Entwicklerhandbuch, Version 2

AWS IoT Greengrass

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

AWS IoT Greengrass: Entwicklerhandbuch, Version 2

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

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

Table of Contents

Was ist AWS IoT Greengrass? 1
Neue Features 1
Für Erstanwender 2
Für bestehende Benutzer 2
Wie AWS IoT Greengrass funktioniert 2
Die wichtigsten Konzepte 3
Funktionen von AWS IoT Greengrass 6
Kompatibilität der Greengrass-Funktionen 7
Wählen Sie Ihre AWS IoT Greengrass Nucleus-Laufzeit 16
Grüngraskern 17
Greengrass Nucleus Lite 17
Was ist neu in Version 2 21
AWS IoT Greengrass Softwareupdate Core v2.16.1 23
Öffentliche Komponenten-Updates 24
AWS IoT Greengrass Softwareupdate Core v2.16.0 25
Öffentliche Komponenten-Updates 25
AWS IoT Greengrass Softwareupdate Core v2.15.1 27
Öffentliche Komponenten-Updates 27
AWS IoT Greengrass Softwareupdate Core v2.15.0 28
Öffentliche Komponenten-Updates 28
AWS IoT Greengrass Softwareupdate Core v2.14.3 31
Öffentliche Komponenten-Updates 31
AWS IoT Greengrass Softwareupdate Core v2.14.2 33
Öffentliche Komponenten-Updates 33
AWS IoT Greengrass Softwareupdate Core v2.14.1 34
Öffentliche Komponenten-Updates 34
AWS IoT Greengrass Softwareupdate Core v2.14.0 36
Öffentliche Komponenten-Updates 38
AWS IoT Greengrass Softwareupdate Core v2.13.0 40
Öffentliche Komponenten-Updates 41
AWS IoT Greengrass Softwareupdate Core v2.12.6 42
Öffentliche Komponenten-Updates 43
AWS IoT Greengrass Softwareupdate Core v2.12.5 44
Öffentliche Komponenten-Updates 44

AWS IoT Greengrass Softwareupdate Core v2.12.4 45
Öffentliche Komponenten-Updates 45
AWS IoT Greengrass Softwareupdate Core v2.12.3 46
Öffentliche Komponenten-Updates 47
AWS IoT Greengrass Softwareupdate Core v2.12.2 49
Öffentliche Komponenten-Updates 49
AWS IoT Greengrass Softwareupdate Core v2.12.1 50
Öffentliche Komponenten-Updates 51
AWS IoT Greengrass Softwareupdate Core v2.12.0 52
Öffentliche Komponenten-Updates 53
AWS IoT Greengrass Softwareupdate Core v2.11.3 54
Öffentliche Komponenten-Updates 54
AWS IoT Greengrass Softwareupdate Core v2.11.2 55
Öffentliche Komponenten-Updates 56
AWS IoT Greengrass Softwareupdate Core v2.11.1 57
Öffentliche Komponenten-Updates 57
AWS IoT Greengrass Softwareupdate Core v2.11.0 58
Öffentliche Komponenten-Updates 59
AWS IoT Greengrass Softwareupdate Core v2.10.3 60
Öffentliche Komponenten-Updates 60
AWS IoT Greengrass Softwareupdate Core v2.10.2 61
Öffentliche Komponenten-Updates 61
AWS IoT Greengrass Softwareupdate Core v2.10.1 63
Öffentliche Komponenten-Updates 64
AWS IoT Greengrass Softwareupdate Core v2.10.0 65
Öffentliche Komponenten-Updates 65
AWS IoT Greengrass Softwareupdate Core v2.9.6 67
Öffentliche Komponenten-Updates 67
AWS IoT Greengrass Softwareupdate Core v2.9.5 68
Öffentliche Komponenten-Updates 68
AWS IoT Greengrass Softwareupdate Core v2.9.4 70
Öffentliche Komponenten-Updates 70
AWS IoT Greengrass Softwareupdate Core v2.9.3 71
Öffentliche Komponenten-Updates 71
AWS IoT Greengrass Softwareupdate Core v2.9.2 72
Öffentliche Komponenten-Updates 73

