehl aus, um die Nucleus Version 2.12.3 Greengrass JAR-Datei mit der Nucleus Version 2.12.2 Greengrass JAR-Datei zu überschreiben.

   Linux or Unix

   ```
   sudo cp ./GreengrassInstaller/lib/Greengrass.jar /greengrass/v2/packages/
   artifacts-unarchived/aws.greengrass.Nucleus/2.12.3/aws.greengrass.nucleus/lib
   ```

Windows Command Prompt (CMD)

```
robocopy ./GreengrassInstaller/lib/Greengrass.jar /greengrass/v2/packages/
artifacts-unarchived/aws.greengrass.Nucleus/2.12.3/aws.greengrass.nucleus/lib /E
```

PowerShell

```
cp -Path ./GreengrassInstaller/lib/Greengrass.jar -Destination /
greengrass/v2/packages/artifacts-unarchived/aws.greengrass.Nucleus/2.12.3/
aws.greengrass.nucleus/lib
```

5. Führen Sie den folgenden Befehl aus, um die Greengrass Core-Software zu starten.

   Linux or Unix

   ```
   sudo systemctl start greengrass
   ```

   Windows Command Prompt (CMD)

   ```
   sc start "greengrass"
   ```

   PowerShell

   ```
   Start-Service -Name "greengrass"
   ```

AWS IoT Greengrass Cloud-Probleme

Verwenden Sie die folgenden Informationen, um Probleme mit der AWS IoT Greengrass Konsole und der API zu beheben. Jeder Eintrag entspricht einer Fehlermeldung, die möglicherweise angezeigt wird, wenn Sie eine Aktion ausführen.

An error occurred (AccessDeniedException) when calling the CreateComponentVersion operation: User: arn:aws:iam::123456789012:user/<username> is not authorized to perform: null

Dieser Fehler wird möglicherweise angezeigt, wenn Sie eine Komponentenversion über die AWS IoT Greengrass Konsole oder während des CreateComponentVersionVorgangs erstellen.

Dieser Fehler weist darauf hin, dass Ihr Rezept kein gültiges JSON- oder YAML-Format ist. Überprüfen Sie die Syntax Ihres Rezepts, beheben Sie alle Syntaxprobleme und versuchen Sie es erneut. Sie können einen Online-JSON- oder YAML-Syntaxprüfer verwenden, um Syntaxprobleme in Ihrem Rezept zu identifizieren.

Invalid Input: Encountered following errors in Artifacts: {<s3ArtifactUri> = Specified artifact resource cannot be accessed}

Dieser Fehler tritt möglicherweise auf, wenn Sie eine Komponentenversion über die AWS IoT Greengrass Konsole oder während des Vorgangs erstellen. CreateComponentVersion Dieser Fehler weist darauf hin, dass ein S3-Artefakt im Komponentenrezept nicht gültig ist.

Gehen Sie wie folgt vor:

- Vergewissern Sie sich, dass sich der S3-Bucket an derselben AWS-Region Stelle befindet, an der Sie die Komponente erstellt haben. AWS IoT Greengrass unterstützt keine regionsübergreifenden Anfragen nach Komponentenartefakten.
- Überprüfen Sie, ob es sich bei der Artefakt-URI um eine gültige S3-Objekt-URL handelt, und überprüfen Sie, ob das Artefakt unter dieser S3-Objekt-URL vorhanden ist.
- Vergewissern Sie sich, dass Sie AWS-Konto über die entsprechende S3-Objekt-URL auf das Artefakt zugreifen dürfen.

INACTIVE deployment status

Möglicherweise erhalten Sie einen `INACTIVE` Bereitstellungsstatus, wenn Sie die ListDeploymentsAPI ohne die erforderlichen abhängigen AWS IoT Richtlinien aufrufen. Sie müssen über die erforderlichen Berechtigungen verfügen, um einen genauen Bereitstellungsstatus zu erhalten. Sie können die abhängigen Aktionen finden, indem Sie in den Aktionen suchen,

die von definiert sind, AWS IoT Greengrass V2 und den erforderlichen Berechtigungen folgen`ListDeployments`. Ohne die erforderlichen abhängigen AWS IoT Berechtigungen wird Ihnen weiterhin der Bereitstellungsstatus angezeigt, aber möglicherweise wird der Bereitstellungsstatus von `INACTIVE` falsch angezeigt.

Hauptprobleme bei der Gerätebereitstellung

Beheben Sie Bereitstellungsprobleme auf Greengrass-Kerngeräten. Jeder Eintrag entspricht einer Protokollnachricht, die Sie möglicherweise auf Ihrem Kerngerät sehen.

Themen

- Error: com.aws.greengrass.componentmanager.exceptions.PackageDownloadException: Failed to download artifact
- Error: com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchException: Integrity check for downloaded artifact failed. Probably due to file corruption.
- Error: com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException: Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements>
- software.amazon.awssdk.services.greengrassv2data.model.ResourceNotFoundException: The latest version of Component <componentName> doesn't claim platform <coreDevicePlatform> compatibility
- com.aws.greengrass.componentmanager.exceptions.PackagingException: The deployment attempts to update the nucleus from aws.greengrass.Nucleus-<version> to aws.greengrass.Nucleus-<version> but no component of type nucleus was included as target component
- Error: com.aws.greengrass.deployment.exceptions.DeploymentException: Unable to process deployment. Greengrass launch directory is not set up or Greengrass is not set up as a system service
- Info: com.aws.greengrass.deployment.exceptions.RetryableDeploymentDocumentDownloadException: Greengrass Cloud Service returned an error when getting full deployment configuration
- Warn: com.aws.greengrass.deployment.DeploymentService: Failed to get thing group hierarchy
- Info: com.aws.greengrass.deployment.DeploymentDocumentDownloader: Calling Greengrass cloud to get full deployment configuration

- Caused by: software.amazon.awssdk.services.greengrassv2data.model.GreengrassV2DataException: null (Service: GreengrassV2Data, Status Code: 403, Request ID: <some_request_id>, Extended Request ID: null)

Error: com.aws.greengrass.componentmanager.exceptions.PackageDownloadExceptio Failed to download artifact

Dieser Fehler wird möglicherweise angezeigt, wenn die AWS IoT Greengrass Core-Software ein Komponentenartefakt nicht herunterlädt, wenn das Kerngerät eine Bereitstellung anwendet. Die Bereitstellung schlägt aufgrund dieses Fehlers fehl.

Wenn Sie diesen Fehler erhalten, enthält das Protokoll auch einen Stack-Trace, anhand dessen Sie das spezifische Problem identifizieren können. Jeder der folgenden Einträge entspricht einer Meldung, die Sie möglicherweise im Stack-Trace der `Failed to download artifact` Fehlermeldung sehen.

Themen

- software.amazon.awssdk.services.s3.model.S3Exception: null (Service: S3, Status Code: 403, Request ID: null, ...)
- software.amazon.awssdk.services.s3.model.S3Exception: Access Denied (Service: S3, Status Code: 403, Request ID: <requestID>

software.amazon.awssdk.services.s3.model.S3Exception: null (Service: S3, Status Code: 403, Request ID: null, ...)

Der PackageDownloadException Fehler kann diesen Stack-Trace in den folgenden Fällen beinhalten:

- Das Komponentenartefakt ist nicht unter der S3-Objekt-URL verfügbar, die Sie im Rezept der Komponente angeben. Überprüfen Sie, ob Sie das Artefakt in den S3-Bucket hochgeladen haben und ob der Artefakt-URI mit der S3-Objekt-URL des Artefakts im Bucket übereinstimmt.
- Die Token-Austauschrolle des Kerngeräts erlaubt es der AWS IoT Greengrass Core-Software nicht, das Komponentenartefakt von der S3-Objekt-URL herunterzuladen, die Sie im Rezept der Komponente angeben. Stellen Sie sicher, dass die Token-Austauschrolle die URL des S3-Objekts zulässts3:GetObject, unter der das Artefakt verfügbar ist.

software.amazon.awssdk.services.s3.model.S3Exception: Access Denied (Service: S3, Status Code: 403, Request ID: <requestID>

Der PackageDownloadException Fehler kann diesen Stack-Trace beinhalten, wenn das Kerngerät nicht über die Berechtigung zum Aufrufen `s3:GetBucketLocation` verfügt. Die Fehlermeldung enthält auch die folgende Meldung.

```
reason: Failed to determine S3 bucket location
```

Vergewissern Sie sich, dass die Token-Austauschrolle des Kerngeräts den S3-Bucket zulässts`3:GetBucketLocation`, in dem das Artefakt verfügbar ist.

Error: com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchI Integrity check for downloaded artifact failed. Probably due to file corruption.

Dieser Fehler wird möglicherweise angezeigt, wenn die AWS IoT Greengrass Core-Software ein Komponentenartefakt nicht herunterladen kann, wenn das Kerngerät eine Bereitstellung anwendet. Die Bereitstellung schlägt fehl, weil die Prüfsumme der heruntergeladenen Artefaktdatei nicht mit der Prüfsumme übereinstimmt, die bei der Erstellung der AWS IoT Greengrass Komponente berechnet wurde.

Gehen Sie wie folgt vor:

- Prüfen Sie, ob sich die Artefaktdatei in dem S3-Bucket, in dem Sie sie hosten, geändert hat. Wenn sich die Datei seit der Erstellung der Komponente geändert hat, stellen Sie die vorherige Version wieder her, die das Kerngerät erwartet. Wenn Sie die Datei nicht auf ihre vorherige Version zurücksetzen können oder wenn Sie die neue Version der Datei verwenden möchten, erstellen Sie eine neue Version der Komponente mit der Artefaktdatei.
- Überprüfen Sie die Internetverbindung Ihres Hauptgeräts. Dieser Fehler kann auftreten, wenn die Artefaktdatei beim Herunterladen beschädigt wird. Erstellen Sie eine neue Bereitstellung und versuchen Sie es erneut.

Error: com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVers Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements>

Dieser Fehler wird möglicherweise angezeigt, wenn ein Kerngerät keine Komponentenversion finden kann, die den Anforderungen der Bereitstellungen für dieses Kerngerät entspricht. Das Kerngerät sucht im AWS IoT Greengrass Dienst und auf dem lokalen Gerät nach der Komponente. Die Fehlermeldung enthält das Ziel jeder Bereitstellung und die Versionsanforderungen dieser Bereitstellung für die Komponente. Das Bereitstellungsziel kann ein Ding, eine Dinggruppe oder sein`LOCAL_DEPLOYMENT`, was die lokale Bereitstellung auf dem Kerngerät darstellt.

Dieses Problem kann in den folgenden Fällen auftreten:

- Das Kerngerät ist das Ziel mehrerer Bereitstellungen mit widersprüchlichen Anforderungen an die Komponentenversion. Beispielsweise kann das Kerngerät das Ziel mehrerer Bereitstellungen sein, die eine `com.example.HelloWorld` Komponente enthalten, wobei für eine Bereitstellung Version 1.0.0 und für die andere Version 1.0.1 erforderlich ist. Es ist unmöglich, eine Komponente zu haben, die beide Anforderungen erfüllt, sodass die Bereitstellung fehlschlägt.
- Die Komponentenversion ist weder im AWS IoT Greengrass Dienst noch auf dem lokalen Gerät vorhanden. Die Komponente könnte beispielsweise gelöscht worden sein.
- Es gibt Komponentenversionen, die die Versionsanforderungen erfüllen, aber keine ist mit der Plattform des Kerngeräts kompatibel.
- Die AWS IoT Richtlinie des Kerngeräts gewährt die `greengrass:ResolveComponentCandidates` Genehmigung nicht. Suchen Sie `Status Code: 403` im Fehlerprotokoll nach, um dieses Problem zu identifizieren. Um dieses Problem zu beheben, fügen Sie die `greengrass:ResolveComponentCandidates` Berechtigung zur AWS IoT Richtlinie des Kerngeräts hinzu. Weitere Informationen finden Sie unter Minimale AWS IoT Richtlinie für -AWS IoT Greengrass V2Core-Geräte.

Um dieses Problem zu beheben, überarbeiten Sie die Bereitstellungen, sodass sie kompatible Komponentenversionen enthalten oder inkompatible Versionen entfernen. Weitere Informationen zur Überarbeitung von Cloud-Bereitstellungen finden Sie unter. Überarbeiten von Bereitstellungen Weitere Informationen zur Überarbeitung lokaler Bereitstellungen finden Sie im Befehl AWS IoT Greengrass CLI deployment create.

software.amazon.awssdk.services.greengrassv2data.model.ResourceNotFoundE The latest version of Component <componentName> doesn't claim platform <coreDevicePlatform> compatibility

Dieser Fehler tritt möglicherweise auf, wenn Sie eine Komponente auf einem Kerngerät bereitstellen und die Komponente keine Plattform auflistet, die mit der Plattform des Kerngeräts kompatibel ist. Führen Sie eine der folgenden Aktionen aus:

- Wenn es sich bei der Komponente um eine benutzerdefinierte Greengrass-Komponente handelt, können Sie die Komponente so aktualisieren, dass sie mit dem Kerngerät kompatibel ist. Fügen Sie ein neues Manifest hinzu, das der Plattform des Kerngeräts entspricht, oder aktualisieren Sie ein vorhandenes Manifest, sodass es mit der Plattform des Kerngeräts übereinstimmt. Weitere Informationen finden Sie unter AWS IoT Greengrass Referenz zum Komponenten-Rezept.
- Wenn die Komponente von bereitgestellt wird AWS, überprüfen Sie, ob eine andere Version der Komponente mit dem Kerngerät kompatibel ist. Wenn keine Version kompatibel ist, kontaktieren Sie uns unter AWS re:PostVerwendung des AWS IoT Greengrass Tags oder kontaktieren Sie uns AWS Support.

com.aws.greengrass.componentmanager.exceptions.PackagingException: The deployment attempts to update the nucleus from aws.greengrass.Nucleus-<version> to aws.greengrass.Nucleus-<version> but no component of type nucleus was included as target component

Dieser Fehler tritt möglicherweise auf, wenn Sie eine Komponente bereitstellen, die vom Greengrass Nucleus abhängt und auf dem Kerngerät eine frühere Greengrass Nucleus-Version als die neueste verfügbare Nebenversion ausgeführt wird. Dieser Fehler tritt auf, weil die AWS IoT Greengrass Core-Software versucht, Komponenten automatisch auf die neueste kompatible Version zu aktualisieren. Die AWS IoT Greengrass Core-Software verhindert jedoch, dass der Greengrass-Kern auf eine neue Nebenversion aktualisiert wird, da mehrere AWS bereitgestellte Komponenten von bestimmten Nebenversionen des Greengrass-Nukleus abhängen. Weitere Informationen finden Sie unter Aktualisierungsverhalten des Greengrass-Kerns.

Sie müssen die Bereitstellung überarbeiten, um die Greengrass Nucleus-Version anzugeben, die Sie verwenden möchten. Führen Sie eine der folgenden Aktionen aus:

- Überarbeiten Sie die Bereitstellung, um die Greengrass Nucleus-Version anzugeben, die derzeit auf dem Kerngerät ausgeführt wird.
- Überarbeiten Sie die Bereitstellung, um eine spätere Nebenversion des Greengrass-Nukleus zu spezifizieren. Wenn Sie diese Option wählen, müssen Sie auch die Versionen aller AWS bereitgestellten Komponenten aktualisieren, die von bestimmten Nebenversionen von Greengrass Nucleus abhängen. Weitere Informationen finden Sie unter AWSVon bereitgestellte Komponenten.

Error: com.aws.greengrass.deployment.exceptions.DeploymentException: Unable to process deployment. Greengrass launch directory is not set up or Greengrass is not set up as a system service

Dieser Fehler tritt möglicherweise auf, wenn Sie ein Greengrass-Gerät von einer Dinggruppe in eine andere verschieben und dann zurück zur ursprünglichen Gruppe mit Bereitstellungen, für die Greengrass neu gestartet werden muss.

Um dieses Problem zu beheben, erstellen Sie das Startverzeichnis für das Gerät neu. Wir empfehlen außerdem dringend, auf Version 2.9.6 oder höher von Greengrass Nucleus zu aktualisieren.

Das Folgende ist ein Linux-Skript zum Neuerstellen des Startverzeichnisses. Speichern Sie das Skript in einer Datei namens`fix_directory.sh`.

```
#!/bin/bash

set -e

GG_ROOT=$1
GG_VERSION=$2

CURRENT="$GG_ROOT/alts/current"

if [ ! -L "$CURRENT" ]; then
  mkdir -p $GG_ROOT/alts/directory_fix
  echo "Relinking $GG_ROOT/alts/directory_fix to $CURRENT"
  ln -sf $GG_ROOT/alts/directory_fix $CURRENT
fi

TARGET=$(readlink $CURRENT)

if [[ ! -d "$TARGET" ]]; then
  echo "Creating directory: $TARGET"
```

```
  mkdir -p "$TARGET"
fi

DISTRO_LINK="$TARGET/distro"
DISTRO="$GG_ROOT/packages/artifacts-unarchived/aws.greengrass.Nucleus/$GG_VERSION/
aws.greengrass.nucleus/"
echo "Relinking Nucleus artifacts to $DISTRO_LINK"
ln -sf $DISTRO $DISTRO_LINK
```

Um das Skript auszuführen, führen Sie den folgenden Befehl aus:

```
[root@ip-172-31-27-165 ~]# ./fix_directory.sh /greengrass/v2 2.9.5
Relinking /greengrass/v2/alts/directory_fix to /greengrass/v2/alts/current
Relinking Nucleus artifacts to /greengrass/v2/alts/directory_fix/distro
```

Info: com.aws.greengrass.deployment.exceptions.RetryableDeploymentDocumentDow Greengrass Cloud Service returned an error when getting full deployment configuration

Dieser Fehler tritt möglicherweise auf, wenn das Core-Gerät ein umfangreiches Bereitstellungsdokument empfängt, bei dem es sich um ein Bereitstellungsdokument mit mehr als 7 KB (für Bereitstellungen, die auf Dinge abzielen) oder 31 KB (für Bereitstellungen, die auf Dinggruppen abzielen) handelt. Um ein umfangreiches Bereitstellungsdokument abzurufen, muss die AWS IoT Richtlinie eines Kerngeräts die `greengrass:GetDeploymentConfiguration` Genehmigung zulassen. Dieser Fehler kann auftreten, wenn das Kerngerät nicht über diese Berechtigung verfügt. Wenn dieser Fehler auftritt, wird die Bereitstellung auf unbestimmte Zeit wiederholt und ihr Status lautet In Bearbeitung ()`IN_PROGRESS`.

Um dieses Problem zu beheben, fügen Sie die `greengrass:GetDeploymentConfiguration` Berechtigung zur Richtlinie des Kerngeräts hinzu. AWS IoT Weitere Informationen finden Sie unter Aktualisieren der AWS IoT Richtlinie eines Core-Geräts.

Warn: com.aws.greengrass.deployment.DeploymentService: Failed to get thing group hierarchy

Diese Warnung wird möglicherweise angezeigt, wenn das Core-Gerät eine Bereitstellung erhält und die AWS IoT Richtlinie des Core-Geräts die `greengrass:ListThingGroupsForCoreDevice`

Genehmigung nicht zulässt. Wenn Sie eine Bereitstellung erstellen, verwendet das Kerngerät diese Berechtigung, um seine Dinggruppen zu identifizieren und Komponenten für alle Dinggruppen zu entfernen, aus denen Sie das Kerngerät entfernt haben. Wenn auf dem Kerngerät Greengrass Nucleus v2.5.0 ausgeführt wird, schlägt die Bereitstellung fehl. Wenn auf dem Kerngerät Greengrass Nucleus v2.5.1 oder höher ausgeführt wird, wird die Bereitstellung fortgesetzt, ohne dass Komponenten entfernt werden. Weitere Informationen zum Verhalten beim Entfernen von Dinggruppen finden Sie unter. Bereitstellen von AWS IoT Greengrass Komponenten auf Geräten

Um das Verhalten des Kerngeräts zu aktualisieren und Komponenten für Dinggruppen zu entfernen, aus denen Sie das Kerngerät entfernen, fügen Sie die `greengrass:ListThingGroupsForCoreDevice` entsprechende Berechtigung zur AWS IoT Richtlinie des Kerngeräts hinzu. Weitere Informationen finden Sie unter Aktualisieren der AWS IoT Richtlinie eines Core-Geräts.

Info: com.aws.greengrass.deployment.DeploymentDocumentDownloader: Calling Greengrass cloud to get full deployment configuration

Diese Informationsmeldung wird möglicherweise mehrmals ohne Fehler gedruckt, da das Kerngerät den Fehler auf Protokollebene `DEBUG` protokolliert. Dieses Problem kann auftreten, wenn das Kerngerät ein umfangreiches Bereitstellungsdokument erhält. Wenn dieses Problem auftritt, wird die Bereitstellung auf unbestimmte Zeit wiederholt und ihr Status lautet In Bearbeitung ()`IN_PROGRESS`. Weitere Informationen zur Behebung dieses Problems finden Sie in diesem Eintrag zur Problembehandlung.