AWS IoT Greengrass Softwareupdate Core v2.9.1 73
Öffentliche Komponenten-Updates 74
AWS IoT Greengrass Softwareupdate Core v2.9.0 75
Öffentliche Komponenten-Updates 76
AWS IoT Greengrass Softwareupdate Core v2.8.1 78
Öffentliche Komponenten-Updates 78
AWS IoT Greengrass Softwareupdate Core v2.8.0 79
Öffentliche Komponenten-Updates 80
AWS IoT Greengrass Softwareupdate Core v2.7.0 82
Öffentliche Komponenten-Updates 82
AWS IoT Greengrass Softwareupdate Core v2.6.0 85
Öffentliche Komponenten-Updates 86
AWS IoT Greengrass Softwareupdate Core v2.5.6 90
Öffentliche Komponenten-Updates 90
AWS IoT Greengrass Softwareupdate Core v2.5.5 92
Öffentliche Komponenten-Updates 92
AWS IoT Greengrass Softwareupdate Core v2.5.4 93
Öffentliche Komponenten-Updates 93
AWS IoT Greengrass Softwareupdate Core v2.5.3 94
Öffentliche Komponenten-Updates 95
AWS IoT Greengrass Softwareupdate Core v2.5.2 96
Öffentliche Komponenten-Updates 96
AWS IoT Greengrass Softwareupdate Core v2.5.1 98
Öffentliche Komponenten-Updates 98
AWS IoT Greengrass Softwareupdate Core v2.5.0 99
Updates zur Plattformunterstützung 100
Öffentliche Komponenten-Updates 101
AWS IoT Greengrass Softwareupdate Core v2.4.0 105
Öffentliche Komponenten-Updates 106
AWS IoT Greengrass Softwareupdate Core v2.3.0 108
Öffentliche Komponenten-Updates 109
AWS IoT Greengrass Softwareupdate Core v2.2.0 110
Öffentliche Komponenten-Updates 111
AWS IoT Greengrass Softwareupdate Core v2.1.0 114
Updates zur Plattformunterstützung 115
Öffentliche Komponenten-Updates 115

AWS IoT Greengrass Softwareupdate Core v2.0.5 ... 123
Öffentliche Komponenten-Updates ... 123
AWS IoT Greengrass Softwareupdate Core v2.0.4 ... 124
Öffentliche Komponenten-Updates ... 124
Migrieren Sie von Version 1 ... 126
Überblick über die Migration ... 126
Unterschiede zwischen V1 und V2 ... 128
Wählen Sie Ihre Laufzeit ... 140
Kompatibilitätsmatrix für die Ereignisquelle ... 140
Entscheidungsablauf bei der Auswahl der Laufzeit ... 141
Nächste Schritte ... 142
Testgerät einrichten ... 142
Testgerät einrichten (Greengrass Nucleus) ... 143
Testgerät einrichten (Greengrass Nucleus Lite) ... 149
Aktualisieren Sie V1-Kerngeräte auf V2 ... 218
Schritt 1: Installieren Sie die AWS IoT Greengrass Core-Software v2.x ... 219
Schritt 2: Stellen Sie Greengrass V2-Komponenten auf den Kerngeräten bereit ... 222
Erste Schritte ... 225
Voraussetzungen ... 226
Schritt 1: Richten Sie ein AWS Konto ein ... 228
Melde dich an für ein AWS-Konto ... 228
Erstellen eines Benutzers mit Administratorzugriff ... 228
Schritt 2: Richten Sie Ihre Umgebung ein ... 230
Schritt 3: Installieren Sie die AWS IoT Greengrass Core-Software ... 236
Installieren Sie die AWS IoT Greengrass Core-Software (Konsole) ... 237
Installieren Sie die AWS IoT Greengrass Core-Software (CLI) ... 244
Führen Sie die Greengrass-Software aus (Linux) ... 249
Überprüfen Sie die Greengrass CLI-Installation auf dem Gerät ... 250
Schritt 4: Entwickeln und testen Sie eine Komponente auf Ihrem Gerät ... 252
Schritt 5: Erstellen Sie Ihre Komponente im AWS IoT Greengrass Service ... 264
Schritt 6: Stellen Sie Ihre Komponente bereit ... 276
Nächste Schritte ... 282
Einrichtung von Greengrass-Core-Geräten ... 283
Unterstützte Plattformen ... 283
Anforderungen an Speichergeräte ... 284
Anforderungen an die Lambda-Funktion ... 284