Caused by: software.amazon.awssdk.services.greengrassv2data.model.GreengrassV2DataE null (Service: GreengrassV2Data, Status Code: 403, Request ID: <some_request_id>, Extended Request ID: null)

Dieser Fehler wird möglicherweise angezeigt, wenn eine Datenebenen-API nicht über `iot:Connect` die entsprechenden Berechtigungen verfügt. Wenn Sie nicht über die richtige Richtlinie verfügen, erhalten Sie eine. `GreengrassV2DataException: 403` Folgen Sie diesen Anweisungen, um eine Berechtigungsrichtlinie zu erstellen:Erstellen einer AWS IoT-Richtlinie.

Probleme mit den wichtigsten Gerätekomponenten

Beheben Sie Probleme mit Greengrass-Komponenten auf Kerngeräten.

Themen

Warn: '<command>' is not recognized as an internal or external command

Möglicherweise wird dieser Fehler in den Protokollen einer Greengrass-Komponente angezeigt, wenn die AWS IoT Greengrass Core-Software einen Befehl im Lifecycle-Skript der Komponente nicht ausführen kann. Der Status der Komponente ergibt `BROKEN` sich aus diesem Fehler. Dieser Fehler kann auftreten, wenn der Systembenutzer, der die Komponente ausführt, z. B. `ggc_user` die ausführbare Datei des Befehls in den Ordnern im PATH nicht finden kann.

Überprüfen Sie auf Windows-Geräten, ob sich der Ordner, der die ausführbare Datei enthält, im Ordner `PATH` für den Systembenutzer befindet, der die Komponente ausführt. Wenn es in der fehlt`PATH`, führen Sie einen der folgenden Schritte aus:

- Fügen Sie den Ordner der ausführbaren Datei zur PATH Systemvariablen hinzu, die für alle Benutzer verfügbar ist. Starten Sie dann die Komponente neu.

  Wenn Sie Greengrass Nucleus 2.5.0 ausführen, müssen Sie nach dem Update der PATH Systemvariablen die AWS IoT Greengrass Core-Software neu starten, um Komponenten mit der aktualisierten Version auszuführen. PATH Wenn die AWS IoT Greengrass Core-Software das Update PATH nach dem Neustart der Software nicht verwendet, starten Sie das Gerät neu und versuchen Sie es erneut. Weitere Informationen finden Sie unter Ausführen der AWS IoT Greengrass -Core-Software.
- Fügen Sie den Ordner der ausführbaren Datei zur PATH Benutzervariablen für den Systembenutzer hinzu, der die Komponente ausführt.

Python-Skript protokolliert keine Nachrichten

Greengrass-Core-Geräte sammeln Protokolle, anhand derer Sie Probleme mit Komponenten identifizieren können. Wenn Ihre Python-Skripte `stdout` und `stderr` -Meldungen nicht in Ihren Komponentenprotokollen erscheinen, müssen Sie möglicherweise den Puffer leeren oder die Pufferung für diese Standardausgabestreams in Python deaktivieren. Führen Sie eine der folgenden Aktionen aus:

- Führen Sie Python mit dem Argument -u aus, um die Pufferung auf `stdout` und zu deaktivieren. `stderr`

  Linux or Unix

  ```
  python3 -u hello_world.py
  ```

  Windows

  ```
  py -3 -u hello_world.py
  ```
- Verwenden Sie Setenv im Rezept Ihrer Komponente, um die Umgebungsvariable PYTHONUNBUFFERED auf eine nicht leere Zeichenfolge zu setzen. Diese Umgebungsvariable deaktiviert die Pufferung bei und. `stdout` `stderr`
- Leert den Puffer für die `stdout` OR-Streams. `stderr` Führen Sie eine der folgenden Aktionen aus:
  - Leert eine Nachricht beim Drucken.

```
import sys

print('Hello, error!', file=sys.stderr, flush=True)
```

- Löscht eine Nachricht nach dem Drucken. Sie können mehrere Nachrichten senden, bevor Sie den Stream leeren.

```
import sys

print('Hello, error!', file=sys.stderr)
sys.stderr.flush()
```

Weitere Hinweise dazu, wie Sie überprüfen können, ob Ihr Python-Skript Logmeldungen ausgibt, finden Sie unterÜberwachen von AWS IoT Greengrass Protokollen.

Die Komponentenkonfiguration wird nicht aktualisiert, wenn die Standardkonfiguration geändert wird

Wenn Sie die `DefaultConfiguration` Rezeptur einer Komponente ändern, ersetzt die neue Standardkonfiguration während einer Bereitstellung nicht die bestehende Konfiguration der Komponente. Um die neue Standardkonfiguration anzuwenden, müssen Sie die Konfiguration der Komponente auf die Standardeinstellungen zurücksetzen. Wenn Sie die Komponente bereitstellen, geben Sie eine einzelne leere Zeichenfolge als Reset-Update an.

Console

Pfade zurücksetzen

```
[""]
```

AWS CLI

Der folgende Befehl erstellt eine Bereitstellung auf einem Kerngerät.

```
aws greengrassv2 create-deployment --cli-input-json file://reset-configuration-
deployment.json
```

Die `reset-configuration-deployment.json` Datei enthält das folgende JSON-Dokument.

```
{
  "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "deploymentName": "Deployment for MyGreengrassCore",
  "components": {
    "com.example.HelloWorld": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {,
        "reset": [""]
      }
    }
  }
}
```

Greengrass CLI

Der folgende Greengrass-CLI-Befehl erstellt eine lokale Bereitstellung auf einem Core-Gerät.

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.HelloWorld=1.0.0" \
  --update-config reset-configuration-deployment.json
```

Die `reset-configuration-deployment.json` Datei enthält das folgende JSON-Dokument.

```
{
  "com.example.HelloWorld": {
    "RESET": [""]
  }
}
```

awsiot.greengrasscoreipc.model.UnauthorizedError

Möglicherweise wird dieser Fehler in den Protokollen einer Greengrass-Komponente angezeigt, wenn die Komponente nicht berechtigt ist, einen IPC-Vorgang für eine Ressource auszuführen. Um einer Komponente die Berechtigung zum Aufrufen einer IPC-Operation zu erteilen, definieren Sie in der Konfiguration der Komponente eine IPC-Autorisierungsrichtlinie. Weitere Informationen finden Sie unter Autorisieren Sie Komponenten zur Ausführung von IPC-Vorgängen.

> **Tip**
>
> Wenn Sie das `DefaultConfiguration` in der Rezeptur einer Komponente ändern, müssen Sie die Konfiguration der Komponente auf die neue Standardkonfiguration zurücksetzen. Wenn Sie die Komponente bereitstellen, geben Sie eine einzelne leere Zeichenfolge als Reset-Update an. Weitere Informationen finden Sie unter Die Komponentenkonfiguration wird nicht aktualisiert, wenn die Standardkonfiguration geändert wird.

com.aws.greengrass.authorization.exceptions.AuthorizationException: Duplicate policy ID "<id>" for principal "<componentList>"

Dieser Fehler tritt möglicherweise auf, wenn mehrere IPC-Autorisierungsrichtlinien, auch für alle Komponenten auf dem Kerngerät, dieselbe Richtlinien-ID verwenden.

Überprüfen Sie die IPC-Autorisierungsrichtlinien Ihrer Komponenten, korrigieren Sie alle Duplikate und versuchen Sie es erneut. Um eindeutige Richtlinien-IDs zu erstellen, empfehlen wir, den Komponentennamen, den IPC-Dienstnamen und einen Zähler zu kombinieren. Weitere Informationen finden Sie unter Autorisieren Sie Komponenten zur Ausführung von IPC-Vorgängen.

> **Tip**
>
> Wenn Sie das `DefaultConfiguration` in der Rezeptur einer Komponente ändern, müssen Sie die Konfiguration der Komponente auf die neue Standardkonfiguration zurücksetzen. Wenn Sie die Komponente bereitstellen, geben Sie eine einzelne leere Zeichenfolge als Reset-Update an. Weitere Informationen finden Sie unter Die Komponentenkonfiguration wird nicht aktualisiert, wenn die Standardkonfiguration geändert wird.

com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 400)

Dieser Fehler tritt möglicherweise auf, wenn ein Core-Gerät keine AWS Anmeldeinformationen vom Token-Austauschdienst abrufen kann. Der HTTP 400-Statuscode gibt an, dass dieser Fehler aufgetreten ist, weil die Token-Exchange-IAM-Rolle des Kerngeräts nicht existiert oder keine Vertrauensbeziehung besteht, die es dem Anbieter der AWS IoT Anmeldeinformationen ermöglicht, sie anzunehmen.

Gehen Sie wie folgt vor:

1. Identifizieren Sie die Token-Austauschrolle, die das Kerngerät verwendet. Die Fehlermeldung enthält den AWS IoT Rollenalias des Kerngeräts, der auf die Token-Austauschrolle verweist. Führen Sie den folgenden Befehl auf Ihrem Entwicklungscomputer aus und *MyGreengrassCoreTokenExchangeRoleAlias*ersetzen Sie ihn durch den Namen des AWS IoT Rollenalias aus der Fehlermeldung.

   ```
   aws iot describe-role-alias --role-alias MyGreengrassCoreTokenExchangeRoleAlias
   ```

   Die Antwort enthält den Amazon-Ressourcennamen (ARN) der Token-Exchange-IAM-Rolle.

   ```
   {
     "roleAliasDescription": {
       "roleAlias": "MyGreengrassCoreTokenExchangeRoleAlias",
       "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/
   MyGreengrassCoreTokenExchangeRoleAlias",
       "roleArn": "arn:aws:iam::123456789012:role/MyGreengrassV2TokenExchangeRole",
       "owner": "123456789012",
       "credentialDurationSeconds": 3600,
       "creationDate": "2021-02-05T16:46:18.042000-08:00",
       "lastModifiedDate": "2021-02-05T16:46:18.042000-08:00"
     }
   }
   ```

2. Überprüfen Sie, ob die Rolle existiert. Führen Sie den folgenden Befehl aus und ersetzen Sie *MyGreengrassV2 TokenExchangeRole* durch den Namen der Token-Austauschrolle.

   ```
   aws iam get-role --role-name MyGreengrassV2TokenExchangeRole
   ```

   Wenn der Befehl einen `NoSuchEntity` Fehler zurückgibt, ist die Rolle nicht vorhanden und Sie müssen sie erstellen. Weitere Informationen zum Erstellen und Konfigurieren dieser Rolle finden Sie unterAutorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

3. Vergewissern Sie sich, dass die Rolle über eine Vertrauensstellung verfügt, die es dem Anbieter der AWS IoT Anmeldeinformationen ermöglicht, diese zu übernehmen. Die Antwort aus dem vorherigen Schritt enthält eine`AssumeRolePolicyDocument`, die die Vertrauensbeziehungen der Rolle definiert. Die Rolle muss eine Vertrauensbeziehung definieren, die es erlaubt`credentials.iot.amazonaws.com`, sie anzunehmen. Dieses Dokument sollte dem folgenden Beispiel ähneln.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Wenn die Vertrauensbeziehungen der Rolle es nicht zulassen, dies `credentials.iot.amazonaws.com` zu übernehmen, müssen Sie diese Vertrauensbeziehung der Rolle hinzufügen. Weitere Informationen finden Sie unter Ändern einer Rolle im AWS Identity and Access Management IAM-Benutzerhandbuch.

com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 403)

Dieser Fehler tritt möglicherweise auf, wenn ein Kerngerät keine AWS Anmeldeinformationen vom Token-Austauschdienst abrufen kann. Der HTTP-Statuscode 403 gibt an, dass dieser Fehler aufgetreten ist, weil die AWS IoT Richtlinien des Kerngeräts die `iot:AssumeRoleWithCertificate` Berechtigung für den AWS IoT Rollenalias des Kerngeräts nicht gewähren.

Überprüfen Sie die AWS IoT Richtlinien des Kerngeräts und fügen Sie die `iot:AssumeRoleWithCertificate` Berechtigung für den AWS IoT Rollenalias des Kerngeräts hinzu. Die Fehlermeldung enthält den aktuellen AWS IoT Rollenalias des Kerngeräts. Weitere Informationen zu dieser Berechtigung und zur Aktualisierung der AWS IoT Richtlinien des Kerngeräts finden Sie unter Minimale AWS IoT Richtlinie für -AWS IoT Greengrass V2Core-Geräte undAktualisieren der AWS IoT Richtlinie eines Core-Geräts.

com.aws.greengrass.tes.CredentialsProviderError: Could not load credentials from any providers

Dieser Fehler tritt möglicherweise auf, wenn die Komponente versucht, AWS Anmeldeinformationen anzufordern und keine Verbindung zum Token-Austauschdienst herstellen kann.

Gehen Sie wie folgt vor:

- Überprüfen Sie, ob die Komponente eine Abhängigkeit von der Token-Exchange-Dienstkomponente deklariert,`aws.greengrass.TokenExchangeService`. Ist dies nicht der Fall, fügen Sie die Abhängigkeit hinzu und stellen Sie die Komponente erneut bereit.
- Wenn die Komponente im Docker ausgeführt wird, stellen Sie sicher, dass Sie entsprechend die richtigen Netzwerkeinstellungen und Umgebungsvariablen anwenden. Verwenden Sie AWS Anmeldeinformationen in Docker-Container-Komponenten (Linux)
- Wenn die Komponente in NodeJS geschrieben ist, legen Sie dns fest. setDefaultResultBestellung zu. **`ipv4first`**
- `/etc/hosts`Suchen Sie nach einem Eintrag, der mit beginnt `::1` und enthält`localhost`. Entfernen Sie den Eintrag, um festzustellen, ob die Komponente dadurch eine Verbindung mit dem Token-Austauschdienst unter der falschen Adresse hergestellt hat.

Received error when attempting to retrieve ECS metadata: Could not connect to the endpoint URL: "<tokenExchangeServiceEndpoint>"

Dieser Fehler tritt möglicherweise auf, wenn die Komponente den Token-Austauschdienst nicht ausführt und eine Komponente versucht, AWS Anmeldeinformationen anzufordern.

Gehen Sie wie folgt vor:

- Überprüfen Sie, ob die Komponente eine Abhängigkeit von der Token-Exchange-Dienstkomponente deklariert,`aws.greengrass.TokenExchangeService`. Ist dies nicht der Fall, fügen Sie die Abhängigkeit hinzu und stellen Sie die Komponente erneut bereit.
- Prüfen Sie, ob die Komponente `install` während ihres Lebenszyklus AWS Anmeldeinformationen verwendet. AWS IoT Greengrass garantiert nicht die Verfügbarkeit des Token-Austauschdienstes während des `install` Lebenszyklus. Aktualisieren Sie die Komponente, um den Code, der AWS Anmeldeinformationen verwendet, in den `startup` `run` OR-Lebenszyklus zu verschieben, und stellen Sie die Komponente dann erneut bereit.

copyFrom: <configurationPath> is already a container, not a leaf

Dieser Fehler tritt möglicherweise auf, wenn Sie einen Konfigurationswert von einem Containertyp (eine Liste oder ein Objekt) in einen Nicht-Containertyp (eine Zeichenfolge, Zahl oder Boolean) ändern. Gehen Sie wie folgt vor:

1. Prüfen Sie anhand der Rezeptur der Komponente, ob die Standardkonfiguration diesen Konfigurationswert auf eine Liste oder ein Objekt festlegt. Wenn ja, entfernen oder ändern Sie diesen Konfigurationswert.
2. Erstellen Sie eine Bereitstellung, um diesen Konfigurationswert auf seinen Standardwert zurückzusetzen. Weitere Informationen finden Sie unter Erstellen von Bereitstellungen und Komponentenkonfigurationen aktualisieren.

Anschließend können Sie diesen Konfigurationswert auf eine Zeichenfolge, eine Zahl oder einen booleschen Wert festlegen.

com.aws.greengrass.componentmanager.plugins.docker.exceptions.DockerLoginError logging into the registry using credentials - 'The stub received bad data.'

Möglicherweise wird dieser Fehler in den Greengrass-Nukleus-Protokollen angezeigt, wenn die Docker Application Manager-Komponente versucht, ein Docker-Image aus einem privaten Repository in Amazon Elastic Container Registry (Amazon ECR) herunterzuladen. Dieser Fehler tritt auf, wenn Sie den `wincred` Docker Credential Helper () verwenden. `docker-credential-wincred` Daher kann Amazon ECR die Anmeldeinformationen nicht speichern.

Ergreifen Sie eine der folgenden Aktionen:

- Wenn Sie den `wincred` Docker Credential Helper nicht verwenden, entfernen Sie das `docker-credential-wincred` Programm vom Kerngerät.
- Wenn Sie den `wincred` Docker Credential Helper verwenden, gehen Sie wie folgt vor:
  1. Benennen Sie das `docker-credential-wincred` Programm auf dem Core-Gerät um. `wincred`Ersetzen Sie es durch einen neuen Namen für den Windows Docker Credential Helper. Sie können ihn beispielsweise umbenennen in. `docker-credential-wincredreal`
  2. Aktualisieren Sie die `credsStore` Option in der Docker-Konfigurationsdatei (`.docker/config.json`), sodass sie den neuen Namen für den Windows Docker Credential

Helper verwendet. Wenn Sie das Programm beispielsweise in umbenannt habendocker-credential-wincredreal, aktualisieren Sie die credsStore Option auf. wincredreal

```
{
  "credsStore": "wincredreal"
}
```

java.io.IOException: Cannot run program "cmd" ...: [LogonUser] The password for this account has expired.

Dieser Fehler tritt möglicherweise auf einem Windows-Core-Gerät auf, wenn der Systembenutzer, der die Prozesse der Komponente ausführt, z. B. ggc_user ein abgelaufenes Passwort hat. Daher kann die AWS IoT Greengrass Core-Software die Komponentenprozesse nicht als dieser Systembenutzer ausführen.

Um das Passwort eines Greengrass-Systembenutzers zu aktualisieren

1. Führen Sie den folgenden Befehl als Administrator aus, um das Passwort des Benutzers festzulegen. Ersetzen Sie *ggc_user* durch den Systembenutzer und ersetzen Sie *password durch das* festzulegende Passwort.

   ```
   net user ggc_user password
   ```

2. Verwenden Sie das PsExec Hilfsprogramm, um das neue Passwort des Benutzers in der Credential Manager-Instanz für das Konto zu speichern. LocalSystem Ersetzen Sie *das Passwort* durch das Passwort des Benutzers, das Sie festgelegt haben.

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

> **Tip**
>
> Abhängig von Ihrer Windows-Konfiguration ist das Benutzerkennwort möglicherweise so eingestellt, dass es an einem Datum in der future abläuft. Um sicherzustellen, dass Ihre Greengrass-Anwendungen weiterhin funktionieren, verfolgen Sie, wann das Passwort abläuft, und aktualisieren Sie es, bevor es abläuft. Sie können das Benutzerkennwort auch so einrichten, dass es niemals abläuft.

- Führen Sie den folgenden Befehl aus, um zu überprüfen, wann ein Benutzer und sein Passwort ablaufen.

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

- Führen Sie den folgenden Befehl aus, um das Passwort eines Benutzers so einzustellen, dass es nie abläuft.

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

- Wenn Sie Windows 10 oder höher verwenden und der `wmic`Befehl veraltet ist, führen Sie den folgenden PowerShell Befehl aus.

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

aws.greengrass.StreamManager: Instant exceeds minimum or maximum instant

Wenn Sie Stream Manager v2.0.7 auf eine Version zwischen v2.0.8 und v2.0.11 aktualisieren, wird möglicherweise der folgende Fehler in den Protokollen der Stream Manager-Komponente angezeigt, wenn die Komponente 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 Stream Manager v2.0.7 bereitgestellt haben und auf eine neuere Version aktualisieren möchten, müssen Sie direkt auf Stream Manager v2.0.12 aktualisieren. Weitere Hinweise zur Stream Manager-Komponente finden Sie unter. Stream-Manager

Probleme mit den Lambda-Funktionskomponenten des Kerngeräts

Beheben Sie Probleme mit Lambda-Funktionskomponenten auf Kerngeräten.

Themen

The following cgroup subsystems are not mounted: devices, memory

In den folgenden Fällen kann dieser Fehler auftreten, wenn Sie eine containerisierte Lambda-Funktion ausführen:

- Auf dem Core-Gerät ist cgroup v1 für die Speicher- oder Geräte-Cgroups nicht aktiviert.
- Auf dem Kerngerät ist cgroups v2 aktiviert. Greengrass Lambda-Funktionen erfordern cgroups v1, und cgroups v1 und v2 schließen sich gegenseitig aus.

Um cgroups v1 zu aktivieren, starten Sie das Gerät mit den folgenden Linux-Kernelparametern.

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

> **Tip**
>
> Bearbeiten Sie auf einem Raspberry Pi die `/boot/cmdline.txt` Datei, um die Kernel-Parameter des Geräts festzulegen.

ipc_client.py:64,HTTP Error 400:Bad Request, b'No subscription exists for the source <label-or-lambda-arn> and subject <label-or-lambda-arn>

Dieser Fehler wird möglicherweise angezeigt, wenn Sie eine V1-Lambda-Funktion, die das AWS IoT Greengrass Core-SDK verwendet, auf einem V2-Core-Gerät ausführen, ohne ein Abonnement in der älteren Abonnement-Router-Komponente anzugeben. Um dieses Problem zu beheben, stellen Sie den älteren Abonnement-Router bereit und konfigurieren Sie ihn, um die erforderlichen Abonnements anzugeben. Weitere Informationen finden Sie unter Importieren von V1-Lambda-Funktionen.

Die Komponentenversion wurde eingestellt

Möglicherweise wird auf Ihrem Personal Health Dashboard (PHD) eine Benachrichtigung angezeigt, wenn eine Komponentenversion auf Ihrem Kerngerät eingestellt wird. Die Komponentenversion sendet diese Benachrichtigung innerhalb von 60 Minuten nach der Einstellung an Ihren PHD.

Gehen Sie wie folgt vor, um zu sehen, welche Bereitstellungen Sie überarbeiten müssen, indem Sie: AWS Command Line Interface

1. Führen Sie den folgenden Befehl aus, um eine Liste Ihrer Kerngeräte abzurufen.

   ```
   aws greengrassv2 list-core-devices
   ```

2. Führen Sie den folgenden Befehl aus, um den Status der Komponenten auf jedem Kerngerät aus Schritt 1 abzurufen. *coreDeviceName*Ersetzen Sie ihn durch den Namen jedes abzufragenden Kerngeräts.

   ```
   aws greengrassv2 list-installed-components --core-device-thing-name coreDeviceName
   ```

3. Stellen Sie die Kerngeräte zusammen, auf denen die nicht mehr verfügbare Komponentenversion aus den vorherigen Schritten installiert wurde.
4. Führen Sie den folgenden Befehl aus, um den Status aller Bereitstellungsaufträge für jedes Kerngerät aus Schritt 3 abzurufen. *coreDeviceName*Ersetzen Sie ihn durch den Namen des abzufragenden Kerngeräts.

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

   Die Antwort enthält die Liste der Bereitstellungsaufträge für das Kerngerät. Sie können die Bereitstellung überarbeiten, um eine andere Komponentenversion auszuwählen. Weitere Informationen zum Überarbeiten einer Bereitstellung finden Sie unter Bereitstellungen überarbeiten.

Probleme mit der Greengrass-Befehlszeilenschnittstelle

Beheben Sie Probleme mit der Greengrass-CLI.

Themen

- java.lang.RuntimeException: Unable to create ipc client

java.lang.RuntimeException: Unable to create ipc client

Dieser Fehler wird möglicherweise angezeigt, wenn Sie einen Greengrass-CLI-Befehl ausführen und einen anderen Stammordner angeben als den, in dem die AWS IoT Greengrass Core-Software installiert ist.

Gehen Sie wie folgt vor, um den Stammpfad festzulegen, und */greengrass/v2* ersetzen Sie ihn durch den Pfad zu Ihrer AWS IoT Greengrass Core-Softwareinstallation:

- Legen Sie die Umgebungsvariable `GGC_ROOT_PATH` auf */greengrass/v2* fest.
- Fügen Sie das `--ggcRootPath` */greengrass/v2* Argument zu Ihrem Befehl hinzu, wie im folgenden Beispiel gezeigt.

```
greengrass-cli --ggcRootPath /greengrass/v2 <command> <subcommand> [arguments]
```

AWS Command Line Interface Probleme

AWS CLI Probleme beheben für AWS IoT Greengrass V2.

Themen

Error: Invalid choice: 'greengrassv2'

Dieser Fehler wird möglicherweise angezeigt, wenn Sie einen AWS IoT Greengrass V2 Befehl mit AWS CLI (z. B.`aws greengrassv2 list-core-devices`) ausführen.

Dieser Fehler weist darauf hin, dass Sie eine Version von haben AWS CLI , die nicht unterstützt wird AWS IoT Greengrass V2. Um es AWS IoT Greengrass V2 mit dem verwenden zu können AWS CLI, benötigen Sie eine der folgenden Versionen oder höher:

- Mindestversion AWS CLI V1: v1.18.197
- Mindestversion AWS CLI V2: v2.1.11

> **Tip**
>
> Sie können den folgenden Befehl ausführen, um zu überprüfen, AWS CLI welche Version Sie haben.
>
> ```
> aws --version
> ```

Um dieses Problem zu beheben, aktualisieren Sie die AWS CLI auf eine neuere Version, die unterstützt AWS IoT Greengrass V2. Weitere Informationen finden Sie unter Installation, Aktualisierung und Deinstallation von AWS CLI im AWS Command Line Interface Benutzerhandbuch.

Detaillierte Bereitstellungsfehlercodes

Verwenden Sie die Fehlercodes und Lösungen in diesen Abschnitten, um Probleme bei der Komponentenbereitstellung zu lösen, wenn Sie die Greengrass Nucleus Version 2.8.0 oder höher verwenden.

Der Greengrass Nucleus meldet Bereitstellungsfehler in einer Hierarchie vom unspezifischen bis zum spezifischsten verfügbaren Code. Sie können diese Hierarchie verwenden, um den Grund für einen Bereitstellungsfehler zu ermitteln. Im Folgenden finden Sie beispielsweise eine mögliche Fehlerhierarchie:

- BEREITSTELLUNGSFEHLER
  - FEHLER BEIM HERUNTERLADEN VON ARTEFAKTEN
    - IO_ERROR
      - DISK_SPACE_CRITICAL

Die Fehlercodes sind in Typen unterteilt. Jeder Typ steht für eine Klasse von Fehlern, die auftreten können.AWS IoT Greengrassmeldet diese Fehlertypen in der Konsole, der API undAWS CLI. Abhängig von den in der Fehlerhierarchie gemeldeten Fehlern kann es mehr als einen Fehlertyp geben. Für das vorherige Beispiel lautet der zurückgegebene Fehlertyp`DEVICE_ERROR`.

Es gibt folgende Typen:

- ERLAUBNISFEHLER— Der Zugriff auf eine Operation, für die eine Genehmigung erforderlich ist, wurde verweigert.

- ANFORDERUNGSFEHLER— Aufgrund eines Problems im Bereitstellungsdokument ist ein Fehler aufgetreten.
- FEHLER BEIM KOMPONENTENREZEPT— Aufgrund eines Fehlers in einer Komponentenrezeptur ist ein Fehler aufgetreten.
- AWS_COMPONENT_ERROR— Beim Starten oder Entfernen einesAWSbereitgestellte Komponente.
- USER_COMPONENT_ERROR— Beim Starten oder Entfernen einer Benutzerkomponente ist ein Fehler aufgetreten.
- KOMPONENTENFEHLER— Beim Starten oder Entfernen einer Komponente ist ein Fehler aufgetreten, aber der Greengrass-Kernkern konnte nicht feststellen, ob es sich bei der Komponente um eineAWSbereitgestellte Komponente oder eine Benutzerkomponente.
- GERÄTE_FEHLER— Bei der lokalen I/O ist ein Fehler aufgetreten, oder es ist ein anderer Gerätefehler aufgetreten.
- ABHÄNGIGKEITSFEHLER— Ein Deployment konnte ein Artefakt von Amazon S3 nicht herunterladen oder ein Image aus einer ECR-Registrierung abrufen.
- HTTP_ERROR— Bei einer HTTP-Anfrage ist ein Fehler aufgetreten.
- NETZWERKFEHLER— Im Gerätenetzwerk ist ein Fehler aufgetreten.
- NUKLEUS_FEHLER— Der Greengrass-Kern konnte eine Komponente nicht lokalisieren oder die aktive Nucleus-Version nicht finden.
- SERVERFEHLER— Ein Server hat als Antwort auf eine Anfrage einen 500-Fehler zurückgegeben.
- CLOUD_SERVICE_FEHLER— Ein Fehler ist aufgetreten mit derAWS IoT GreengrassCloud-Dienst.
- UNBEKANNTER_FEHLER— Eine ungeprüfte Ausnahme wurde von der Komponente ausgelöst.

Viele der Fehler in diesem Abschnitt enthalten zusätzliche Informationen in derAWS IoT GreengrassKernprotokolle. Diese Protokolle werden im lokalen Dateisystem des Core-Geräts gespeichert. Es gibt Protokolle fürAWS IoT GreengrassKernsoftware und für jede einzelne Komponente. Hinweise zum Zugriff auf die Protokolle finden Sie unterZugriff auf Dateisystemprotokolle.

Berechtigungsfehler

ZUGRIFF_VERWEIGERT

Dieser Fehler kann auftreten, wennAWSDer Dienstvorgang gibt einen 403-Fehler zurück, da die Berechtigungen nicht korrekt eingerichtet sind. Einzelheiten finden Sie im spezifischeren Fehlercode.

GET_DEPLOYMENT_CONFIGURATION_ACCESS DENIED

Möglicherweise erhalten Sie diesen Fehler, wennAWS IoTDie Richtlinie erlaubt nicht die Erlaubnis, die anzurufen`GetDeploymentConfiguration`Betrieb. Füge das hinzu`greengrass::GetDeploymentConfiguration`Erlaubnis zur Richtlinie des Kerngeräts.

GET_COMPONENT_VERSION_ARTIFACT_ACCESS_DENIED

Dieser Fehler kann auftreten, wenn das HauptgerätAWS IoTDie Richtlinie erlaubt das nicht`greengrass:GetComponentVersionArtifact`erlaubnis. Fügen Sie die Erlaubnis zur Richtlinie des Hauptgeräts hinzu.

RESOLVE_COMPONENT_CANDIDATES_ACCESS DENIED

Dieser Fehler kann auftreten, wenn das HauptgerätAWS IoTDie Richtlinie erlaubt das nicht`greengrass:ResolveComponentCandidates`erlaubnis. Fügen Sie die Erlaubnis zur Richtlinie des Hauptgeräts hinzu.

GET_ECR_CREDENTIALS ERROR

Dieser Fehler kann auftreten, wenn das Deployment nicht mit einer privaten Registrierung in ECR authentifiziert werden konnte. Überprüfen Sie das Protokoll auf einen bestimmten Fehler und versuchen Sie dann erneut, die Bereitstellung durchzuführen.

BENUTZER, DER FÜR DOCKER NICHT AUTORISIERT IST

Dieser Fehler kann auftreten, wenn der Greengrass-Benutzer nicht autorisiert ist, Docker zu verwenden. Vergewissern Sie sich, dass Sie Greengrass als Root-Benutzer ausführen oder dass der Benutzer zum`docker`Gruppe. Versuchen Sie dann erneut, das Deployment durchzuführen.

S3_ACCESS_DENIED

Dieser Fehler kann auftreten, wenn ein Amazon S3-Vorgang einen 403-Fehler zurückgibt. Einzelheiten finden Sie in allen zusätzlichen Fehlercodes oder Protokollen.

S3_HEAD_OBJECT_ACCESS_DENIED

Dieser Fehler kann auch auftreten, wenn die Token-Exchange-Rolle des Geräts das nicht zulässtAWS IoT GreengrassKernsoftware zum Herunterladen des Komponentenartefakts von der S3-Objekt-URL, die Sie im Rezept der Komponente angeben oder die besagt, dass das Komponentenartefakt nicht verfügbar ist. Prüfen Sie, ob die Token-Exchange-Rolle dies zulässts3:GetObjectfür die S3-Objekt-URL, unter der das Artefakt verfügbar ist und dass das Artefakt vorhanden ist.

S3_GET_BUCKET_LOCATION_ACCESS_DENIED

Dieser Fehler kann auftreten, wenn die Token-Exchange-Rolle des Geräts das nicht zulässts3:GetBucketLocationGenehmigung für den Amazon S3-Bucket, in dem das Artefakt verfügbar ist. Überprüfen Sie, ob das Gerät die Erlaubnis zulässt, und versuchen Sie dann erneut, die Bereitstellung durchzuführen.

S3_GET_OBJECT_ACCESS_DENIED

Dieser Fehler kann auch auftreten, wenn die Token-Exchange-Rolle des Geräts das nicht zulässtAWS IoT GreengrassKernsoftware zum Herunterladen des Komponentenartefakts von der S3-Objekt-URL, die Sie im Rezept der Komponente angeben oder die besagt, dass das Komponentenartefakt nicht verfügbar ist. Prüfen Sie, ob die Token-Exchange-Rolle dies zulässts3:GetObjectfür die S3-Objekt-URL, unter der das Artefakt verfügbar ist und dass das Artefakt vorhanden ist.

Fehler bei der Anfrage

DEM NUKLEUS FEHLEN DIE ERFORDERLICHEN FÄHIGKEITEN

Dieser Fehler kann auftreten, wenn die Nucleus-Version in der Bereitstellung einen angeforderten Vorgang nicht unterstützt, z. B. das Herunterladen einer großen Konfiguration oder das Festlegen von Linux-Ressourcenlimits. Versuchen Sie die Bereitstellung erneut mit einer Nucleus-Version, die den Vorgang unterstützt.

MEHRER_NUKLEUS_GELÖSTE_FEHLER

Dieser Fehler kann auftreten, wenn ein Deployment versucht, mehrere Nucleus-Komponenten bereitzustellen. Sehen Sie im Protokoll nach, was den Fehler verursacht hat, und schauen Sie dann auf der Nucleus-Softwareupdate-Seite nach, ob das Problem in einer neueren Version von Nucleus behoben wurde, oder wenden Sie sich anAWS Support.

COMPONENT_CIRCULAR_DEPENDENCY_ERROR

Dieser Fehler kann auftreten, wenn zwei Komponenten in Ihrem Deployment voneinander abhängen. Überarbeiten Sie das Komponenten-Setup, sodass die Komponenten in Ihrem Deployment nicht voneinander abhängig sind.

UNAUTORISIERTE_NUCLEUS_MINOR_VERSIONSUPDATE

Dieser Fehler kann auftreten, wenn für eine Komponente in Ihrer Bereitstellung ein Nucleus-Minor-Versionsupdate erforderlich ist, diese Version jedoch nicht in der Bereitstellung angegeben ist. Dies trägt dazu bei, versehentliche kleinere Versionsupdates für Komponenten zu reduzieren, die von einer anderen Version abhängig sind. Nehmen Sie die neue Nucleus-Minor-Version in das Deployment auf.

FEHLENDER_DOCKER_APPLICATION_MANAGER

Dieser Fehler kann auftreten, wenn Sie eine Docker-Komponente bereitstellen, ohne den Docker-Anwendungsmanager bereitzustellen. Stellen Sie sicher, dass Ihr Deployment den Docker-Anwendungsmanager beinhaltet.

FEHLENDER_TOKEN-EXCHANGE-SERVICE

Dieser Fehler kann auftreten, wenn das Deployment ein Docker-Image-Artefakt aus einer privaten ECR-Registrierung herunterladen möchte, ohne den Token-Exchange-Dienst bereitzustellen. Stellen Sie sicher, dass Ihr Deployment den Token-Austauschdienst beinhaltet.

DIE ANFORDERUNGEN AN DIE KOMPONENTENVERSION WERDEN NICHT ERFÜLLT

Dieser Fehler kann auftreten, wenn ein Versionsbeschränkungskonflikt vorliegt oder eine Komponentenversion nicht existiert. Weitere Informationen finden Sie unter Error: com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException: Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements>.

DROSSELUNGSFEHLER

Dieser Fehler kann auftreten, wennAWSDer Servicebetrieb hat ein Tarifkontingent überschritten. Wiederholen Sie die Bereitstellung.

WIDERSPRÜCHLICHE_ANFRAGE

Dieser Fehler kann auftreten, wennAWSDer Dienstvorgang gibt einen 409-Fehler zurück, da Ihr Deployment versucht, mehr als einen Vorgang gleichzeitig auszuführen. Wiederholen Sie die Bereitstellung.

RESOURCE_NICHT_GEFUNDEN

Dieser Fehler kann auftreten, wennAWSDer Servicevorgang gibt einen 404-Fehler zurück, da eine Ressource nicht gefunden werden konnte. Überprüfen Sie das Protokoll auf die fehlende Ressource.

RUN_WITH_CONFIG_NOT_VALID

Möglicherweise erhalten Sie diesen Fehler, wenn`posixUser`,`posixGroup`, oder`windowsUser`Die zur Ausführung der Komponente angegebenen Informationen sind nicht gültig. Überprüfen Sie, ob der Benutzer gültig ist, und versuchen Sie dann erneut, die Bereitstellung durchzuführen.

NICHT UNTERSTÜTZTE_REGION

Dieser Fehler kann auftreten, wenn die für die Bereitstellung angegebene Region nicht unterstützt wird vonAWS IoT Greengrass. Überprüfen Sie die Region und versuchen Sie die Bereitstellung erneut.

IOT_CRED_ENDPOINT_NOT_VALID

Möglicherweise erhalten Sie diesen Fehler, wennAWS IoTDer in der Konfiguration angegebene Endpunkt der Anmeldeinformationen ist nicht gültig. Überprüfen Sie den Endpunkt und versuchen Sie es erneut mit Ihrer Anfrage.

IOT-DATENENDPUNKT IST NICHT GÜLTIG

Möglicherweise erhalten Sie diesen Fehler, wennAWS IoTDer in der Konfiguration angegebene Datenendpunkt ist nicht gültig. Überprüfen Sie den Endpunkt und versuchen Sie es erneut mit Ihrer Anfrage.

S3_HEAD_OBJECT_RESOURCE_NOT_FOUND

Dieser Fehler kann auftreten, wenn das Komponentenartefakt unter der S3-Objekt-URL, die Sie im Rezept der Komponente angeben, nicht verfügbar ist. Vergewissern Sie sich, dass Sie das Artefakt in den S3-Bucket hochgeladen haben und ob die Artefakt-URI mit der S3-Objekt-URL des Artefakts im Bucket übereinstimmt.

S3_GET_BUCKET_LOCATION_RESOURCE_NOT_FOUND

Dieser Fehler kann auftreten, wenn der Amazon S3-Bucket nicht gefunden wird. Überprüfen Sie, ob der Bucket vorhanden ist, und versuchen Sie erneut, das Deployment durchzuführen.

S3_GET_OBJECT_RESOURCE_NOT_FOUND

Dieser Fehler kann auftreten, wenn das Komponentenartefakt unter der S3-Objekt-URL, die Sie im Rezept der Komponente angeben, nicht verfügbar ist. Vergewissern Sie sich, dass Sie das Artefakt in den S3-Bucket hochgeladen haben und ob die Artefakt-URI mit der S3-Objekt-URL des Artefakts im Bucket übereinstimmt.

IO_MAPPING_ERROR

Dieser Fehler kann auftreten, wenn beim Parsen des Bereitstellungsdokuments oder Rezepts ein I/O-Fehler auftritt. Einzelheiten finden Sie in allen zusätzlichen Fehlercodes oder Protokollen.

Fehler im Komponentenrezept

FEHLER BEIM PARSEN DES REZEPTS

Dieser Fehler kann auftreten, wenn das Bereitstellungsrezept nicht analysiert werden konnte, weil ein Fehler in der Struktur des Rezepts vorliegt. Vergewissern Sie sich, dass das Rezept korrekt formatiert ist, und versuchen Sie die Bereitstellung erneut.

FEHLER BEIM PARSEN VON REZEPTMETADATEN

Dieser Fehler kann auftreten, wenn die aus der Cloud heruntergeladenen Metadaten des Bereitstellungsrezepts nicht analysiert werden konnten. Wenden Sie sich an AWS Support.

ARTIFACT_URI_NICHT_GÜLTIG

Dieser Fehler kann auftreten, wenn ein Artefakt-URI in einem Rezept nicht richtig formatiert ist. Überprüfen Sie das Protokoll auf die ungültige URI, aktualisieren Sie die URI im Rezept und versuchen Sie dann erneut, die Bereitstellung durchzuführen.

S3_ARTIFACT_URI_NICHT_GÜLTIG

Dieser Fehler kann auftreten, wenn die Amazon S3-URI eines Artefakts in einem Rezept nicht gültig ist. Überprüfen Sie das Protokoll auf die ungültige URI, aktualisieren Sie die URI im Rezept und versuchen Sie dann erneut, die Bereitstellung durchzuführen.

DOCKER_ARTIFACT_URI_NICHT_GÜLTIG

Dieser Fehler kann auftreten, wenn die Docker-URI eines Artefakts in einem Rezept nicht gültig ist. Überprüfen Sie das Protokoll auf die ungültige URI, aktualisieren Sie die URI im Rezept und versuchen Sie dann erneut, die Bereitstellung durchzuführen.

EMPTY_ARTIFACT_URI

Dieser Fehler kann auftreten, wenn die URI eines Artefakts in einem Rezept nicht angegeben ist. Suchen Sie im Protokoll nach dem Artefakt, für das eine URI fehlt, aktualisieren Sie die URI im Rezept und versuchen Sie dann erneut, die Bereitstellung durchzuführen.

LEERES ARTEFAKTSCHEMA

Dieser Fehler kann auftreten, wenn kein URI-Schema für ein Artefakt definiert ist. Überprüfen Sie das Protokoll auf die ungültige URI, aktualisieren Sie die URI im Rezept und versuchen Sie dann erneut, die Bereitstellung durchzuführen.

NICHT UNTERSTÜTZTES ARTEFAKTSCHEMA

Dieser Fehler kann auftreten, wenn ein URI-Schema von der laufenden Nucleus-Version nicht unterstützt wird. Entweder ist eine URI nicht gültig oder Sie müssen die Nucleus-Version aktualisieren. Wenn die URI nicht gültig ist, überprüfen Sie das Protokoll auf die ungültige URI, aktualisieren Sie die URI im Rezept und versuchen Sie dann erneut, die Bereitstellung durchzuführen.

REZEPT_FEHLENDES_MANIFEST

Dieser Fehler kann auftreten, wenn der Manifestabschnitt nicht im Rezept enthalten ist. Fügen Sie das Manifest zum Rezept hinzu und versuchen Sie erneut, die Bereitstellung durchzuführen.

REZEPT_FEHLENDER_ARTEFAKT-HASH-ALGORITHMUS

Dieser Fehler kann auftreten, wenn in einem Rezept ohne Hash-Algorithmus ein Artefakt angegeben wird, das nicht lokal ist. Fügen Sie den Algorithmus zum Artefakt hinzu und versuchen Sie es erneut mit der Anfrage.

ARTIFACT_CHECKSUM_MISMATCH

Dieser Fehler kann auftreten, wenn ein heruntergeladenes Artefakt einen anderen Digest als den im Rezept angegebenen hat. Stellen Sie sicher, dass das Rezept den richtigen Digest enthält, und versuchen Sie dann erneut, das Deployment durchzuführen. Weitere Informationen finden Sie unter Error: com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchException: Integrity check for downloaded artifact failed. Probably due to file corruption..

KOMPONENTENABHÄNGIGKEIT_NICHT GÜLTIG

Dieser Fehler kann auftreten, wenn der in einem Bereitstellungsrezept angegebene Abhängigkeitstyp nicht gültig ist. Überprüfe das Rezept und versuche es dann erneut mit deiner Anfrage.

CONFIG_INTERPOLATE_ERROR

Dieser Fehler kann auftreten, wenn Sie eine Rezeptvariable interpolieren. Einzelheiten finden Sie im Protokoll.

IO_MAPPING_ERROR

Dieser Fehler kann auftreten, wenn beim Parsen des Bereitstellungsdokuments oder Rezepts ein I/O-Fehler auftritt. Einzelheiten finden Sie in allen zusätzlichen Fehlercodes oder Protokollen.

AWSKomponentenfehler, Benutzerkomponentenfehler, Komponentenfehler

Die folgenden Fehlercodes werden zurückgegeben, wenn ein Problem mit einer Komponente auftritt. Der tatsächlich gemeldete Fehlertyp hängt von der spezifischen Komponente ab, die den Fehler ausgelöst hat. Wenn der Greengrass-Kern die Komponente als eine Komponente identifiziert, die bereitgestellt wird vonAWS IoT Greengrass, es kehrt zurück`AWS_COMPONENT_ERROR`. Wenn die Komponente als Benutzerkomponente identifiziert wird, kehrt der Greengrass-Kern zurück`USER_COMPONENT_ERROR`. Wenn der Greengrass-Kern es nicht weiß, kehrt er zurück`COMPONENT_ERROR`.

FEHLER BEIM KOMPONENTEN-UPDATE

Dieser Fehler kann auftreten, wenn eine Komponente während einer Bereitstellung nicht aktualisiert wird. Überprüfen Sie alle zusätzlichen Fehlercodes oder sehen Sie im Protokoll nach, was den Fehler verursacht hat.

COMPONENT_DEFEKT

Dieser Fehler kann auftreten, wenn eine Komponente während einer Bereitstellung defekt ist. Überprüfen Sie das Komponentenprotokoll auf Fehlerdetails und versuchen Sie dann erneut, die Bereitstellung durchzuführen.

ENTFERNEN_KOMPONENTENFEHLER

Dieser Fehler kann auftreten, wenn der Nucleus eine Komponente während einer Bereitstellung nicht entfernen kann. Überprüfen Sie das Protokoll auf Fehlerdetails und versuchen Sie dann erneut, die Bereitstellung durchzuführen.

COMPONENT_BOOTSTRAP_TIMEOUT

Dieser Fehler kann auftreten, wenn die Bootstrap-Aufgabe einer Komponente länger als das konfigurierte Timeout dauerte. Erhöhen Sie das Timeout oder reduzieren Sie die Ausführungszeit der Bootstrap-Aufgabe, und versuchen Sie dann erneut, die Bereitstellung durchzuführen.

COMPONENT_BOOTSTRAP_ERROR

Dieser Fehler kann auftreten, wenn die Bootstrap-Aufgabe einer Komponente einen Fehler aufweist. Überprüfen Sie das Protokoll auf Fehlerdetails, und versuchen Sie dann erneut, die Bereitstellung durchzuführen.

KOMPONENTENKONFIGURATION IST NICHT GÜLTIG

Dieser Fehler kann auftreten, wenn der Nucleus die bereitgestellte Konfiguration für die Komponente nicht überprüfen kann. Überprüfen Sie das Protokoll auf Fehlerdetails, und versuchen Sie dann erneut, die Bereitstellung durchzuführen.

Gerätefehler

IO_WRITE_ERROR

Dieser Fehler kann auftreten, wenn Sie in eine Datei schreiben. Einzelheiten finden Sie im Protokoll.

IO_READ_ERROR

Dieser Fehler kann auftreten, wenn Sie aus einer Datei lesen. Einzelheiten finden Sie im Protokoll.

DISK_SPACE_CRITICAL

Dieser Fehler kann auftreten, wenn nicht genügend Speicherplatz zur Verfügung steht, um eine Bereitstellungsanfrage abzuschließen. Sie müssen über mindestens 20 MB freien Speicherplatz oder ausreichend Speicherplatz für ein größeres Artefakt verfügen. Geben Sie Speicherplatz frei und versuchen Sie dann erneut, die Bereitstellung durchzuführen.

IO_FILE_ATTRIBUTE_ERROR

Dieser Fehler kann auftreten, wenn die vorhandene Dateigröße nicht aus dem Dateisystem abgerufen werden kann. Einzelheiten finden Sie im Protokoll.

SET_PERMISSION_ERROR

Dieser Fehler kann auftreten, wenn die Berechtigungen für ein heruntergeladenes Artefakt oder Artefaktverzeichnis nicht festgelegt werden können. Einzelheiten finden Sie im Protokoll.

IO_UNZIP_ERROR

Dieser Fehler kann auftreten, wenn ein Artefakt nicht entpackt werden kann. Einzelheiten finden Sie im Protokoll.

LOKALES_REZEPT_NICHT_GEFUNDEN

Dieser Fehler kann auftreten, wenn die lokale Kopie einer Rezeptdatei nicht gefunden werden konnte. Versuchen Sie das Deployment erneut.

LOCAL_RECIPE_BROKEN

Dieser Fehler kann auftreten, wenn sich die lokale Kopie des Rezepts seit dem Herunterladen geändert hat. Löschen Sie die vorhandene Kopie des Rezepts und versuchen Sie erneut, das Deployment durchzuführen.

DIE METADATEN DES LOKALEN REZEPTS WURDEN NICHT GEFUNDEN

Dieser Fehler kann auftreten, wenn die lokale Kopie der Rezept-Metadatendatei nicht gefunden werden konnte. Versuchen Sie das Deployment erneut.

LAUNCH_DIRECTORY_BROKEN

Dieser Fehler kann auftreten, wenn das Verzeichnis, das zum Starten des Greengrass-Nucleus verwendet wurde (`/greengrass/v2/alts/current`) wurde seit dem letzten Start des Kerns modifiziert. Starten Sie den Nucleus neu und versuchen Sie dann erneut, die Bereitstellung durchzuführen.

HASHING-ALGORITHMUS_NICHT VERFÜGBAR

Dieser Fehler kann auftreten, wenn die Java-Distribution des Geräts den erforderlichen Hash-Algorithmus nicht unterstützt oder wenn der in einem Komponentenrezept angegebene Hash-Algorithmus nicht gültig ist.

DIE GERÄTEKONFIGURATION IST FÜR DEN DOWNLOAD DES ARTEFAKTS NICHT GÜLTIG

Dieser Fehler kann auftreten, wenn ein Fehler in der Gerätekonfiguration vorliegt, der das Deployment daran gehindert hat, das Artefakt aus Amazon S3 oder der Greengrass-Cloud herunterzuladen. Überprüfen Sie das Protokoll auf einen bestimmten Konfigurationsfehler und versuchen Sie dann erneut, die Bereitstellung durchzuführen.

Abhängigkeitsfehler

DOCKER_ERROR

Dieser Fehler kann auftreten, wenn Sie ein Docker-Image abrufen. Einzelheiten finden Sie in allen zusätzlichen Fehlercodes oder Protokollen.

DOCKER_SERVICE_NICHT VERFÜGBAR

Dieser Fehler kann auftreten, wenn Greengrass sich nicht in die Docker-Registry einloggen konnte. Überprüfen Sie das Protokoll auf einen bestimmten Fehler und versuchen Sie dann erneut, die Bereitstellung durchzuführen.

DOCKER_LOGIN_ERROR

Dieser Fehler kann auftreten, wenn bei der Anmeldung bei Docker ein unerwarteter Fehler auftritt. Überprüfen Sie das Protokoll auf einen bestimmten Fehler und versuchen Sie dann erneut, die Bereitstellung durchzuführen.

DOCKER_PULL_ERROR

Dieser Fehler kann auftreten, wenn beim Abrufen eines Docker-Images aus der Registrierung ein unerwarteter Fehler auftritt. Überprüfen Sie das Protokoll auf einen bestimmten Fehler und versuchen Sie dann erneut, die Bereitstellung durchzuführen.

DOCKER_IMAGE_NOT_VALID

Möglicherweise wird dieser Fehler angezeigt, wenn das angeforderte Docker-Image nicht existiert. Überprüfen Sie das Protokoll auf einen bestimmten Fehler und versuchen Sie die Bereitstellung erneut.

DOCKER_IMAGE_QUERY_ERROR

Dieser Fehler kann auftreten, wenn bei der Abfrage von Docker nach verfügbaren Images ein unerwarteter Fehler auftritt. Überprüfen Sie das Protokoll auf den spezifischen Fehler und versuchen Sie die Bereitstellung erneut.

S3_FEHLER

Dieser Fehler kann auftreten, wenn Sie ein Amazon S3-Artefakt herunterladen. Einzelheiten finden Sie in allen zusätzlichen Fehlercodes oder Protokollen.

S3_RESOURCE_NICHT_GEFUNDEN