Richten Sie eine ein AWS-Konto ... 286
Installieren Sie die AWS IoT Greengrass Core-Software ... 288
Installieren Sie mit automatischer Bereitstellung ... 291
Installieren Sie mit manueller Bereitstellung ... 308
Installation mit Flottenbereitstellung ... 348
Installieren Sie mit benutzerdefinierter Bereitstellung ... 396
Argumente des Installers ... 415
Führen Sie die AWS IoT Greengrass Core-Software aus ... 420
Überprüfen Sie, ob die AWS IoT Greengrass Core-Software als Systemdienst ausgeführt wird ... 420
Führen Sie die AWS IoT Greengrass Core-Software als Systemdienst aus ... 422
Führen Sie die AWS IoT Greengrass Core-Software ohne Systemdienst aus ... 423
AWS IoT Greengrass In Docker ausführen ... 423
Unterstützte Plattformen und Anforderungen ... 424
Software wird heruntergeladen ... 425
Wählen Sie aus, wie Ressourcen bereitgestellt werden AWS ... 425
Erstellen Sie das AWS IoT Greengrass Image aus einem Dockerfile ... 426
AWS IoT Greengrass In Docker mit automatischer Bereitstellung ausführen ... 433
AWS IoT Greengrass In Docker mit manueller Bereitstellung ausführen ... 442
Fehlerbehebung AWS IoT Greengrass in einem Docker-Container ... 461
Konfiguration der AWS IoT Greengrass Core-Software ... 465
Stellen Sie die Greengrass Nucleus-Komponente bereit ... 465
Den Greengrass Nucleus als Systemdienst konfigurieren ... 465
Steuern Sie die Speicherzuweisung mit JVM-Optionen ... 469
Konfigurieren Sie den Benutzer, der die Komponenten ausführt ... 472
Konfigurieren Sie Systemressourcenlimits ... 477
Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy ... 480
Verwenden Sie ein Gerätezertifikat, das von einer privaten Zertifizierungsstelle signiert wurde ... 488
Konfigurieren Sie MQTT-Timeouts und Cache-Einstellungen ... 488
Greengrass Nucleus im Netzwerk IPv6 konfigurieren ... 489
Aktualisieren Sie die AWS IoT Greengrass Core-Software (OTA) ... 490
Voraussetzungen ... 490
Überlegungen zu Kerngeräten ... 491
Aktualisierungsverhalten von Greengrass Nucleus ... 491
Führen Sie ein OTA-Update durch ... 493