Dieser Fehler kann auftreten, wenn ein Amazon S3-Vorgang einen 404-Fehler zurückgibt. Einzelheiten finden Sie in allen zusätzlichen Fehlercodes oder Protokollen.

S3_BAD_ANFRAGE

Dieser Fehler kann auftreten, wenn ein Amazon S3-Vorgang einen 400-Fehler zurückgibt. Überprüfen Sie das Protokoll auf einen bestimmten Fehler und versuchen Sie die Anfrage erneut.

HTTP-Fehler

HTTP_REQUEST_ERROR

Dieser Fehler kann auftreten, wenn bei einer HTTP-Anfrage ein Fehler aufgetreten ist. Überprüfen Sie das Protokoll auf den spezifischen Fehler.

FEHLER BEIM HERUNTERLADEN DES DEPLOYMENT-DOKUMENTS

Dieser Fehler kann auftreten, wenn beim Herunterladen des Bereitstellungsdokuments ein HTTP-Fehler aufgetreten ist. Überprüfen Sie das Protokoll auf den spezifischen HTTP-Fehler.

GET_GREENGRASS_ARTIFACT_SIZE_ERROR

Dieser Fehler kann auftreten, wenn beim Abrufen der Größe eines Artefakts einer öffentlichen Komponente ein HTTP-Fehler aufgetreten ist. Überprüfen Sie das Protokoll auf den spezifischen HTTP-Fehler.

DOWNLOAD_GREENGRASS_ARTIFACT_ERROR

Dieser Fehler kann auftreten, wenn beim Herunterladen eines Artefakts einer öffentlichen Komponente ein HTTP-Fehler aufgetreten ist. Überprüfen Sie das Protokoll auf den spezifischen HTTP-Fehler.

Netzwerkfehler

NETZWERKFEHLER

Dieser Fehler kann auftreten, wenn während einer Bereitstellung ein Verbindungsproblem auftritt. Überprüfen Sie die Verbindung des Geräts mit dem Internet und versuchen Sie die Bereitstellung erneut.

Kernfehler

SCHLECHTE_ANFRAGE

Dieser Fehler kann auftreten, wennAWSDer Cloud-Vorgang gibt einen 400-Fehler zurück. Prüfen Sie im Protokoll, welche API den Fehler verursacht hat, und schauen Sie dann auf der Nucleus-Softwareupdate-Seite nach, ob das Problem in einer späteren Version von Nucleus behoben wurde, oder wenden Sie sich anAWS Support.

NUCLEUS_VERSION_NICHT_GEFUNDEN

Dieser Fehler kann auftreten, wenn ein Core-Gerät die Version des aktiven Kerns nicht finden kann. Sehen Sie im Protokoll nach, was den Fehler verursacht hat, und schauen Sie dann auf der Nucleus-Softwareupdate-Seite nach, ob das Problem in einer neueren Version von Nucleus behoben wurde, oder wenden Sie sich anAWS Support.

FEHLER BEIM NEUSTART DES KERNS

Dieser Fehler kann auftreten, wenn der Nucleus während einer Bereitstellung, die einen Nucleus-Neustart erfordert, nicht neu gestartet wird. Sehen Sie im Loader-Protokoll nach, was den Fehler verursacht hat, und schauen Sie dann auf der Nucleus-Softwareupdate-Seite nach, ob das Problem in einer neueren Version von Nucleus behoben wurde, oder wenden Sie sich anAWS Support.

DIE INSTALLIERTE KOMPONENTE WURDE NICHT GEFUNDEN

Dieser Fehler kann auftreten, wenn der Nucleus eine installierte Komponente nicht finden kann. Sehen Sie im Protokoll nach, was den Fehler verursacht hat, und schauen Sie dann auf der Nucleus-Softwareupdate-Seite nach, ob das Problem in einer neueren Version von Nucleus behoben wurde, oder wenden Sie sich anAWS Support.

BEREITSTELLUNGSDOKUMENT_NICHT GÜLTIG

Dieser Fehler kann auftreten, wenn das Gerät ein ungültiges Bereitstellungsdokument empfängt. Überprüfen Sie alle zusätzlichen Fehlercodes oder sehen Sie im Protokoll nach, was den Fehler verursacht hat.

LEERE_BEREITSTELLUNGSANFRAGE

Dieser Fehler kann auftreten, wenn ein Gerät eine leere Bereitstellungsanfrage erhält. Sehen Sie im Protokoll nach, was den Fehler verursacht hat, und schauen Sie dann auf der Nucleus-Softwareupdate-Seite nach, ob das Problem in einer neueren Version von Nucleus behoben wurde, oder wenden Sie sich anAWS Support.

FEHLER BEIM PARSEN DES BEREITSTELLUNGSDOKUMENTS

Dieser Fehler kann auftreten, wenn das Format der Bereitstellungsanforderung nicht dem erwarteten Format entspricht. Sehen Sie im Protokoll nach, was den Fehler verursacht hat, und schauen Sie dann auf der Nucleus-Softwareupdate-Seite nach, ob das Problem in einer neueren Version von Nucleus behoben wurde, oder wenden Sie sich anAWS Support.

KOMPONENTEN-METADATEN SIND BEI DER BEREITSTELLUNG NICHT GÜLTIG

Dieser Fehler kann auftreten, wenn die Bereitstellungsanforderung ungültige Komponentenmetadaten enthält. Sehen Sie im Protokoll nach, was den Fehler verursacht hat, und schauen Sie dann auf der Nucleus-Softwareupdate-Seite nach, ob das Problem in einer neueren Version von Nucleus behoben wurde, oder wenden Sie sich anAWS Support.

LAUNCH_DIRECTORY_BROKEN

Dieser Fehler kann auftreten, wenn Sie ein Greengrass-Gerät von einer Dinggruppe in eine andere verschieben und dann bei Bereitstellungen, für die Greengrass neu gestartet werden muss, zur ursprünglichen Gruppe zurückkehren. Um den Fehler zu beheben, erstellen Sie das Startverzeichnis für Greengrass auf dem Gerät neu.

Weitere Informationen finden Sie unter Error: com.aws.greengrass.deployment.exceptions.DeploymentException: Unable to process deployment. Greengrass launch directory is not set up or Greengrass is not set up as a system service.

Serverfehler

SERVERFEHLER

Dieser Fehler kann auftreten, wennAWSDer Servicevorgang gibt einen 500-Fehler zurück, da der Dienst die Anfrage derzeit nicht verarbeiten kann. Versuchen Sie das Deployment später erneut.

S3_SERVERFEHLER

Dieser Fehler kann auftreten, wenn ein Amazon S3-Vorgang einen 500-Fehler zurückgibt. Einzelheiten finden Sie in allen zusätzlichen Fehlercodes oder Protokollen.

Cloud-Dienstfehler

RESOLVE_COMPONENT_CANDIDATES_BAD_RESPONSE

Dieser Fehler kann auftreten, wenn der Greengrass Cloud-Dienst eine inkompatible Antwort an die`ResolveComponentCandidates`Betrieb. Sehen Sie im Protokoll nach, was den Fehler verursacht hat, und schauen Sie dann auf der Nucleus-Softwareupdate-Seite nach, ob das Problem in einer neueren Version von Nucleus behoben wurde, oder wenden Sie sich anAWS Support.

DIE GRÖSSE DES BEREITSTELLUNGSDOKUMENTS WURDE ÜBERSCHRITTEN

Dieser Fehler kann auftreten, wenn das angeforderte Bereitstellungsdokument das maximale Größenkontingent überschreitet. Reduzieren Sie die Größe des Bereitstellungsdokuments und versuchen Sie erneut, die Bereitstellung durchzuführen.

DIE GRÖSSE DES GREENGRASS-ARTEFAKTS WURDE NICHT GEFUNDEN

Dieser Fehler kann auftreten, wenn Greengrass die Größe eines Artefakts einer öffentlichen Komponente nicht ermitteln kann. Sehen Sie im Protokoll nach, was den Fehler verursacht hat, und schauen Sie dann auf der Nucleus-Softwareupdate-Seite nach, ob das Problem in einer neueren Version von Nucleus behoben wurde, oder wenden Sie sich anAWS Support.

BEREITSTELLUNGSDOKUMENT_NICHT GÜLTIG

Dieser Fehler kann auftreten, wenn das Gerät ein ungültiges Bereitstellungsdokument empfängt. Überprüfen Sie alle zusätzlichen Fehlercodes oder sehen Sie im Protokoll nach, was den Fehler verursacht hat.

LEERE_BEREITSTELLUNGSANFRAGE

Dieser Fehler kann auftreten, wenn ein Gerät eine leere Bereitstellungsanfrage erhält. Sehen Sie im Protokoll nach, was den Fehler verursacht hat, und schauen Sie dann auf der Nucleus-Softwareupdate-Seite nach, ob das Problem in einer neueren Version von Nucleus behoben wurde, oder wenden Sie sich anAWS Support.

FEHLER BEIM PARSEN DES BEREITSTELLUNGSDOKUMENTS

Dieser Fehler kann auftreten, wenn das Format der Bereitstellungsanforderung nicht dem erwarteten Format entspricht. Sehen Sie im Protokoll nach, was den Fehler verursacht hat, und schauen Sie dann auf der Nucleus-Softwareupdate-Seite nach, ob das Problem in einer neueren Version von Nucleus behoben wurde, oder wenden Sie sich anAWS Support.

KOMPONENTEN-METADATEN SIND BEI DER BEREITSTELLUNG NICHT GÜLTIG

Dieser Fehler kann auftreten, wenn die Bereitstellungsanforderung ungültige Komponentenmetadaten enthält. Sehen Sie im Protokoll nach, was den Fehler verursacht hat, und schauen Sie dann auf der Nucleus-Softwareupdate-Seite nach, ob das Problem in einer neueren Version von Nucleus behoben wurde, oder wenden Sie sich anAWS Support.

Generische Fehler

Diesen generischen Fehlern ist kein entsprechender Fehlertyp zugeordnet.

BEREITSTELLUNG_UNTERBROCHEN

Dieser Fehler kann auftreten, wenn eine Bereitstellung aufgrund eines Nucleus-Shutdowns oder eines anderen externen Ereignisses nicht abgeschlossen werden kann. Einzelheiten finden Sie in allen zusätzlichen Fehlercodes oder Protokollen.

FEHLER BEIM HERUNTERLADEN VON ARTEFAKTEN

Dieser Fehler kann auftreten, wenn beim Herunterladen eines Artefakts ein Problem auftritt. Einzelheiten finden Sie in allen zusätzlichen Fehlercodes oder Protokollen.

KEINE_VERFÜGBARE_KOMPONENTENVERSION

Dieser Fehler kann auftreten, wenn eine Komponentenversion nicht in der Cloud oder lokal existiert oder wenn ein Konflikt zur Lösung von Abhängigkeiten besteht. Einzelheiten finden Sie in allen zusätzlichen Fehlercodes oder Protokollen.

FEHLER BEIM LADEN DES KOMPONENTENPAKETS

Dieser Fehler kann auftreten, wenn bei der Verarbeitung der heruntergeladenen Artefakte ein Fehler auftritt. Einzelheiten finden Sie in allen zusätzlichen Fehlercodes oder Protokollen.

CLOUD_API_ERROR

Möglicherweise erhalten Sie diesen Fehler, wenn ein Fehler beim Aufrufen einesAWSDienst-API. Einzelheiten finden Sie in allen zusätzlichen Fehlercodes oder Protokollen.

IO_ERROR

Dieser Fehler kann auftreten, wenn während einer Bereitstellung ein I/O-Fehler auftritt. Einzelheiten finden Sie in allen zusätzlichen Fehlercodes oder Protokollen.

FEHLER BEIM KOMPONENTEN-UPDATE

Dieser Fehler kann auftreten, wenn eine Komponente während einer Bereitstellung nicht aktualisiert wird. Überprüfen Sie alle zusätzlichen Fehlercodes oder sehen Sie im Protokoll nach, was den Fehler verursacht hat.

Unbekannter Fehler

BEREITSTELLUNGSFEHLER

Dieser Fehler kann auftreten, wenn eine Bereitstellung fehlschlägt, weil eine ungeprüfte Ausnahme ausgelöst wurde. Sehen Sie im Protokoll nach, was den Fehler verursacht hat, und

schauen Sie dann auf der Nucleus-Softwareupdate-Seite nach, ob das Problem in einer neueren Version von Nucleus behoben wurde, oder wenden Sie sich anAWS Support.

BEREITSTELLUNGSTYP IST NICHT GÜLTIG

Dieser Fehler kann auftreten, wenn der Bereitstellungstyp nicht gültig ist. Sehen Sie im Protokoll nach, was den Fehler verursacht hat, und schauen Sie dann auf der Nucleus-Softwareupdate-Seite nach, ob das Problem in einer neueren Version von Nucleus behoben wurde, oder wenden Sie sich anAWS Support.

Detaillierte Komponenten-Statuscodes

Verwenden Sie die Statuscodes und Lösungen in diesen Abschnitten, um Probleme mit Komponenten zu lösen, wenn Sie die Greengrass Nucleus Version 2.8.0 oder höher verwenden.

Viele der Statusangaben in diesem Thema enthalten zusätzliche Informationen in den AWS IoT Greengrass Core-Protokollen. Diese Protokolle werden im lokalen Dateisystem des Core-Geräts gespeichert. Für jede einzelne Komponente gibt es Protokolle. Hinweise zum Zugriff auf die Protokolle finden Sie unterZugriff auf Dateisystemprotokolle.

INSTALLATIONSFEHLER

Dies kann auftreten, wenn beim Ausführen eines Installationsskripts ein Fehler auftritt. Der Fehlercode wird im Komponentenprotokoll gemeldet. Überprüfen Sie das Installationsskript auf Fehler und stellen Sie Ihre Komponente erneut bereit.

INSTALL_CONFIG_NOT_VALID

Dieser Fehler kann auftreten, wenn die Installation einer Komponente nicht abgeschlossen werden konnte, weil der `Install` Abschnitt des Rezepts nicht gültig ist. Überprüfen Sie den Installationsabschnitt Ihres Rezepts auf Fehler und versuchen Sie erneut, das Deployment durchzuführen.

INSTALL_IO_ERROR

Dies kann auftreten, wenn bei der Installation einer Komponente ein I/O-Fehler aufgetreten ist. Details zu dem Fehler können Sie der Fehler entnehmen.

INSTALL_MISSING_DEFAULT_RUNWITH

Dieser Fehler kann auftreten, wenn Sie den Benutzer oder die Gruppe nicht ermitteln AWS IoT Greengrass können, die für die Installation einer Komponente verwendet werden soll.

Vergewissern Sie sich, dass der `runWith` Abschnitt Ihres Installationsrezepts einen gültigen Benutzer oder eine gültige Gruppe enthält.

INSTALL_TIMEOUT

Dieser Fehler kann auftreten, wenn das Installationsskript nicht innerhalb des konfigurierten Timeout-Zeitraums abgeschlossen wurde. Erhöhen Sie entweder den im `Install` Abschnitt des Rezepts angegebenen `Timeout` Zeitraum oder ändern Sie Ihr Installationsskript so, dass es innerhalb des konfigurierten Timeouts abgeschlossen wird.

STARTFEHLER

Dies kann auftreten, wenn beim Ausführen eines Startskripts ein Fehler auftritt. Der Fehlercode wird im Komponentenprotokoll gemeldet. Überprüfen Sie das Installationsskript auf Fehler und stellen Sie Ihre Komponente erneut bereit.

STARTUP_CONFIG_NICHT_GÜLTIG

Dieser Fehler kann auftreten, wenn die Installation einer Komponente nicht abgeschlossen werden konnte, weil der `Startup` Abschnitt des Rezepts nicht gültig ist. Überprüfen Sie den Startabschnitt Ihres Rezepts auf Fehler und versuchen Sie die Bereitstellung erneut.

STARTUP_IO_ERROR

Dies kann auftreten, wenn beim Start einer Komponente ein I/O-Fehler aufgetreten ist. Details zu dem Fehler können Sie der Fehler entnehmen.

STARTUP_MISSING_DEFAULT_RUNWITH

Dieser Fehler kann auftreten, wenn Sie den Benutzer oder die Gruppe nicht ermitteln AWS IoT Greengrass können, die für die Ausführung einer Komponente verwendet werden soll. Vergewissern Sie sich, dass der `runWith` Abschnitt Ihres Startrezepts einen gültigen Benutzer oder eine gültige Gruppe enthält.

STARTUP_TIMEOUT

Dieser Fehler kann auftreten, wenn das Startskript nicht innerhalb des konfigurierten Timeout-Zeitraums abgeschlossen wurde. Erhöhen Sie entweder den im `Startup` Abschnitt des Rezepts angegebenen `Timeout` Zeitraum oder ändern Sie Ihr Startskript so, dass es innerhalb des konfigurierten Timeouts abgeschlossen wird.

RUN_ERROR

Dies kann auftreten, wenn beim Ausführen eines Komponentenskripts ein Fehler auftritt. Der Fehlercode wird im Komponentenprotokoll gemeldet. Überprüfen Sie das Run-Skript auf Fehler und stellen Sie Ihre Komponente erneut bereit.

RUN_MISSING_DEFAULT_RUNWITH

Dieser Fehler kann auftreten, wenn Sie den Benutzer oder die Gruppe nicht ermitteln AWS IoT Greengrass können, die für die Ausführung einer Komponente verwendet werden soll. Vergewissere dich, dass der `runWith` Abschnitt deines Run-Rezepts einen gültigen Benutzer oder eine gültige Gruppe enthält.

RUN_CONFIG_NOT_VALID

Dieser Fehler kann auftreten, wenn eine Komponente nicht ausgeführt werden konnte, weil der `Run` Abschnitt des Rezepts nicht gültig ist. Überprüfen Sie den Abschnitt „Ausführen“ Ihres Rezepts auf Fehler und versuchen Sie die Bereitstellung erneut.

RUN_IO_ERROR

Dies kann auftreten, wenn während der Ausführung der Komponente ein I/O-Fehler aufgetreten ist. Details zu dem Fehler können Sie der Fehler entnehmen.

RUN_TIMEOUT

Dieser Fehler kann auftreten, wenn das Ausführungsskript nicht innerhalb des konfigurierten Timeout-Zeitraums abgeschlossen wurde. Erhöhen Sie entweder den im `Run` Abschnitt des Rezepts angegebenen `Timeout` Zeitraum oder ändern Sie Ihr Ausführungsskript so, dass es innerhalb des konfigurierten Timeouts abgeschlossen wird.

FEHLER BEIM HERUNTERFAHREN

Dies kann auftreten, wenn beim Beenden eines Komponentenskripts ein Fehler auftritt. Der Fehlercode wird im Komponentenprotokoll gemeldet. Überprüfen Sie das Shutdown-Skript auf Fehler und stellen Sie Ihre Komponente erneut bereit.

SHUTDOWN_TIMEOUT

Dieser Fehler kann auftreten, wenn das Shutdown-Skript nicht innerhalb des konfigurierten Timeout-Zeitraums abgeschlossen wurde. Erhöhen Sie entweder den im `Shutdown` Abschnitt des Rezepts angegebenen `Timeout` Zeitraum oder ändern Sie Ihr Ausführungsskript so, dass es innerhalb des konfigurierten Timeouts abgeschlossen wird.

Markieren Ihrer AWS IoT Greengrass Version 2-Ressourcen mit Tags

Mit Tags können Sie Ihre Ressourcen in AWS IoT Greengrass organisieren und verwalten. Sie können Tags verwenden, um Ihren Ressourcen Metadaten zuzuweisen, und Sie können Tags in IAM-Richtlinien verwenden, um den bedingten Zugriff auf Ihre Ressourcen zu definieren.

> **Note**
>
> Derzeit werden Greengrass-Ressourcen-Tags nicht für die AWS IoT-Fakturierungsgruppen oder Kostenzuordnungsberichte unterstützt.

Verwendung von Tags in AWS IoT Greengrass V2

Sie können Tags verwenden, um Ihre AWS IoT Greengrass-Ressourcen nach Zweck, Besitzer, Umgebung oder einer anderen Klassifizierung für Ihren Anwendungsfall zu kategorisieren. Wenn Sie eine bestimmte Ressource desselben Typs können, können, können, können.

Jeder Tag (Markierung) besteht aus einem Schlüssel und einem optionalen Wert, beides können Sie bestimmen. Sie können zum Beispiel eine Reihe von Tags für Ihre Kerngeräte definieren, mit denen Sie den Besitzer der Geräte verfolgen können. Wir empfehlen die Erstellung von Tag-Schlüsseln, die die Anforderungen der jeweiligen Ressourcenart erfüllen. Durch die Verwendung eines konsistenten Satzes von Tag-Schlüsseln können Sie Ihre Ressourcen einfacher verwalten.

Tag mit demAWS Management Console

Der Tag Editor (Tag-Editor) in der AWS Management Console bietet eine zentrale, einheitliche Möglichkeit, für Ressourcen aus allen AWS-Services Tags zu erstellen und zu verwalten. Weitere Informationen finden Sie unter Tag-Editor im AWS Resource Groups-Benutzerhandbuch.

Tag mit derAWS IoT Greengrass V2 API

Sie können dieAWS IoT Greengrass V2 API verwenden, um mit Tags zu arbeiten. 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 AWS-Referenz.

- Wenn Sie bei der Erstellung einer Ressource Tags hinzufügen möchten, definieren Sie diese in der Eigenschaft `tags` der Ressource.
- Verwenden, um Tags zu einer bestehenden -Ressource hinzuzufügen oder um die TagResource-Operation zu aktualisieren.
- Verwenden, um Tags aus einer -Ressource zu UntagResourceentfernen.
- Um die mit einer Ressource verknüpften Tags abzurufen, verwenden Sie die ListTagsForResourceOperation oder beschreiben Sie die Ressource und überprüfen Sie ihre`tags` Eigenschaft.

In der folgenden Tabelle sind Ressourcen aufgeführt, die Sie mithilfe derAWS IoT Greengrass V2 API und der entsprechenden`Describe` und/oder`CreateGet` Operationen taggen können.

Markierbare AWS IoT Greengrass V2-Ressourcen

Ressource	Operation erstellen	Bedienung beschreiben oder ausführen
Kerngerät	Keine. Führen Sie dieAWS IoT Greengrass Core-Software auf einem Gerät aus, um ein Core-Gerät zu erstellen.	GetCoreDevice
Komponente	CreateComponentVersion	DescribeComponent, GetComponent
Bereitstellung	CreateDeployment	GetDeployment

Mit den folgenden Operationen können Sie Tags für Ressourcen anzeigen und verwalten, die die Markierung mit Tags unterstützen:

- TagResource— Fügt einer Ressource Tags hinzu oder aktualisiert den Wert eines vorhandenen Tags.
- ListTagsForResource— Listet die Tags für eine Ressource auf.
- UntagResourceEntfernt Entfernt Entfernt Entfernt, Entfernt Entfernt Entfernt, Entfernt Entfernt Entfernt,

Sie können die Tags einer Ressource jederzeit hinzufügen oder entfernen. 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 ersetzt den vorherigen Wert. Sie können einen Wert zwar auf eine leere Zeichenfolge, jedoch nicht Null festlegen.

Wenn Sie eine Ressource löschen, werden der Ressource zugeordnete Tags ebenfalls gelöscht.

Verwenden von Tags mit IAM-Richtlinien

In Ihren IAM-Richtlinien können können., können können, können können. 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.

> **Note**
>
> Wenn Sie Tags verwenden, um den Zugriff von Benutzern auf Ressourcen zuzulassen oder abzulehnen, sollten Sie Benutzern nicht die Möglichkeit geben, diese Tags diesen Ressourcen hinzuzufügen oder aus diesen Ressourcen zu entfernen. Andernfalls könnte ein Benutzer Ihre Einschränkungen umgehen und Zugang zu einer Ressource erhalten, indem er seine Tags ändert.

Sie können die folgenden Schlüssel und Werte für den Bedingungskontext in dem`Condition` Element, auch`Condition` Block genannt, einer Richtlinienanweisung verwenden.

`greengrassv2:ResourceTag/`*`tag-key`*`:` *`tag-value`*

Sie können mithilfe bestimmter Tags Aktionen für Ressourcen zulassen oder ablehnen.

`aws:RequestTag/`*`tag-key`*`:` *`tag-value`*

Erfordert, dass ein bestimmtes Tag verwendet oder nicht verwendet wird, wenn Sie eine mit Tags versehene Ressource erstellen oder ändern.

`aws:TagKeys: [`*`tag-key`*`, ...]`

Erfordert, dass beim Erstellen oder Ändern einer taggbaren Ressource ein bestimmter Satz von Tag-Schlüsseln verwendet oder nicht verwendet wird.

> **Note**
>
> Die Schlüssel und Werte für den Bedingungskontext in einer IAM-Richtlinie gelten nur für Aktionen, für die eine Ressource mit Tags als erforderlichem Parameter erforderlich ist. Sie können beispielsweise den tagbasierten bedingten Zugriff für festlegen ListCoreDevices.

Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter Steuern des Zugriffs aufAWS Ressourcen mithilfe von Ressourcen-Tags und der Referenz zur IAM-JSON-Richtlinie.

AWS IoT Greengrass Ressourcen erstellen mit AWS CloudFormation

AWS IoT Greengrass ist in AWS CloudFormation integriert, ein Service, der Ihnen hilft, Ihre AWS-Ressourcen zu modellieren und einzurichten, damit Sie weniger Zeit mit der Erstellung und Verwaltung Ihrer Ressourcen und Infrastruktur verbringen können. Sie erstellen eine Vorlage, die alleAWSRessourcen, die Sie benötigen (wie Komponentenversionen und Bereitstellungen), undAWS CloudFormationstellt diese Ressourcen für Sie bereit und konfiguriert sie.

Wenn Sie AWS CloudFormation verwenden, können Sie Ihre Vorlage wiederverwenden, um Ihre AWS IoT Greengrass-Ressourcen einheitlich und wiederholt einzurichten. Beschreiben Sie Ihre Ressourcen einmal und stellen Sie dann dieselben Ressourcen immer wieder in mehreren AWS-Konten und Regionen bereit.

AWS IoT Greengrass- und AWS CloudFormation-Vorlagen

Um Ressourcen für AWS IoT Greengrass und verwandte Dienstleistungen bereitzustellen und zu konfigurieren, müssen Sie AWS CloudFormation-Vorlagen kennen und verstehen. Vorlagen sind formatierte Textdateien in JSON oder YAML. Diese Vorlagen beschreiben die Ressourcen, die Sie in Ihren AWS CloudFormation-Stacks bereitstellen möchten. Wenn Sie noch keine Erfahrungen mit JSON oder YAML haben, können Sie AWS CloudFormation Designer verwenden, der den Einstieg in die Arbeit mit AWS CloudFormation-Vorlagen erleichtert. Weitere Informationen finden Sie unter Was ist AWS CloudFormation-Designer? im AWS CloudFormation-Benutzerhandbuch.

AWS IoT Greengrassunterstützt das Erstellen von Komponentenversionen und Bereitstellungen inAWS CloudFormation. Weitere Informationen, einschließlich Beispiele für JSON- und YAML-Vorlagen für Komponentenversionen und -Bereitstellungen, finden Sie in derAWS IoT GreengrassRessourcentyp Referenzin derAWS CloudFormationBenutzerhandbuch.

ComponentVersion Vorlagenbeispiel

Im Folgenden finden Sie die YAML-Vorlage für eine Version einer einfachen Komponente. Das JSONRezept enthält Zeilenumbrüche für eine bessere Lesbarkeit.