Deinstallieren Sie die AWS IoT Greengrass Core-Software 493
Lernprogramme 497
Video-Tutorials 497
Videos 497
Entwickeln Sie eine Komponente, die Komponenten-Updates verzögert 498
Voraussetzungen 499
Schritt 1: Installieren Sie das Greengrass Development Kit CLI 501
Schritt 2: Entwickeln Sie eine Komponente, die Updates verzögert 501
Schritt 3: Veröffentlichen Sie die Komponente im AWS IoT Greengrass Service 511
Schritt 4: Bereitstellen und Testen der Komponente auf einem Kerngerät 514
Interagieren Sie mit lokalen IoT-Geräten über MQTT 520
Voraussetzungen 520
Schritt 1: Überprüfen und aktualisieren Sie die AWS IoT Core-Geräterichtlinie 521
Schritt 2: Aktivieren Sie die Unterstützung für Client-Geräte 523
Schritt 3: Client-Geräte Connect 529
Schritt 4: Entwickeln Sie eine Komponente, die mit Client-Geräten kommuniziert 533
Schritt 5: Entwickeln Sie eine Komponente, die mit Schatten auf Client-Geräten interagiert . 540
Schützen Sie Greengrass Nucleus mit TPM 567
Voraussetzungen 567
Schritt 1: Installieren Sie TPM2 Tools und Abhängigkeiten 568
Schritt 2: Initialisieren Sie den PKCS #11 -Speicher und erstellen Sie einen Slot 568
Schritt 3: Erstellen Sie ein Token und einen Schlüssel 570
Schritt 4: Generieren Sie eine Zertifikatsignieranforderung (CSR) 572
Schritt 5: Erstellen Sie das Ding-Zertifikat 573
Schritt 6: Importieren Sie das Ding-Zertifikat in TPM 573
Schritt 7: Erfassen Sie die PKCS #11 -Objekt-URL 574
Schritt 8: Greengrass mit TPM2 Support konfigurieren und installieren 575
Schritt 9: Überprüfen Sie die Installation 577
Fehlerbehebung 577
Nächste Schritte 578
Schützen Sie Greengrass Nucleus Lite mit TPM 578
Voraussetzungen 579
Schritt 1: Richten Sie eine NitroTPM-Instanz ein 579
Schritt 2: TPM-Tools installieren und konfigurieren 580
Schritt 3: Den OpenSSL-Anbieter TPM2 konfigurieren 580
Schritt 4: Generieren Sie persistente TPM-Schlüssel 581

Schritt 5: Generieren Sie eine Zertifikatssignieranforderung (CSR) 581
Schritt 6: Erstellen Sie das Zertifikat aus CSR 582
Schritt 7: Greengrass Nucleus Lite mit TPM-Unterstützung einrichten 583
Fehlerbehebung 584
Nächste Schritte 585
Beginnen Sie mit SageMaker AI Edge Manager 585
Voraussetzungen 586
Richten Sie es im SageMaker AI Edge Manager ein 588
Erstellen Sie die Beispielkomponenten 589
Führen Sie ein Beispiel für die Inferenz zur Bildklassifizierung aus 591
Führen Sie eine Inferenz zur Klassifizierung von Beispielbildern durch 595
Voraussetzungen 596
Schritt 1: Abonnieren Sie das Thema Standardbenachrichtigungen 597
Schritt 2: Stellen Sie die TensorFlow Lite-Komponente zur Bildklassifizierung bereit 597
Schritt 3: Inferenzergebnisse anzeigen 600
Nächste Schritte 602
Führen Sie eine Beispielinferenz zur Bildklassifizierung für Bilder aus einer Kamera durch 602
Voraussetzungen 603
Schritt 1: Konfigurieren Sie das Kameramodul auf Ihrem Gerät 604
Schritt 2: Bestätigen Sie Ihr Abonnement für das Thema Standardbenachrichtigungen 607
Schritt 3: Ändern Sie die Konfiguration der TensorFlow Lite-Image-Klassifizierungskomponente und stellen Sie sie bereit 607
Schritt 4: Inferenzergebnisse anzeigen 609
Nächste Schritte 610
Komponenten 611
AWS-mitgelieferte Komponenten 611
Grüngraskern 631
Nucleus lite von Greengrass 678
Authentifizierung auf Client-Geräten 684
CloudWatch Metriken 766
AWS IoT Device Defender 792
Festplattenspooler 810
Docker-Anwendungsmanager 815
Edge-Anschluss für Kinesis Video Streams 826
Greengrass CLI 835
IP-Detektor 849