```
Parameters:
  ComponentVersion:
    Type: String
```

```
Resources:
  TestSimpleComponentVersion:
    Type: AWS::GreengrassV2::ComponentVersion
    Properties:
      InlineRecipe: !Sub
        - "{\n
            \"RecipeFormatVersion\": \"2020-01-25\",\n
            \"ComponentName\": \"component1\",\n
            \"ComponentVersion\": \"${ComponentVersion}\",\n
            \"ComponentType\": \"aws.greengrass.generic\",\n
            \"ComponentDescription\": \"This\",\n
            \"ComponentPublisher\": \"You\",\n
            \"Manifests\": [\n
            {\n
              \"Platform\": {\n
              \"os\": \"darwin\"\n
            },\n
            \"Lifecycle\": {},\n
            \"Artifacts\": []\n
           },\n
           {\n
             \"Lifecycle\": {},\n
             \"Artifacts\": []\n
           }\n
          ],\n
          \"Lifecycle\": {\n
          \"install\": {\n
            \"script\": \"yuminstallpython\"\n
          }\n
         }\n
        }"
        - { ComponentVersion: !Ref ComponentVersion }
```

Beispiel einer Bereitstellungsvorlage

Im Folgenden finden Sie eine YAML-Datei, die eine einfache Vorlage für eine Bereitstellung definiert.

```
Parameters:
  ComponentVersion:
    Type: String
  TargetArn:
    Type: String
Resources:
```

```
  TestDeployment:
    Type: AWS::GreengrassV2::Deployment
    Properties:
      Components:
        component1:
          ComponentVersion: !Ref ComponentVersion
      TargetArn: !Ref TargetArn
      DeploymentName: CloudFormationIntegrationTest
      DeploymentPolicies:
        FailureHandlingPolicy: DO_NOTHING
        ComponentUpdatePolicy:
          TimeoutInSeconds: 5000
          Action: SKIP_NOTIFY_COMPONENTS
        ConfigurationValidationPolicy:
          TimeoutInSeconds: 30000
Outputs:
  TestDeploymentArn:
    Value: !Sub
      - arn:${AWS::Partition}:greengrass:${AWS::Region}:${AWS::AccountId}:deployments:
${DeploymentId}
      - DeploymentId: !GetAtt TestDeployment.DeploymentId
```

Weitere Informationen zu AWS CloudFormation

Weitere Informationen zu AWS CloudFormation finden Sie in den folgenden Ressourcen.

- AWS CloudFormation
- AWS CloudFormation-Benutzerhandbuch
- AWS CloudFormation API Referenz
- AWS CloudFormation-Benutzerhandbuch für die Befehlszeilenschnittstelle

Open-SourceAWS IoT Greengrass-Core-Software

Die AWS IoT Greengrass Version 2 Edge-Laufzeit (Kern) und andere Komponenten der AWS IoT Greengrass Core-Software sind Open Source. Das bedeutet, dass Sie den Code überprüfen können, um Interaktionen mit Ihren Anwendungen zu beheben. Sie können die AWS IoT Greengrass-Core-Software auch an Ihre spezifischen Software- und Hardwareanforderungen anpassen und erweitern.

Informationen zu den Open-Source-Repositorys für die -AWS IoT GreengrassCore-Software finden Sie in der aws-greengrass-Organisation auf GitHub. Ihre Verwendung von Open-Source-Software wird durch die Open-Source-Lizenz im entsprechenden GitHub Repository geregelt.

Ihre Verwendung der AWS IoT Greengrass -Core-Software und -Komponenten, die keiner Open-Source-Lizenz unterliegen, unterliegt der AWS Greengrass Core Software License .

Dokumentenverlauf für das AWS IoT Greengrass V2 Developer Guide

In der folgenden Tabelle wird die Dokumentation für diese Version von beschrieben AWS IoT Greengrass Version 2.

- API-Version: 2020-11-30

Änderung	Beschreibung	Datum
Shadow Manager v2.3.8 veröffentlicht	Shadow Manager v2.3.8 ist verfügbar. Diese Version behebt ein Problem, bei dem der Shadow Manager während der MQTT-Client-Verbindung eine Deadlock-Situation verursacht.	5. Juni 2024
Greengrass CLI v2.12.6 veröffentlicht	Die Greengrass CLI-Komponente v2.12.6 ist verfügbar.	24. Mai 2024
AWS IoT Greengrass Softwareupdate Core v2.12.6	Diese Version stellt Version 2.12.6 der Greengrass Nucleus-Komponente und mit Updates AWS bereitgestellte Komponenten bereit.	24. Mai 2024
AWS IoT Device Tester v4.9.4 mit GGV2Q v2.5.4 veröffentlicht	Version 4.9.4 von IDT für V2 ist verfügbar. AWS IoT Greengrass Diese Version enthält die AWS IoT Greengrass V2 Qualification Suite (GGV2Q) v2.5.4 und unterstützt die Greengrass Nucleus-Versionen 2.12.0, 2.11.0, 2.10.0, 2.9.5.	3. Mai 2024

Secure Tunneling v1.0.19 veröffentlicht	Secure Tunneling v1.0.19 ist verfügbar. Diese Version aktualisiert den zugrunde liegenden AWS IoT Device Client, der von der Komponente aufgerufen wird, von Version 1.8.0 auf Version 1.9.0. Secure Tunneling v1.0.19 erhöht das Limit für gleichzeitige Tunnel auf Komponentenebene auf 20 Tunnel. Diese neue Version erhöht auch das AWS IoT Greengrass Core IPC-Timeout von 3 Sekunden auf 10 Sekunden.	1. Mai 2024
Edge-Connector für Kinesis Video Streams Streams-Komponente v1.0.5 veröffentlicht	Version 1.0.5 der Komponente Edge Connector für Kinesis Video Streams ist verfügbar. Diese Version enthält allgemeine Fehlerkorrekturen und Verbesserungen.	29. April 2024
Greengrass CLI v2.12.5 veröffentlicht	Die Greengrass CLI-Komponente v2.12.5 ist verfügbar.	25. April 2024
Die Authentifizierungskomponente für Client-Geräte v2.5.0 wurde veröffentlicht	Die Komponente zur Authentifizierung für Client-Geräte v2.5.0 ist verfügbar. Diese Version bietet Unterstützung für Richtlinienvariablen für Dingnamen. In dieser Version sind auch Richtlinienressourcen mit Platzhaltern zulässig.	25. April 2024

AWS IoT Greengrass Softwareupdate Core v2.12.5	Diese Version stellt Version 2.12.5 der Greengrass Nucleus-Komponente und von Updates AWS bereitgestellte Komponenten bereit.	25. April 2024
AWS IoT Device Tester v4.9.3 mit GGV2Q v2.5.3 veröffentlicht	Version 4.9.3 von IDT für V2 ist verfügbar. AWS IoT Greengrass Diese Version enthält die AWS IoT Greengrass V2 Qualification Suite (GGV2Q) v2.5.3 und unterstützt die Greengrass Nucleus-Versionen 2.12.0, 2.11.0, 2.10.0, 2.9.5.	5. April 2024
Greengrass CLI v2.12.4 veröffentlicht	Die Greengrass CLI-Komponente v2.12.4 ist verfügbar.	2. April 2024
AWS IoT Greengrass Softwareupdate Core v2.12.4	Diese Version stellt Version 2.12.4 der Greengrass Nucleus-Komponente und mit Updates AWS bereitgestellte Komponenten bereit.	2. April 2024
Shadow Manager v2.3.7 veröffentlicht	Shadow Manager v2.3.7 ist verfügbar. Diese Version behebt ein Problem, bei dem Shadow Manager während einer Shadow Manager-Synchronisierung regelmäßig einen `NullPointerException` Fehler protokolliert.	27. März 2024

Moquette MQTT 3.1.1 Broker v2.3.6 veröffentlicht	Die Broker-Komponente v2.3.6 für Moquette MQTT 3.1.1 ist verfügbar. Diese Version enthält allgemeine Bugfixes und Verbesserungen.	27. März 2024
Lokale Debug-Konsole v2.4.2 veröffentlicht	Die lokale Debug-Kon solenkomponente v2.4.2 ist verfügbar. Diese Version enthält allgemeine Bugfixes und Verbesserungen.	27. März 2024
Lambda Manager v2.3.3 veröffentlicht	Die Lambda-Manager-Kom ponente v2.3.3 ist verfügbar . Diese Version enthält allgemeine Bugfixes und Verbesserungen.	27. März 2024
IP-Detektor v2.1.9 veröffent licht	Die IP-Detektor-Komponente v2.1.9 ist verfügbar. In dieser Version wird der Schritt zur Erfassung der IP-Adresse so angepasst, dass nur Protokoll e auf Debug-Protokollebene gesendet werden.	27. März 2024
AWS IoT Das Flottenbereitstell ungs-Plugin v1.2.1 wurde veröffentlicht	AWS IoT Das Fleet Provision ing Plugin v1.2.1 ist verfügbar . Diese Version behebt ein Problem, bei dem das Fleet Provisioning Plugin während eines Greengrass Nucleus-Starts offline ist. Das Fleet Provisioning-Plugin versucht nun auf unbestimmte Zeit erneut MQTT Connect-A ufrufe.	27. März 2024

AWS IoT Greengrass Softwareupdate Core v2.12.3	Diese Version stellt Version 2.12.3 der Greengrass Nucleus-Komponente und mit Updates AWS bereitgestellte Komponenten bereit.	27. März 2024
Greengrass CLI v2.12.3 veröffentlicht	Die Greengrass CLI-Komponente v2.12.3 ist verfügbar.	25. März 2024
AWS IoT Device Tester v4.9.2 mit GGV2Q v2.5.2 veröffentlicht	Version 4.9.2 von IDT für V2 ist verfügbar. AWS IoT Greengrass Diese Version enthält die AWS IoT Greengrass V2 Qualification Suite (GGV2Q) v2.5.2 und unterstützt die Greengrass Nucleus-Versionen 2.12.0, 2.11.0, 2.10.0, 2.9.5.	18. März 2024
Lookout for Vision Edge Agent v1.2.0 veröffentlicht	Lookout for Vision Edge Agent v1.2.0 ist verfügbar.	11. März 2024
AWS IoT Greengrass Softwareupdate Core v2.12.2	Diese Version stellt Version 2.12.2 der Greengrass Nucleus-Komponente und mit Updates AWS bereitgestellte Komponenten bereit.	15. Februar 2024

Shadow Manager v2.3.6 veröffentlicht	Shadow Manager v2.3.6 ist verfügbar. Diese Version behebt ein Problem, bei dem Shadow-Eigenschaften, die durch AWS Cloud Updates gelöscht wurden, während das Gerät offline ist, nach Wiederherstellung der Konnektivität weiterhin im lokalen Shadow vorhanden sind.	14. Februar 2024
Lambda Launcher v2.0.13 veröffentlicht	Version 2.0.13 der Lambda-Launcher-Komponente ist verfügbar. Diese Version enthält allgemeine Fehlerkorrekturen und Verbesserungen.	14. Februar 2024
Disk Spooler v1.0.3 veröffentlicht	Die Disk Spooler-Komponente v1.0.3 ist verfügbar. Diese Version verbessert die Leistung durch die Wiederverwendung von Datenbankverbindungen.	14. Februar 2024
Lookout for Vision Edge Agent v1.1.9 veröffentlicht	Lookout for Vision Edge Agent v1.1.9 ist verfügbar.	17. Januar 2024
Greengrass-Entwicklungskit CLI v1.6.2	Version 1.6.2 des Greengrass Development Kit CLI ist verfügbar. Diese Version behebt ein Problem, bei dem Windows gradlew.bat aufgrund des relativen Pfads nicht funktioniert. Diese Version enthält auch zusätzliche Verbesserungen.	16. Januar 2024

Neue CloudTrail Datenereignisse	Sie können jetzt AWS CloudTrail Datenereignisse protokollieren, um Informationen über Ressourcenoperationen wie das Abrufen einer Komponente oder die Konfiguration einer Bereitstellung abzurufen. Nutzen Sie diese Ereignisse, um einen Einblick in den Betrieb Ihrer Greengrass-Geräte zu erhalten.	20. Dezember 2023
Lookout for Vision Edge Agent v1.1.8 veröffentlicht	Lookout for Vision Edge Agent v1.1.8 ist verfügbar.	12. Dezember 2023
Stream Manager v2.1.12 veröffentlicht	Stream Manager v2.1.12 ist jetzt verfügbar. Diese Version ändert die Reihenfolge, in der Greengrass eine Reihe von Anmeldeinformationen für AWS Serviceanrufe auswählt.	8. Dezember 2023
MQTT Bridge v2.3.1 veröffentlicht	MQTT Bridge v2.3.1 ist verfügbar. Diese Version behebt ein seltenes Problem, bei dem der lokale MQTT-Client in eine Trennschleife gerät.	8. Dezember 2023
Disk Spooler v1.0.2 veröffentlicht	Die Disk Spooler-Komponente v1.0.2 ist verfügbar. Diese Version behebt ein Problem, bei dem das Feld für das MQTT-Nachrichtenformat in bestimmten Fällen nicht dauerhaft angezeigt wird.	8. Dezember 2023

Die Authentifizierungskomponente für Client-Geräte v2.4.5 wurde veröffentlicht	Die Authentifizierungskomponente für Client-Geräte v2.4.5 ist verfügbar. Diese Version bietet Unterstützung für Platzhalter am Ende von Dingnamen in einer Auswahlregel und behebt ein Problem, bei dem Zertifikate in bestimmten Fällen nicht mit neuen Konnektivitätsinformationen aktualisiert werden.	8. Dezember 2023
AWS IoT Greengrass Core-Softwareupdate v2.12.1	Diese Version stellt Version 2.12.1 der Greengrass Nucleus-Komponente und mit Updates AWS bereitgestellte Komponenten bereit.	8. Dezember 2023
Greengrass-Entwicklungskit CLI v1.6.1	Version 1.6.1 des Greengrass Development Kit CLI ist verfügbar. Diese Version enthält Fehlerkorrekturen und Verbesserungen.	6. Dezember 2023
Validierung des Rezepts	Es wurde eine Funktion zur Rezeptvalidierung hinzugefügt, die ein Komponentenrezept bei der Erstellung einer Komponentenversion validiert.	16. November 2023
Vom Publisher unterstützte Komponenten	AWS IoT Greengrass bietet jetzt von Publisher unterstützte Komponenten an. Diese Komponenten werden von Drittanbietern entwickelt, angeboten und gewartet.	16. November 2023

Greengrass Testing Framework v1.2.0 veröffentlicht	Greengrass Testing Framework v1.2.0 ist verfügbar.	15. November 2023
Greengrass-Entwicklungskit CLI v1.6.0	Version 1.6.0 des Greengrass Development Kit CLI ist verfügbar. Diese Version fügt während der `component publish` Befehle und eine Überprüfung der Rezeptvalidierung anhand des Greengrass-Rezepts chemas `component build` hinzu. Dieses Update hilft Entwicklern dabei, umsetzbare Probleme in ihren Komponentenrezepten zu einem früheren Zeitpunkt des Komponentenerstellungsprozesses zu identifizieren. Diese Version fügt der Vorlage außerdem eine Suite für Sicherheitstests hinzu, die mit dem `test-e2e init` Befehl abgerufen werden kann. Diese Konfidenz-Testsuite umfasst acht generische Tests, die verwendet und erweitert werden können, um den grundlegenden Anforderungen an Komponententests gerecht zu werden.	15. November 2023

AWS IoT Device Tester v4.9.1 unterstützt Greengrass Nucleus Version 2.12.0	Version 4.9.1 von IDT für AWS IoT Greengrass V2 unterstützt jetzt Greengrass Nucleus Version 2.12.0.	7. November 2023
AWS IoT Greengrass Softwareupdate Core v2.12.0	Diese Version stellt Version 2.12.0 der Greengrass Nucleus-Komponente und mit Updates AWS bereitgestellte Komponenten bereit.	7. November 2023
Betreiben Sie ein Greengrass-Core-Gerät in VPC	Der Betrieb eines Greengrass-Core-Geräts in VPC ist verfügbar. Mit dieser Funktion können Sie Bereitstellungen in VPC ohne öffentlichen Internetzugang durchführen.	3. November 2023
Greengrass CLI v2.12.0 veröffentlicht	Die Greengrass CLI-Komponente v2.12.0 ist verfügbar.	30. Oktober 2023
Stream Manager v2.1.10 veröffentlicht	Stream Manager v2.1.10 ist jetzt verfügbar. Diese Version behebt ein Problem, bei dem die HTTPS-Proxykonfiguration der Zertifikatskette von Greengrass CA nicht vertraut.	26. Oktober 2023
Lambda Launcher v2.0.12 veröffentlicht	Version 2.0.12 der Lambda-Launcher-Komponente ist verfügbar. Diese Version behebt ein Problem, bei dem der Lambda-Launcher einen Fehler ausgeben konnte, wenn der vorherige Prozess nicht ordnungsgemäß gestoppt wurde.	26. Oktober 2023

Greengrass-Entwicklungskit CLI v1.5.0	Version 1.5.0 des Greengrass Development Kit CLI ist verfügbar. Diese Version aktualisiert die von der `excludes` Build-Option erkannten Muster, wenn `build_system` dies der Fall ist. `zip` Diese Version erkennt jetzt Glob-Muster, die Pfadnamen anhand ihrer Platzhalterzeichen entsprechen. Dies ermöglicht die benutzerdefinierte Angabe, aus welchen Verzeichnissen ausgeschlossen werden soll.	26. Oktober 2023
Lookout for Vision Edge Agent v1.1.7 veröffentlicht	Lookout for Vision Edge Agent v1.1.7 ist verfügbar.	24. Oktober 2023
Shadow Manager v2.3.4 veröffentlicht	Shadow Manager v2.3.4 ist verfügbar. Diese Version bietet Unterstützung für Null-Dokumente und leere Shadow-State-Dokumente.	18. Oktober 2023
Log Manager v2.3.6 veröffentlicht	Die Log Manager-Komponente v2.3.6 ist verfügbar.	18. Oktober 2023
Lokale Debug-Konsole v2.4.0 veröffentlicht	Die lokale Debug-Konsolenkomponente v2.4.0 ist verfügbar.	18. Oktober 2023
Lambda Manager v2.3.1 veröffentlicht	Die Lambda-Manager-Komponente v2.3.1 ist verfügbar.	18. Oktober 2023
Greengrass CLI v2.11.3 veröffentlicht	Die Greengrass CLI-Komponente v2.11.3 ist verfügbar.	18. Oktober 2023

AWS IoT Greengrass Core v2.11.3-Softwareupdate	Diese Version stellt Version 2.11.3 der Greengrass Nucleus-Komponente und mit Updates AWS bereitgestellte Komponenten bereit.	18. Oktober 2023
Secure Tunneling v1.0.17 veröffentlicht	Secure Tunneling v1.0.17 ist verfügbar.	04. Oktober 2023
Greengrass-Entwicklungskit CLI v1.4.0	Version 1.4.0 des Greengras s Development Kit CLI ist verfügbar. Diese Version fügt einen neuen `config` Befehl hinzu, der eine interaktive Aufforderung zum Ändern von Feldern in einer vorhandenen GDK-Konfi gurationsdatei startet. In dieser Version werden auch die `gdk component publish` Befehle `gdk component build` und geändert, um zu überprüfen, ob die Rezeptgrö ße den Greengrass-Anforde rungen entspricht (<=16000 Byte), bevor Sie fortfahren.	2. Oktober 2023
Moquette MQTT 3.1.1 Broker v2.3.5 veröffentlicht	Die Moquette MQTT 3.1.1 Broker-Komponente v2.3.5 ist verfügbar. Diese Version aktualisiert Moquette auf Version 0.17.	28. September 2023

MQTT Bridge v2.3.0 veröffentlicht	MQTT Bridge v2.3.0 ist verfügbar. Diese Version fügt MQTT 5-Unterstützung für das Bridging zwischen AWS IoT Core und lokalen MQTT-Quellen hinzu.	28. September 2023
Lookout for Vision Edge Agent v1.1.6 veröffentlicht	Lookout for Vision Edge Agent v1.1.6 ist verfügbar.	27. September 2023
Lambda Manager v2.3.0 veröffentlicht	Die Lambda-Manager-Komponente v2.3.0 ist verfügbar.	15. September 2023
Lambda Launcher v2.0.11 veröffentlicht	Version 2.0.11 der Lambda-Launcher-Komponente ist verfügbar. Diese Version unterstützt Lambda Manager 2.3.0.	15. September 2023
Moquette MQTT 3.1.1 Broker v2.3.4 veröffentlicht	Die Moquette MQTT 3.1.1 Broker-Komponente v2.3.4 ist verfügbar.	1. September 2023
Greengrass-Testframework	GTF ist eine Sammlung von Bausteinen zur Unterstützung end-to-end der Automatisierung. Es ermöglicht AWS IoT Greengrass Version 2 internen Kunden, dasselbe Test-Framework zu verwenden, das das Serviceteam zur Qualifizierung von Softwareänderungen, zur automatisierten Abnahme und zur Qualitätssicherung verwendet.	11. August 2023

AWS IoT Greengrass Core-Softwareupdate v2.11.2	Diese Version stellt Version 2.11.2 der Greengrass Nucleus-Komponente und mit Updates AWS bereitgestellte Komponenten bereit.	9. August 2023
Greengrass-Entwicklungskit CLI v1.3.0	Version 1.3.0 des Greengras s Development Kit CLI ist verfügbar. Diese Version fügt einen neuen `test-e2e` Befehl hinzu, der das end-to-end Testen von Komponenten mit Open Test Framework unterstützt.	21. Juli 2023
AWS IoT Greengrass Core v2.11.1 Softwareupdate	Diese Version stellt Version 2.11.1 der Greengrass Nucleus-Komponente und mit Updates AWS bereitgestellte Komponenten bereit.	21. Juli 2023
Disk Spooler v1.0.0 veröffent licht	Die Disk Spooler-Komponente v1.0.0 ist verfügbar.	28. Juni 2023
AWS IoT Greengrass Softwareupdate Core v2.11.0	Diese Version stellt Version 2.11.0 der Greengrass Nucleus-Komponente und mit Updates AWS bereitgestellte Komponenten bereit.	28. Juni 2023
AWS IoT Greengrass Softwareupdate Core v2.10.3	Diese Version stellt Version 2.10.3 der Greengrass Nucleus-Komponente und mit Updates AWS bereitgestellte Komponenten bereit.	21. Juni 2023

AWS IoT Greengrass Softwareupdate Core v2.10.2	Diese Version stellt Version 2.10.2 der Greengrass Nucleus-Komponente und von Updates AWS bereitgestellte Komponenten bereit.	5. Juni 2023
AWS IoT Greengrass Softwareupdate Core v2.10.1	Diese Version enthält Version 2.10.1 der Greengrass Nucleus-Komponente und von Updates AWS bereitgestellte Komponenten.	11. Mai 2023
AWS IoT Greengrass Core-Softwareupdate v2.10.0	Diese Version stellt Version 2.10.0 der Greengrass Nucleus-Komponente und von Updates AWS bereitgestellte Komponenten bereit.	09. Mai 2023
SageMaker Edge Manager wurde eingestellt	Die Amazon SageMaker Edge Manager-Komponente wird am 26. April 2024 eingestellt.	28. April 2023
AWS IoT Greengrass Softwareupdate Core v2.9.6	Diese Version enthält Version 2.9.6 der Greengrass Nucleus-Komponente und von Updates AWS bereitgestellte Komponenten.	20. April 2023
Log Manager v2.3.2 veröffentlicht	Die Log Manager-Komponente v2.3.2 ist verfügbar.	19. April 2023

Stream Manager v2.1.4 veröffentlicht	Stream Manager v2.1.4 ist jetzt verfügbar. Diese Version behebt ein Problem, bei dem Einträge für dasselbe Eigenschafts-Asset mit demselben Zeitstempel innerhalb eines einzigen Batches `ConflictingOperationException` von der SiteWise API zurückgegeben werden, was dazu führt, dass der Stream-Manager es kontinuierlich wiederholt. In dieser Version wurde auch das standardmäßige Verbindungstimeout von 3 Sekunden auf 1 Minute aktualisiert.	13. April 2023