Firehose ... 860
Lambda-Launcher ... 879
Lambda-Manager ... 883
Lambda-Laufzeiten ... 893
Legacy-Abonnement-Router ... 895
Lokale Debug-Konsole ... 908
Protokollmanager ... 925
Komponenten für maschinelles Lernen ... 977
Modbus-RTU-Protokolladapter ... 1102
MQTT-Brücke ... 1134
MQTT 3.1.1-Broker (Moquette) ... 1159
MQTT 5-Broker (EMAX) ... 1167
Nucleus-Telemetrie-Sender ... 1185
PKCS #11 -Anbieter ... 1199
Geheimer Manager ... 1208
Sicheres Tunneling ... 1220
Schattenmanager ... 1232
Amazon SNS ... 1263
Stream-Manager ... 1281
Systemprotokollweiterleitung ... 1297
Systemmanager-Agent ... 1302
Token-Austauschdienst ... 1310
SiteWise IoT-OPC-UA-Kollektor ... 1314
IoT SiteWise OPC UA-Datenquellensimulator ... 1324
SiteWise IoT-Herausgeber ... 1327
SiteWise IoT-Prozessor ... 1344
Von Publishern unterstützte Komponenten ... 1360
AIShield. Edge ... 1361
EdgeLabs KI-Sensor ... 1361
Greengrass S3 Ingestor ... 1362
Komponenten der Gemeinschaft ... 1363
Greengrass-Entwicklungstools ... 1367
Greengrass-Entwicklungskit CLI ... 1368
Greengrass-Befehlszeilenschnittstelle ... 1400
Verwenden Sie das Greengrass Testing Framework ... 1420
Komponenten entwickeln ... 1437

Lebenszyklus der Komponenten 1439
Komponententypen 1439
Komponenten erstellen 1441
Testen Sie Komponenten mit lokalen Bereitstellungen 1454
Veröffentlichen Sie die bereitzustellenden Komponenten 1457
Interagieren Sie mit AWS Diensten 1464
Führen Sie einen Docker-Container aus 1468
Referenz zum Rezept 1492
Umgebungsvariablen 1525
Stellen Sie Komponenten auf Geräten bereit 1526
Bereitstellungen von Kerngeräten 1527
Auflösung der Plattformabhängigkeit 1527
Auflösung der Abhängigkeit von Komponenten 1527
Ein Gerät aus einer Dinggruppe entfernen 1535
Bereitstellungen 1536
Optionen für die Bereitstellung 1537
Erstellen von Bereitstellungen 1539
Unterbereitstellungen erstellen 1559
Bereitstellungen überarbeiten 1564
Bereitstellungen stornieren 1566
Überprüfen Sie den Bereitstellungsstatus 1567
Protokollierung und Überwachung 1571
Überwachungstools 1571
Überwachen Sie Greengrass-Protokolle 1572
Greifen Sie auf Dateisystemprotokolle zu 1573
Zugriffs-Logs CloudWatch 1575
Greifen Sie auf Systemdienstprotokolle zu 1578
Aktivieren Sie die Protokollierung in CloudWatch Logs 1579
Konfigurieren Sie die Protokollierung für AWS IoT Greengrass 1581
AWS CloudTrail Logs 1583
API-Aufrufe protokollieren mit CloudTrail 1583
AWS IoT Greengrass V2 Informationen in CloudTrail 1584
AWS IoT Greengrass Datenereignisse in CloudTrail 1585
AWS IoT Greengrass Verwaltungsereignisse in CloudTrail 1589
Grundlegendes zu AWS IoT Greengrass V2 Einträgen in Protokolldateien 1590
Sammeln Sie Telemetriedaten zur Systemintegrität 1591