Greengrass-Entwicklungskit CLI v1.2.3	Version 1.2.3 des Greengrass Development Kit CLI ist verfügbar. Diese Version enthält Fehlerkorrekturen.	13. April 2023
Die Authentifizierungskomponente für Client-Geräte v2.4.0 wurde veröffentlicht	Die Authentifizierungskomponente für Client-Geräte v2.4.0 ist verfügbar. Diese Version bietet Unterstützung für die Client-Geräteauthentifizierung zur Ausgabe von Betriebsmetriken, die auf dem Greengrass Client Device Dashboard angezeigt werden können.	10. April 2023

Greengrass-Entwicklungskit CLI v1.2.2	Version 1.2.2 des Greengrass Development Kit CLI ist verfügbar. Diese Version enthält Verbesserungen und Fehlerkorrekturen.	7. April 2023
AWS IoT Greengrass Core v2.9.5 Softwareupdate	Diese Version enthält Version 2.9.5 der Greengrass Nucleus-Komponente und von Updates AWS bereitgestellte Komponenten.	30. März 2023
Stream Manager v2.1.3 veröffentlicht	Stream Manager v2.1.3 ist jetzt verfügbar. Diese Version behebt ein Startproblem unter Windows OS, wenn es als SYSTEM-Benutzer ausgeführt wird.	7. März 2023
Modbus-RTU-Protokolladapter v2.1.5 veröffentlicht	Die Modbus-RTU-Protokolladapter-Komponente v2.1.5 ist verfügbar. Diese Version behebt ein Problem mit dem Vorgang. `ReadDiscreteInput`	7. März 2023
Die Komponente für die Authentifizierung auf Client-Geräten, Version 2.3.2 wurde veröffentlicht	Die Authentifizierungskomponente für Client-Geräte v2.3.2 ist verfügbar. Diese Version bietet Unterstützung für das Zwischenspeichern von Hostnameninformationen, sodass die Komponente die Zertifikatssubjekte korrekt generiert, wenn sie offline neu gestartet wird.	7. März 2023

AWS IoT Device Tester v4.7.0 unterstützt Greengrass Nucleus Version 2.9.4	Version 4.7.0 von IDT für AWS IoT Greengrass V2 unterstützt jetzt Greengrass Nucleus Version 2.9.4.	2. März 2023
Greengrass Command Line Interface v1.2.0 veröffentlicht	Greengrass Command Line Interface v1.2.0 ist verfügbar.	28. Februar 2023
AWS IoT Greengrass Softwareupdate Core v2.9.4	Diese Version enthält Version 2.9.4 der Greengrass Nucleus-Komponente und von Updates AWS bereitgestellte Komponenten.	24. Februar 2023
Shadow Manager v2.3.1 veröffentlicht	Shadow Manager v2.3.1 ist verfügbar. Diese Version behebt ein Problem, das die Synchronisierung von Cloud-Shadow-Updates verhindern kann. Diese Version behebt auch ein Problem, bei dem Änderungen an der Named Shadow-Sync-Konfiguration nur für einen benannten Shadow gelten.	21. Februar 2023
AWS IoT Device Tester v4.7.0 unterstützt Greengrass Nucleus Version 2.9.3	Version 4.7.0 von IDT für AWS IoT Greengrass V2 unterstützt jetzt Greengrass Nucleus Version 2.9.3.	9. Februar 2023
Bewährte IAM-Praktiken wurden aktualisiert	Aktualisierter Leitfaden, angepasst an die bewährten IAM-Methoden. Weitere Informationen finden Sie unter Bewährte IAM-Methoden.	3. Februar 2023

AWS IoT Greengrass Core-Softwareupdate v2.9.3	Diese Version enthält Version 2.9.3 der Greengrass Nucleus-Komponente und von Updates AWS bereitgestellte Komponenten.	1. Februar 2023
Log Manager v2.3.1 veröffentlicht	Log Manager v2.3.1 ist verfügbar.	27. Januar 2023
AWS IoT Device Tester v4.7.0 unterstützt Greengrass Nucleus Version 2.9.2	Version 4.7.0 von IDT für AWS IoT Greengrass V2 unterstützt jetzt Greengrass Nucleus Version 2.9.2.	3. Januar 2023
Shadow Manager v2.3.0 veröffentlicht	Shadow Manager v2.3.0 ist verfügbar. Diese Version behebt ein Problem, das dazu führen kann, dass Schatten nicht synchronisiert werden, wenn ein Gerät den privaten Schlüssel des Greengrass-Geräts in einem Hardware-Sicherheitsmodul speichert.	29. Dezember 2022
AWS IoT Das Fleet Provisioning Plugin v1.2.0 wurde veröffentlicht	AWS IoT Das Fleet Provisioning Plugin v1.2.0 ist verfügbar. Diese Version bietet Unterstützung für die Gerätebereitstellung per Zertifikatsignieranforderung mit konfigurierbarem privaten Schlüsselpfad.	22. Dezember 2022

AWS IoT Greengrass Core v2.9.2 Softwareupdate	Diese Version enthält Version 2.9.2 der Greengrass Nucleus-Komponente und von Updates AWS bereitgestellte Komponenten.	22. Dezember 2022
AWS IoT Device Tester v4.7.0 mit GGV2Q v2.5.0 veröffentlicht	Version 4.7.0 von IDT für V2 ist verfügbar. AWS IoT Greengrass Diese Version enthält die AWS IoT Greengrass V2 Qualification Suite (GGV2Q) v2.5.0 und unterstützt die Greengrass Nucleus-Versionen 2.9.1, 2.9.0, 2.8.1, 2.8.0, 2.7.0 und 2.6.0.	13. Dezember 2022
Shadow Manager v2.2.4 veröffentlicht	Behebt ein Problem, bei dem die Überprüfung der Schattengröße nicht mit der der Cloud übereinstimmte, wenn das lokale Shadow-Dokument aktualisiert wurde. Dies behebt auch ein Problem, bei dem der Shadow-Manager keine Konfigurationsupdates mehr abhört, wenn eine Bereitstellung `RESET` auf den Konfigurationsknoten ausgeführt wird.	8 Dezember 2022
Lookout for Vision Edge Agent 1.1.1 veröffentlicht	Die Komponente Lookout for Vision Edge Agent v1.1.1 ist verfügbar.	5. Dezember 2022
Log Manager v2.3.0 veröffentlicht	Die Log Manager-Komponente v2.3.0 ist verfügbar.	18. November 2022

AWS IoT Device Tester v4.5.11 unterstützt Greengrass Nucleus Version 2.9.1	Version 4.5.11 von IDT für AWS IoT Greengrass V2 unterstützt jetzt Greengrass Nucleus Version 2.9.1.	18. November 2022
AWS IoT Greengrass Softwareupdate Core v2.9.1	Diese Version enthält Version 2.9.1 der Greengrass Nucleus-Komponente und von Updates AWS bereitgestellte Komponenten.	18. November 2022
AWS IoT Device Tester v4.5.11 unterstützt Greengrass Nucleus Version 2.9.0	Version 4.5.11 von IDT für AWS IoT Greengrass V2 unterstützt jetzt Greengrass Nucleus Version 2.9.0.	17. November 2022
Stream Manager v2.1.2 veröffentlicht	Stream Manager v2.1.2 ist jetzt verfügbar. Diese Version behebt ein Problem unter Windows-Betriebssystemen, die eine andere Sprache als Englisch verwenden.	15. November 2022
AWS IoT Greengrass Core v2.9.0 Softwareupdate	Diese Version stellt Version 2.9.0 der Greengrass Nucleus-Komponente und mit Updates AWS bereitgestellte Komponenten bereit.	15. November 2022
AWS IoT Device Tester v4.5.11 unterstützt Greengrass Nucleus Version 2.8.1	Version 4.5.11 von IDT für AWS IoT Greengrass V2 unterstützt jetzt Greengrass Nucleus Version 2.8.1.	19. Oktober 2022

AWS IoT Device Tester v4.5.11 mit GGV2Q v2.4.1 veröffentlicht	Version 4.5.11 von IDT für V2 ist verfügbar. AWS IoT Greengrass Diese Version enthält die AWS IoT Greengrass V2 Qualification Suite (GGV2Q) v2.4.1 und unterstützt die Greengrass Nucleus-Versionen 2.8.0, 2.7.0 und 2.6.0.	13. Oktober 2022
AWS IoT Greengrass Softwareupdate Core v2.8.1	Diese Version enthält Version 2.8.1 der Greengrass Nucleus-Komponente und von Updates AWS bereitgestellte Komponenten.	13. Oktober 2022
AWS IoT Greengrass Core-Softwareupdate v2.8.0	Diese Version enthält Version 2.8.0 der Greengrass Nucleus-Komponente und von Updates AWS bereitgestellte Komponenten.	7. Oktober 2022
Unterstützung für Bereitstellungen hinzugefügt AWS CloudFormation	AWS CloudFormation unterstützt jetzt AWS IoT Greengrass Bereitstellungen als Ressource.	6. Oktober 2022

SageMaker Edge Manager v1.3.0 veröffentlicht	Die Amazon SageMaker Edge Manager-Komponente v1.3.0 ist verfügbar. Diese Version bietet Unterstützung für diese Komponente zur Festlegung der Festplattengröße für den TensorRT-Modellcache und verbessert die Parallelität von Vorhersagen, um Gerätebeschleuniger-Engines wie GPUs besser nutzen zu können.	01. September 2022
Verwenden Sie den IPC-Client (Interprocess Communication) V2	Es wurden Informationen zum IPC-Client V2 hinzugefügt, wodurch die Menge an Code reduziert wird, die Sie für die Verwendung von IPC-Vorgängen schreiben müssen, und um häufige Fehler zu vermeiden, die beim IPC-Client V1 auftreten können.	12. August 2022
AWS IoT Device Tester v4.5.8 mit GGV2Q v2.4.0 veröffentlicht	Version 4.5.8 von IDT für V2 ist verfügbar. AWS IoT Greengrass Diese Version enthält die AWS IoT Greengrass V2 Qualification Suite (GGV2Q) v2.4.0 und unterstützt die Greengrass Nucleus-Versionen 2.7.0, 2.6.0 und 2.5.6.	12. August 2022

SageMaker Edge Manager v1.2.0 wurde veröffentlicht	Die Amazon SageMaker Edge Manager-Komponente v1.2.0 ist verfügbar. Diese Version bietet Unterstützung für diese Komponente zum automatischen Abrufen von SageMaker NEO-kompilierten Modellen, die Sie auf Amazon S3 hochladen, sodass Sie neue Modelle bereitstellen können, ohne eine AWS IoT Greengrass Bereitstellung erstellen zu müssen.	03. August 2022
AWS IoT Device Tester v4.5.3 unterstützt Greengrass Nucleus Version 2.7.0	Version 4.5.3 von IDT für AWS IoT Greengrass V2 unterstützt jetzt Greengrass Nucleus Version 2.7.0.	1. August 2022
Stream Manager v2.1.0 veröffentlicht	Stream Manager v2.1.0 ist jetzt verfügbar. Diese Version unterstützt Sie beim Senden von Telemetriedaten an Amazon EventBridge.	28. Juli 2022
AWS IoT Greengrass Softwareupdate Core v2.7.0	Diese Version enthält Version 2.7.0 der Greengrass Nucleus-Komponente und von Updates AWS bereitgestellte Komponenten. Es beinhaltet Unterstützung für Sie beim Senden von Telemetriedaten an Amazon EventBridge.	28. Juli 2022

SiteWise IoT-Publisher v2.2.0 veröffentlicht	Die SiteWise IoT-Publisher-Komponente v2.2.0 ist verfügbar. In dieser Version wird die Komponente so aktualisiert, dass sie Daten komprimiert, bevor sie an den AWS IoT SiteWise Service gesendet werden, wodurch die Bandbreitennutzung um bis zu 75 Prozent reduziert wird.	19. Juli 2022
Tutorial: Entwickeln Sie eine Komponente, die mit Schatten auf Client-Geräten interagiert	Dem Tutorial: Interagieren Sie mit lokalen IoT-Geräten über MQTT wurde ein neues Modul hinzugefügt, in dem Sie lernen können, wie Sie eine Komponente entwickeln, die mit Client-Geräteschatten interagiert.	18. Juli 2022
Wählen Sie einen lokalen MQTT-Broker	Es wurden Informationen zur Auswahl eines lokalen MQTT-Brokers hinzugefügt, bei dem Client-Geräte eine Verbindung zu einem Kerngerät herstellen.	18. Juli 2022
AWS IoT Device Tester v4.5.3 unterstützt Greengrass Nucleus Version 2.6.0	Version 4.5.3 von IDT für AWS IoT Greengrass V2 unterstützt jetzt Greengrass Nucleus Version 2.6.0.	29. Juni 2022

AWS IoT Greengrass Softwareupdate Core v2.6.0	Diese Version stellt Version 2.6.0 der Greengrass Nucleus-Komponente und von Updates AWS bereitgestellte Komponenten bereit. Es beinhaltet Unterstützung für Client-Geräteschatten und einen lokalen MQTT 5-Broker für Client-Geräte. Es umfasst auch Unterstützung für Platzhalter in lokalen Themen zum Veröffentlichen und Abonnieren, Rezeptvariablen in Komponentenkonfigurationen und Platzhalter in IPC-Autorisierungsrichtlinien. Diese Funktionen ermöglichen es Ihnen, Komponenten, die Sie auf Flotten von Kerngeräten bereitstellen, einfacher zu entwickeln und zu konfigurieren. Diese Version bietet auch Unterstützung für Komponenten zur Verwendung von IPC-Vorgängen, die lokale Bereitstellungen und Komponenten auf einem Kerngerät verwalten.	27. Juni 2022

Updates für Komponenten auf Client-Geräten	Client Device Auth v2.1.0, MQTT Broker (Moquette) v2.1.0, MQTT Bridge v2.1.1 und IP Detector v2.1.2 sind verfügbar. Diese Version verbessert die Zertifikatsrotation, verbessert die Leistung des MQTT-Brokers und behebt Probleme mit der Art und Weise, wie diese Komponenten Aktualisierungen zum Zurücksetzen der Konfiguration handhaben.	14. Juni 2022
AWS IoT Device Tester v4.5.3 unterstützt Greengrass Nucleus Version 2.5.6	Version 4.5.3 von IDT für AWS IoT Greengrass V2 unterstützt jetzt Greengrass Nucleus Version 2.5.6.	1. Juni 2022
AWS IoT Greengrass Softwareupdate Core v2.5.6	Diese Version enthält Version 2.5.6 der Greengrass Nucleus-Komponente und von Updates AWS bereitgestellte Komponenten. Es beinhaltet Unterstützung für Hardware-Sicherheitsmodule mit ECC-Schlüsseln. Es enthält auch andere Fehlerkorrekturen und Verbesserungen.	31. Mai 2022
AWS IoT Das Fleet Provisioning Plugin v1.1.0 wurde veröffentlicht	AWS IoT Das Fleet Provisioning Plugin v1.1.0 ist verfügbar. Diese Version bietet Unterstützung für zusätzliche Dateipfadformate, wenn Sie das Plugin auf Windows-Geräten konfigurieren.	12. Mai 2022

Neue Lambda-Laufzeiten veröffentlicht	Unterstützung für neue Lambda-Laufzeiten hinzugefügt: Python 3.9, Java 11 und NodeJS 14.	10. Mai 2022
Entwickeln Sie eine Greengrass-Komponente, die Komponenten-Updates verzögert	Es wurde ein Tutorial hinzugefügt, dem Sie folgen können, um zu erfahren, wie Sie eine Greengrass-Komponente entwickeln, die Komponenten-Updates von Bereitstellungen zurückhält. Möglicherweise möchten Sie ein Update verzögern, wenn der Akkustand eines Geräts niedrig ist oder wenn ein Vorgang ausgeführt wird, der nicht unterbrochen werden kann.	4. Mai 2022
CloudWatch Die Metriken v3.1.0 und v3.1.0 wurden veröffentlicht AWS IoT Device Defender	CloudWatch Die Metrik-Komponente v3.1.0 und die Komponente v3.1.0 sind verfügbar. AWS IoT Device Defender Diese Versionen bieten Unterstützung für HTTPS-Netzwerk-Proxykonfigurationen. Weitere Informationen finden Sie unter Connect über Port 443 oder über einen Netzwerk-Proxy und Ermöglichen Sie dem Kerngerät, einem HTTPS-Proxy zu vertrauen.	27. April 2022

Migrieren Sie von AWS IoT Greengrass Version 1	Es wurde eine Anleitung hinzugefügt, der Sie folgen können, um von AWS IoT Greengrass V1 zu migrieren AWS IoT Greengrass V2.	26. April 2022
AWS IoT Device Tester v4.5.3 mit aktualisiertem GGV2Q v2.3.1 und IDT v4.5.1 mit GGV2Q v2.3.0 zu den unterstützten Versionen hinzugefügt	Version 4.5.3 von IDT für AWS IoT Greengrass V2 mit AWS IoT Greengrass V2 Qualification Suite (GGV2Q) v2.3.1 wurde aktualisiert und unterstützt nun die Greengrass Nucleus-Versionen 2.5.5, 2.5.4 und 2.5.3. Dieses Update enthält auch IDT 4.5.1 mit V2 Qualification Suite (GGV2Q) v2.3.0 als unterstützte Version. AWS IoT Greengrass IDT 4.5.1 mit AWS IoT Greengras s V2 Qualification Suite (GGV2Q) v2.3.0 unterstützt Greengrass Nucleus Version 2.5.3.	25. April 2022
Modbus-RTU-Protokolladapter v2.1.0 veröffentlicht	Die Modbus-RTU-Protoko lladapter-Komponente v2.1.0 ist verfügbar. Diese Version fügt neue Parameter hinzu, die Sie angeben können, um die serielle Kommunikation mit Modbus RTU-Geräten zu konfigurieren.	20. April 2022

CloudWatch Metrics v2.1.0, Firehose v2.1.0 und Amazon SNS v2.1.0 veröffentlicht	CloudWatch Die Metrikkomponente v2.1.0, Firehose Firehose-Komponente v2.1.0 und die Amazon SNS SNS-Komponente v2.1.0 sind verfügbar. Diese Versionen bieten Unterstützung für HTTPS-Netzwerk-Proxykonfigurationen. Weitere Informationen finden Sie unter Connect über Port 443 oder über einen Netzwerk-Proxy und Ermöglichen Sie dem Kerngerät, einem HTTPS-Proxy zu vertrauen.	19. April 2022
AWS IoT Device Tester v4.5.3 mit GGV2Q v2.3.1 veröffentlicht	Version 4.5.3 von IDT für V2 ist verfügbar. AWS IoT Greengrass Diese Version enthält die AWS IoT Greengrass V2 Qualification Suite (GGV2Q) v2.3.1 und unterstützt Greengrass Nucleus Version 2.5.5.	15. April 2022

AWS IoT Greengrass Softwareupdate Core v2.5.5	Diese Version enthält Version 2.5.5 der Greengras s Nucleus-Komponente und von Updates AWS bereitges tellte Komponenten. Es bietet Unterstützung für Windows-Geräte, die eine andere Anzeigesprache als Englisch verwenden. Es behebt auch ein Problem, bei dem das Kerngerät seinen Status nach der Bereitstellung in bestimmte n Szenarien nicht an den AWS IoT Greengrass Cloud-Dienst gemeldet hat.	6. April 2022
AWS IoT Greengrass Core v2.5.4 Softwareupdate	Diese Version enthält Version 2.5.4 der Greengras s Nucleus-Komponente und von Updates AWS bereitges tellte Komponenten. Sie enthält Fehlerkorrekturen und Verbesserungen.	23. März 2022
AWS IoT Device Tester Programmgesteuert herunterl aden	Es wurden Informationen zum programmgesteuerte n Herunterladen von IDT hinzugefügt. AWS IoT Greengrass V2	15. März 2022

Greengrass-Entwicklungskit CLI v1.1.0	Version 1.1.0 des Greengrass Development Kit CLI ist verfügbar. Diese Version fügt den Befehlen `component init` und `component publish` neue Argumente hinzu. In dieser Version wird auch der `component publish` Befehl aktualisiert, um die Komponente zu erstellen, falls sie nicht erstellt wurde.	24. Februar 2022
Shadow Manager v2.1.0 veröffentlicht	Die Shadow Manager-Komponente v2.1.0 ist verfügbar. Diese Version bietet die Option, das Intervall zu konfigurieren, mit dem die Komponente Schatten synchronisiert. AWS IoT Core Sie können beispielsweise ein längeres Intervall angeben, um die Bandbreitennutzung und die Gebühren zu reduzieren.	3. Februar 2022
Dockerfile und Docker-Images für AWS IoT Greengrass die Core-Software v2.5.3	Das Dockerfile und das Docker-Image für die Core-Software v2.5.3 sind jetzt verfügbar. AWS IoT Greengrass	12. Januar 2022

AWS IoT Device Tester v4.5.1 mit GGV2Q v2.3.0 veröffentlicht	Version 4.5.1 von IDT für V2 ist verfügbar. AWS IoT Greengrass Diese Version enthält die AWS IoT Greengrass V2 Qualification Suite (GGV2Q) v2.3.0 und unterstützt die Validierung und Qualifizierung von Linux-basierten Geräten, die ein Hardware-Sicherheitsmodul (HSM) zum Speichern des von der Core-Software verwendeten privaten Schlüssels und Zertifikats verwenden. AWS IoT Greengrass	11. Januar 2022
AWS IoT Greengrass Softwareupdate Core v2.5.3	Diese Version enthält Version 2.5.3 der Greengrass Nucleus-Komponente und von Updates AWS bereitgestellte Komponenten. Es unterstützt Sie bei der Konfiguration der AWS IoT Greengrass Core-Software für die Verwendung eines privaten Schlüssels und Zertifikats, die Sie sicher in einem Hardware-Sicherheitsmodul (HSM) speichern.	6. Januar 2022
Dockerfile und Docker-Images für die Core-Software v2.5.2 AWS IoT Greengrass	Das Dockerfile und das Docker-Image für die Core-Software v2.5.2 sind jetzt verfügbar. AWS IoT Greengrass	20. Dezember 2021

AWS IoT Device Tester v4.4.1 mit GGV2Q v2.2.1 veröffentlicht	Version 4.4.1 von IDT für V2 ist verfügbar. AWS IoT Greengrass Diese Version beinhaltet die AWS IoT Greengrass V2 Qualification Suite (GGV2Q) v2.2.1 und unterstützt Greengrass Nucleus Version 2.5.2 für die Gerätequalifizierung.	12. Dezember 2021
Führen Sie mit Amazon Lookout for Vision Inferenz für maschinelles Lernen durch	Es wurden Informationen zur Durchführung von Inferenzen für maschinelles Lernen mit Lookout for Vision auf Greengrass-Kerngeräten hinzugefügt. Lookout for Vision verwendet Computer Vision, um Sehfehler in Industrieprodukten zu finden.	8. Dezember 2021