Telemetrie-Metriken 1593
Konfigurieren Sie die Einstellungen des Telemetrie-Agents 1598
Abonnieren Sie Telemetriedaten in EventBridge 1598
Erhalten Sie Benachrichtigungen über den Bereitstellungs- und Komponentenstatus 1607
Ereignis zur Änderung des Bereitstellungsstatus 1608
Ereignis zur Änderung des Komponentenstatus 1610
Voraussetzungen für die Erstellung von EventBridge Regeln 1612
Benachrichtigungen zum Gerätestatus konfigurieren (Konsole) 1613
Benachrichtigungen zum Gerätestatus (CLI) konfigurieren 1614
Benachrichtigungen zum Gerätestatus konfigurieren (CloudFormation) 1615
Weitere Informationen finden Sie auch unter 1615
Überprüfen Sie den Status des Kerngeräts 1615
Überprüfen Sie den Zustand eines Kerngeräts 1616
Überprüfen Sie den Zustand einer Kerngerätegruppe 1617
Überprüfen Sie den Status der Komponenten des Kerngeräts 1617
Lambda-Funktionen ausführen 1619
Voraussetzungen 1620
Lambda-Funktionslebenszyklus konfigurieren 1620
Containerisierung der Lambda-Funktion konfigurieren 1622
Eine Lambda-Funktion als Komponente importieren (Konsole) 1625
Schritt 1: Wählen Sie eine Lambda-Funktion für den Import 1626
Schritt 2: Lambda-Funktionsparameter konfigurieren 1626
Schritt 3: (Optional) Geben Sie die unterstützten Plattformen für die Lambda-Funktion an . 1629
Schritt 4: (Optional) Komponentenabhängigkeiten für die Lambda-Funktion angeben 1630
Schritt 5: (Optional) Führen Sie die Lambda-Funktion in einem Container aus 1631
Schritt 6: Erstellen Sie die Lambda-Funktionskomponente 1632
Eine Lambda-Funktion als Komponente importieren (CLI) 1633
Schritt 1: Definieren Sie die Lambda-Funktionskonfiguration 1633
Schritt 2: Erstellen Sie die Lambda-Funktionskomponente 1654
Kommunizieren Sie mit dem Greengrass-Kern, anderen Komponenten und AWS IoT Core 1656
IPC-Client-Versionen 1657
Wird unterstützt SDKs 1658
Connect zum AWS IoT Greengrass Core IPC-Dienst her 1658
Autorisieren Sie Komponenten zur Ausführung von IPC-Vorgängen 1664
Platzhalter in Autorisierungsrichtlinien 1666
Rezeptvariablen in Autorisierungsrichtlinien 1666

Sonderzeichen in Autorisierungsrichtlinien ... 1667
Beispiele für Autorisierungsrichtlinien ... 1668
Abonnieren Sie IPC-Event-Streams ... 1671
Definieren Sie Abonnement-Handler ... 1672
Beispiel für Abonnement-Handler ... 1675
Bewährte IPC-Praktiken ... 1683
Lokale Nachrichten veröffentlichen/abonnieren ... 1684
Minimale SDK-Versionen ... 1685
Autorisierung ... 1685
PublishToTopic ... 1688
SubscribeToTopic ... 1696
Beispiele ... 1709
MQTT-Nachrichten veröffentlichen/abonnieren AWS IoT Core ... 1731
Minimale SDK-Versionen ... 1732
Autorisierung ... 1732
PublishToIoTCore ... 1737
SubscribeToIoTCore ... 1747
Beispiele ... 1761
Interagieren Sie mit dem Lebenszyklus von Komponenten ... 1769
Minimale SDK-Versionen ... 1770
Autorisierung ... 1771
UpdateState ... 1772
SubscribeToComponentUpdates ... 1772
DeferComponentUpdate ... 1774
PauseComponent ... 1775
ResumeComponent ... 1777
Interagieren Sie mit der Komponentenkonfiguration ... 1779
Minimale SDK-Versionen ... 1779
GetConfiguration ... 1780
UpdateConfiguration ... 1781
SubscribeToConfigurationUpdate ... 1782
SubscribeToValidateConfigurationUpdates ... 1783
SendConfigurationValidityReport ... 1784
Geheime Werte abrufen ... 1785
Minimale SDK-Versionen ... 1786
Autorisierung ... 1786