AWS IoT Device Tester v4.4.1 mit GGV2Q v2.2.0 veröffentlicht	Version 4.4.1 von IDT für V2 ist verfügbar. AWS IoT Greengrass Diese Version beinhaltet die AWS IoT Greengrass V2 Qualification Suite (GGV2Q) v2.2.0 und unterstützt Greengrass Nucleus Version 2.5.2 für die Gerätequalifizierung.	6. Dezember 2021

AWS IoT Greengrass Softwareupdate Core v2.5.2	Diese Version enthält Version 2.5.2 der Greengrass Nucleus-Komponente und von Updates AWS bereitgestellte Komponenten. Es behebt ein Problem mit dem Windows-Dienst, das nach den Greengrass Nucleus-Updates auftritt. Es beinhaltet auch Unterstützung für die AWS IoT Device Defender Komponente auf Windows-Geräten.	3. Dezember 2021
Neuer Edge-Connector für die Kinesis Video Streams Streams-Komponente	Version 1.0.0 der Komponente Edge Connector für Kinesis Video Streams ist verfügbar. Dieser AWS-provided liest Video-Feeds von lokalen Kameras und veröffentlicht die Streams in Kinesis Video Streams. Diese Komponente lässt sich in integrieren AWS IoT TwinMaker, sodass Sie Videostreams und andere Daten in Grafana-Dashboards anzeigen und verwalten können.	30. November 2021

Verwalten Sie Greengrass-Kerngeräte mit AWS Systems Manager	Es wurden Informationen zur Verwaltung von Greengras s-Core-Geräten mit AWS Systems Manager hinzugefü gt. Systems Manager ist ein AWS Service, mit dem Sie Betriebsdaten anzeigen, Betriebsaufgaben automatis ieren und Sicherheit und Compliance gewährleisten können.	29. November 2021
Greengrass-Entwicklungskit CLI	Es wurden Informationen zum AWS IoT Greengrass Development Kit Command-Line Interface (GDK CLI) hinzugefügt. Dabei handelt es sich um ein Tool, das Sie auf Ihren lokalen Entwicklu ngscomputer herunterladen können, um Sie bei der Entwicklung benutzerdefinierte r Greengrass-Komponenten zu unterstützen. Sie können die GDK-CLI verwenden, um benutzerdefinierte Komponent en zu erstellen, zu erstellen und zu veröffentlichen.	29. November 2021

Von der Community bereitgestellte Greengrass-Komponenten	Es wurden Informationen zum Greengrass-Softwarekatalog hinzugefügt, bei dem es sich um einen Index von Greengrass-Komponenten handelt, die von der Greengrass-Community entwickelt wurden. Aus diesem Katalog können Sie Komponenten herunterladen, ändern und bereitstellen, um Ihre Greengrass-Anwendungen zu erstellen.	29. November 2021
AWS IoT Greengrass Core v2.5.1 Softwareupdate	Diese Version enthält Version 2.5.1 der Greengrass Nucleus-Komponente und von Updates AWS bereitgestellte Komponenten. Es beinhaltet Unterstützung für 32-Bit-Java auf Windows-Geräten. Außerdem werden Probleme mit dem neuen Verhalten beim Entfernen von Dinggruppen und beim Laden von Systemumgebungsvariablen auf Windows-Geräten behoben.	23. November 2021

AWS IoT Device Tester v4.4.0 mit GGV2Q v2.1.0 veröffentlicht	Version 4.4.0 von IDT für V2 ist verfügbar. AWS IoT Greengrass Diese Version enthält die AWS IoT Greengrass V2 Qualification Suite (GGV2Q) v2.1.0 und unterstützt die Qualifizierung von Windows-basierten Greengrass-Geräten, auf denen Greengrass Nucleus Version 2.5.0 ausgeführt wird.	19. November 2021
AWS IoT Greengrass Softwareupdate Core v2.5.0	Diese Version enthält Version 2.5.0 der Greengrass Nucleus-Komponente und von Updates AWS bereitgestellte Komponenten. Es beinhaltet Unterstützung für die Ausführung der AWS IoT Greengrass Core-Software auf Windows-Geräten. Es ändert auch das Verhalten beim Entfernen von Dinggruppen und fügt Unterstützung für HTTPS-Proxys hinzu.	12. November 2021

SageMaker Edge Manager v1.1.0 veröffentlicht	Die Amazon SageMaker Edge Manager-Komponente v1.1.0 ist verfügbar. Diese Version bietet Unterstützung für Greengrass-Core-Geräte, auf denen Amazon Linux 2 ausgeführt wird, und fügt einen neuen Konfigurationsparameter hinzu, um den Speicherort des Capture-Datenordners auf Ihrem Gerät anzugeben.	3. November 2021
Update zur Vermeidung des Problems des verwirrten Stellvertreters (dienstübergreifend)	AWS IoT Greengrass V2 unterstützt die Verwendung der Kontextschlüssel `aws:SourceArn` und der `aws:SourceAccount` globalen Bedingungsschlüssel in IAM-Ressourcenrichtlinien, um das Problem mit dem verwirrten Stellvertreter zu verhindern.	1. November 2021
Updates für Komponenten auf Client-Geräten	Client Device Auth v2.0.3, IP Detector v2.1.0, MQTT Bridge v2.1.0 und MQTT Broker (Moquette) v2.0.2 sind verfügbar. Diese Version bietet volle Unterstützung für nicht standardmäßige MQTT-Broker-Ports und enthält weitere Bugfixes und Verbesserungen.	28. Oktober 2021

Shadow Manager v2.0.4 veröffentlicht	Die Shadow Manager-Komponente v2.0.4 ist verfügbar. Diese Version behebt ein Problem, das dazu führte, dass Shadow Manager neu erstellte Versionen aller zuvor gelöschten Shadows löschte. Ab dieser Version inkrementiert der `DeleteThingShadow` IPC-Vorgang die Shadow-Version.	20. Oktober 2021
Log Manager v2.2.0 veröffentlicht	Die Log Manager-Komponente v2.2.0 ist verfügbar. Der Log Manager unterstützt jetzt die Verwendung einer Konfigurationsübersicht zur Bereitstellung von Komponenten-Protokollkonfigurationen.	20. Oktober 2021
Lambda Manager v2.1.4 veröffentlicht	Die Lambda-Manager-Komponente v2.1.4 ist verfügbar. Diese Version behebt ein Problem, das dazu führte, dass Lambda-Funktionen, die NodeJS-Laufzeiten verwenden, nur eine Nachricht verarbeiteten.	20. Oktober 2021
Verwenden Sie die Kommunikation zwischen Prozessen, AWS Anmeldeinformationen und den Stream-Manager in Docker-Container-Komponenten	Es wurden Informationen zur Verwendung von Interprozesskommunikation (IPC), AWS Anmeldeinformationen und Stream-Manager in Ihren benutzerdefinierten Docker-Container-Komponenten hinzugefügt.	19. Oktober 2021

Neue Nucleus-Telemetrie-Emitter-Komponente	Version 1.0.0 der Nucleus-Telemetrie-Emitter-Komponente ist verfügbar. Diese AWS zur Verfügung gestellte Komponente sammelt Telemetriedaten zur Systemintegrität und veröffentlicht sie kontinuierlich zu einem lokalen Thema und einem MQTT-Thema. AWS IoT Core	30. September 2021
Geräteverkehr über einen Proxy oder eine Firewall zulassen	Es wurden Informationen zu den Endpunkten und Ports hinzugefügt, die das Greengrass-Core-Gerät verwendet, sodass Sie den Datenverkehr als Sicherheitsmaßnahme einschränken können.	16. September 2021
AWS IoT Device Tester v4.2.0 mit GGV2Q v2.0.1 veröffentlicht	Version 4.2.0 von IDT für AWS IoT Greengrass V2 wurde mit der V2 Qualification Suite (GGV2Q) v2.0.1 aktualisiert. AWS IoT Greengrass Diese Version unterstützt Greengrass Nucleus Version 2.4.0 für die Gerätequalifizierung.	31. August 2021

Die Komponenten des Installationsprogramms für maschinelles Lernen wurden aktualisiert	Die DLR-Installationskomponente v1.6.5 und die TensorFlow Lite-Installationskomponente v2.5.4 sind verfügbar. Diese Komponentenversionen enthalten den neuen `UseInstaller` Konfigurationsparameter, mit dem Sie das Standardinstallationsskript deaktivieren können.	30. August 2021
Integrierte Linux-Unterstützung für AWS IoT Greengrass	Das BitBake Rezept für AWS IoT Greengrass V2 ist im `meta-aws` Projekt unter verfügbar GitHub. Sie können dieses Rezept verwenden, um mithilfe des Yocto-Projekts ein benutzerdefiniertes Linux-basiertes Betriebssystem zu erstellen.	20. August 2021
Integrität des Codes	Es wurden Informationen darüber hinzugefügt, wie die Integrität von Software AWS IoT Greengrass V2 überprüft wird, die Greengrass-Core-Geräte von der herunterladen. AWS Cloud	19. August 2021

VPC-Endpunkte ()AWS PrivateLink	AWS IoT Greengrass unterstützt jetzt Schnittstellen-VPC-Endpunkte (AWS PrivateLink) für die AWS IoT Greengrass Steuerungsebene. Sie können eine private Verbindung zwischen Ihrer VPC und der AWS IoT Greengrass Kontrollebene herstellen.	16. August 2021
Stream Manager v2.0.12 veröffentlicht	Stream Manager v2.0.12 ist jetzt verfügbar. Diese Version behebt ein Problem, das Upgrades von Version 2.0.7 der Stream Manager-Komponente auf eine Version zwischen v2.0.8 und v2.0.11 verhinderte.	10. August 2021
Dockerfile und Docker-Images für die Core-Software v2.4.0 AWS IoT Greengrass	Das Dockerfile und das Docker-Image für die Core-Software v2.4.0 sind jetzt verfügbar. AWS IoT Greengrass	9. August 2021
AWS IoT Greengrass Core v2.4.0-Softwareupdate	Diese Version enthält Version 2.4.0 der Greengrass Nucleus-Komponente und von Updates AWS bereitgestellte Komponenten. Es umfasst Unterstützung für Ressourcenbeschränkungen für Komponentensysteme, IPC-Operationen zum Anhalten und Wiederaufnehmen von Komponenten und Bereitstellungs-Plug-ins.	3. August 2021

Neue Komponenten AWS IoT SiteWise	Die folgenden AWS bereitgestellten Komponenten wurden hinzugefügt für AWS IoT SiteWise: SiteWise IoT-OPC-UA-Collector, SiteWiseIoT-Publisher und IoT-Prozessor. SiteWise	29. Juli 2021
AWS IoT Device Tester v4.2.0 mit GGV2Q v2.0.0 veröffentlicht	Version 4.2.0 von IDT für V2 ist verfügbar. AWS IoT Greengrass Diese Version enthält die AWS IoT Greengrass V2 Qualification Suite (GGV2Q) v2.0.0 und unterstützt optionale Qualifizierungstests für Docker-Komponenten, maschinelles Lernen und Stream Manager.	14. Juli 2021
AWS IoT Greengrass Die IPC-Kernbibliothek ist in C++ v2 verfügbar AWS IoT Device SDK	Version 1.13.0 von AWS IoT Device SDK für C++ v2 unterstützt AWS IoT Greengrass Core IPC, sodass Sie Komponenten in C++ entwickeln können, die mit der Core-Software interagieren. AWS IoT Greengrass	14. Juli 2021

SageMaker Die Edge Manager-Komponente v1.0.2 wurde veröffentlicht	Die Amazon SageMaker Edge Manager-Komponente v1.0.2 ist verfügbar. Diese Version aktualisiert das Installat ionsskript im Komponent enlebenszyklus. Auf Ihren Kerngeräten muss jetzt Python 3.6 oder höher, auch `pip` für Ihre Version von Python, auf dem Gerät installiert sein, bevor Sie diese Komponente bereitstellen.	12. Juli 2021
Support-Update AWS IoT Device Tester für AWS IoT Greengrass V2	IDT für AWS IoT Greengras s V2 Version 4.1.0 unterstüt zt jetzt die Verwendung von Greengrass Nucleus Version 2.3.0 für die Gerätequalifizieru ng.	8. Juli 2021
Dockerfile und Docker-Images für die Core-Software v2.3.0 AWS IoT Greengrass	Das Dockerfile und das Docker-Image für die Core-Software v2.3.0 sind jetzt verfügbar. AWS IoT Greengrass	7. Juli 2021
AWS verwaltete Richtlinien	Es wurden Informationen zu AWS verwalteten Richtlini en für hinzugefügt AWS IoT Greengrass.	2. Juli 2021
Neue empfohlene JVM-Optio nen	Es wurden Informationen zu empfohlenen JVM-Optionen zur Steuerung der Speicherz uweisung für die AWS IoT Greengrass Core-Software hinzugefügt.	30. Juni 2021

AWS IoT Greengrass Core v2.3.0-Softwareupdate	Diese Version enthält Version 2.3.0 der Greengras s Nucleus-Komponente und von Updates AWS bereitges tellte Komponenten. Sie bietet Unterstützung für umfangrei che Dokumente zur Konfigura tion von Komponenten in Bereitstellungen.	29. Juni 2021
Dockerfile und Docker-Images für die Kernsoftware v2.2.0 AWS IoT Greengrass	Das Dockerfile und das Docker-Image für die Core-Software v2.2.0 sind jetzt verfügbar. AWS IoT Greengrass	28. Juni 2021
AWS IoT Device Tester v4.1.0 mit GGV2Q v1.1.1 veröffent licht	Version 4.1.0 von IDT für V2 ist verfügbar. AWS IoT Greengrass Diese Version enthält die AWS IoT Greengrass V2 Qualification Suite (GGV2Q) v1.1.1 und unterstützt die Verwendun g von Greengrass Nucleus v2.2.0, v2.1.0 und v2.0.5 für die Gerätequalifizierung.	18. Juni 2021

AWS IoT Greengrass Core v2.2.0-Softwareupdate	Diese Version enthält Version 2.2.0 der Greengrass Nucleus-Komponente und von Updates AWS bereitgestellte Komponenten. Sie enthält Komponenten, die Sie bereitstellen können, um Unterstützung für Client-Geräte hinzuzufügen und den lokalen Shadow-Dienst hinzuzufügen.	18. Juni 2021
Lambda Launcher v2.0.6 veröffentlicht	Version 2.0.6 der Lambda-Launcher-Komponente ist verfügbar. Diese Version beinhaltet Leistungsverbesserungen und Fehlerkorrekturen.	13. Juni 2021
Neue SageMaker Edge Manager-Komponente veröffentlicht	Version 1.0.0 der Amazon SageMaker Edge Manager-Komponente ist verfügbar für AWS IoT Greengrass. Diese Komponente installiert die SageMaker Edge Manager-Agent-Binärdatei auf Greengrass-Core-Geräten.	10. Juni 2021
Typen von Komponenten	Informationen zu Komponententypen wurden in hinzugefügt AWS IoT Greengrass. Der Komponententyp gibt an, wie die AWS IoT Greengrass Core-Software eine Komponente ausführt.	3. Juni 2021

AWS IoT Device Tester v4.0.2 mit GGV2Q v1.1.0 veröffentlicht	Version 4.0.2 von IDT für V2 ist verfügbar. AWS IoT Greengrass Diese Version enthält die AWS IoT Greengrass V2 Qualification Suite (GGV2Q) v1.1.0 und unterstützt die Verwendung von Greengrass Nucleus v2.1.0 mit Greengrass CLI v2.1.0 für die Gerätequalifizierung. Dies beinhaltet auch neue erforderliche Testgruppen für MQTT und Lambda sowie andere kleinere Bugfixes und Verbesserungen.	5. Mai 2021
Dockerfile und Docker-Images für die Core-Software v2.1.0 AWS IoT Greengrass	Das Dockerfile und das Docker-Image für die Core-Software v2.1.0 sind jetzt verfügbar. AWS IoT Greengrass Mit dem Docker-Image können Sie die AWS IoT Greengrass Core-Software in einem Docker-Container ausführen, der Amazon Linux 2 als Basisbetriebssystem verwendet.	27. April 2021

AWS IoT Greengrass Softwareupdate Core v2.1.0	Diese Version enthält Version 2.1.0 der Greengrass Nucleus-Komponente und von Updates AWS bereitgestellte Komponenten. Es enthält eine neue Komponente, mit der Sie Docker-Images aus privaten Amazon ECR-Repositorys herunterladen können, sowie neue Beispielkomponenten zur Durchführung von Inferenze n für maschinelles Lernen mit Lite. TensorFlow	26. April 2021
Beispielkomponente, die Secrets Manager verwendet	Es wurde eine Beispielk omponente hinzugefügt, die den Wert eines AWS Secrets Manager Secrets ausgibt, das Sie auf einem Core-Gerät bereitstellen.	8. April 2021
Minimale AWS IoT Richtlinie für Greengrass-Kerngeräte	Es wurden Informationen zu den Mindestberechtigungen hinzugefügt, die zur Unterstüt zung der grundlegenden Greengrass-Funktionalität auf einem Kerngerät erforderlich sind.	2. April 2021
Abonnieren Sie IPC-Event-Streams	Es wurden Informationen zur Verwendung von IPC-Operationen (Interprocess Communication) hinzugefügt, um Ereignisstreams auf einem Greengrass-Core-Gerät zu abonnieren.	01. April 2021

Support-Update AWS IoT Device Tester für AWS IoT Greengrass	IDT für AWS IoT Greengrass V2 Version 4.0.1 unterstützt jetzt die Verwendung von Greengrass Nucleus Version 2.0.5 mit Greengrass CLI Version 2.0.5 für die Gerätequalifizierung.	17. März 2021
Erstellen Sie benutzerdefinierte Komponenten, die Stream Manager verwenden	Es wurden Informationen zur Konfiguration von Komponentenrezepten und Artefakten hinzugefügt, um Anwendungen zur Verwaltung von Datenströmen zu entwickeln.	9. März 2021
AWS IoT Greengrass Softwareupdate Core v2.0.5	Diese Version enthält Version 2.0.5 der Greengrass Nucleus-Komponente und von Updates AWS bereitgestellte Komponenten. Es behebt ein Problem mit der Netzwerk-Proxyunterstützung und ein Problem mit dem Greengrass-Datenebenen-Endpunkt in AWS chinesischen Regionen.	9. März 2021
Referenz zur Umgebungsvariablen der Komponente	Es wurden Informationen zu den Umgebungsvariablen hinzugefügt, die die AWS IoT Greengrass Core-Software für Komponenten festlegt. Sie können diese Umgebungsvariablen verwenden, um den Dingnamen und die Greengrass Nucleus-Version abzurufen. AWS-Region	23. Februar 2021

Manuelle Installation	Es wurden Informationen darüber hinzugefügt, wie erforderliche AWS Ressourcen manuell erstellt oder hinter einer Firewall oder einem Netzwerk-Proxy installiert werden können. Wenn Sie eine manuelle Installation verwenden, müssen Sie dem Installateur nicht die Erlaubnis geben, Ressourcen in Ihrem zu erstellen AWS-Konto, da Sie die erforderlichen Ressourcen AWS IoT und die IAM-Ressourcen selbst erstellen. Sie können Ihr Gerät auch so konfigurieren, dass es eine Verbindung über Port 443 oder über einen Netzwerk-Proxy herstellt.	17. Februar 2021
AWS IoT Greengrass Update der IPC-Kernbibliothek AWS IoT Device SDK für Python v2	Version 1.5.4 von AWS IoT Device SDK for Python v2 vereinfacht die Schritte, die erforderlich sind, um eine Verbindung zum AWS IoT Greengrass Core IPC-Dienst herzustellen.	11. Februar 2021
Support-Update AWS IoT Device Tester für AWS IoT Greengrass	IDT für AWS IoT Greengrass V2 Version 4.0.1 unterstützt jetzt die Verwendung von Greengrass Nucleus Version 2.0.4 mit Greengrass CLI Version 2.0.4 für die Gerätequalifizierung.	5. Februar 2021

Neues Tutorial zum Import von Lambda-Funktionen	Es wurde ein neues konsolenbasiertes Tutorial hinzugefügt, um eine Lambda-Funktion als Komponente zu importieren, die auf einem Greengrass-Core-Gerät ausgeführt wird.	5. Februar 2021
AWS IoT Greengrass Core-Softwareupdate v2.0.4	Diese Version stellt Version 2.0.4 der Greengrass Nucleus-Komponente bereit. Sie enthält den neuen `greengrasDataPlanePort` Parameter zur Konfiguration der HTTPS-Kommunikation über Port 443 und behebt Fehler. Die minimale IAM-Richtlinie erfordert jetzt das `iam:GetPolicy` und, `sts:GetCallerIdentity` wenn das AWS IoT Greengrass Core-Softwareinstallationsprogramm ausgeführt wird. `--provision true`	4. Februar 2021

Neue Secure Tunneling-Komponente veröffentlicht	Version 1.0.0 der Secure Tunneling-Komponente ist verfügbar für. AWS IoT Greengrass Diese AWS bereitgestellte Komponente verwendet AWS IoT sicheres Tunneling, um eine sichere bidirektionale Kommunikation mit einem Greengrass-Core-Gerät herzustellen, das sich hinter eingeschränkten Firewalls befindet.	21. Januar 2021
AWS IoT Device TesterAWS IoT Greengrass für v4.0.1 veröffentlicht	Version 4.0.1 von IDT für AWS IoT Greengrass V2 ist verfügbar. Mit dieser Version können Sie IDT verwenden, um Ihre benutzerdefinierten Testsuiten für die Geräteevaluierung zu entwickeln und auszuführen. Dazu gehören auch codesignierte IDT-Anwendungen für macOS und Windows.	22. Dezember 2020
Erste Veröffentlichung von AWS IoT Greengrass Version 2	AWS IoT Greengrass V2 ist eine neue Hauptversion von AWS IoT Greengrass. Diese Version bietet mehrere Funktionen wie modulare Softwarekomponenten und kontinuierliche Bereitstellungen. Diese Funktionen erleichtern Ihnen die Entwicklung und Verwaltung von Edge-Anwendungen.	15. Dezember 2020

AWS Glossar

Die neueste AWS Terminologie finden Sie im AWS Glossar in der AWS-Glossar Referenz.

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.