GetSecretValue ... 1788
Beispiele ... 1794
Interagiere mit lokalen Schatten ... 1800
Minimale SDK-Versionen ... 1801
Autorisierung ... 1801
GetThingShadow ... 1814
UpdateThingShadow ... 1821
DeleteThingShadow ... 1829
ListNamedShadowsForThing ... 1835
Lokale Bereitstellungen und Komponenten verwalten ... 1843
Minimale SDK-Versionen ... 1844
Autorisierung ... 1844
CreateLocalDeployment ... 1847
ListLocalDeployments ... 1850
GetLocalDeploymentStatus ... 1851
ListComponents ... 1852
GetComponentDetails ... 1853
RestartComponent ... 1854
StopComponent ... 1855
CreateDebugPassword ... 1856
Client-Geräte authentifizieren und autorisieren ... 1857
Minimale SDK-Versionen ... 1857
Autorisierung ... 1858
VerifyClientDeviceIdentity ... 1860
GetClientDeviceAuthToken ... 1860
AuthorizeClientDeviceAction ... 1862
SubscribeToCertificateUpdates ... 1863
Interagieren Sie mit lokalen IoT-Geräten ... 1865
Komponenten von Client-Geräten ... 1866
Client-Geräte mit Core-Geräten Connect ... 1869
Voraussetzungen ... 1871
Greengrass-Komponenten für die Unterstützung von Client-Geräten ... 1884
Konfigurieren Sie Cloud Discovery (Konsole) ... 1886
Cloud Discovery konfigurieren ()AWS CLI ... 1886
Client-Geräte zuordnen ... 1887
Authentifizierung von Clients im Offlinemodus ... 1889

Endpunkte von Kerngeräten verwalten ... 1891
Wählen Sie einen MQTT-Broker ... 1898
Verbindung zu einem MQTT-Broker herstellen ... 1900
Testen der Kommunikation ... 1902
Greengrass-Entdeckungs-API RESTful ... 1914
MQTT-Nachrichten zwischen Client-Geräten weiterleiten und AWS IoT Core ... 1921
Konfigurieren und implementieren Sie die MQTT-Bridge-Komponente ... 1922
MQTT-Nachrichten weiterleiten ... 1923
Interagieren Sie mit Client-Geräten in Komponenten ... 1924
Konfigurieren und implementieren Sie die MQTT-Bridge-Komponente ... 1925
Empfangen Sie MQTT-Nachrichten von Client-Geräten ... 1926
Senden Sie MQTT-Nachrichten an Client-Geräte ... 1927
Interagieren Sie mit Schatten auf Client-Geräten und synchronisieren Sie sie ... 1927
Voraussetzungen ... 1928
Aktivieren Sie den Shadow-Manager für die Kommunikation mit Client-Geräten ... 1928
Interagieren Sie mit Schatten auf Client-Geräten in Komponenten ... 1931
Synchronisieren Sie Schatten auf Client-Geräten mit AWS IoT Core ... 1931
IPv6 Für lokales Messaging verwenden ... 1932
Konfigurieren Sie den zu verwendenden IP-Detektor IPv6 ... 1932
Fehlerbehebung ... 1937
Probleme bei der Entdeckung von Greengrass ... 1937
Probleme mit der MQTT-Verbindung ... 1945
Interagiere mit Geräteschatten ... 1952
Interagiere mit Schatten in Komponenten ... 1953
Schattenstatus abrufen und ändern ... 1953
Reagieren Sie auf Änderungen des Schattenstatus ... 1954
Synchronisieren Sie lokale Geräteschatten mit AWS IoT Core ... 1955
Voraussetzungen ... 1956
Konfigurieren Sie die Shadow Manager-Komponente ... 1956
Synchronisieren Sie lokale Schatten ... 1958
Konfliktverhalten bei Shadow Merge ... 1958
Verwalten von Daten-Streams ... 1960
Stream-Management-Workflow ... 1961
Voraussetzungen ... 1962
Datensicherheit ... 1963
Lokale Datensicherheit ... 1963

Client-Authentifizierung 1963
Weitere Informationen finden Sie auch unter 1964
Konfigurieren des Stream-Managers 1964
Stream-Manager-Parameter 1964
Weitere Informationen finden Sie auch unter 1967
Erstellen Sie benutzerdefinierte Komponenten, die den Stream-Manager verwenden 1968
Definieren Sie Komponentenrezepte, die den Stream-Manager verwenden 1968
Stellen Sie im Anwendungscode eine Connect zum Stream Manager her 1980
Wird verwendet StreamManagerClient , um mit Streams zu arbeiten 1983
Erstellen eines Nachrichten-Streams 1985
Anhängen einer Nachricht 1988
Lesen von Nachrichten 1995
Auflisten von Streams 1997
Beschreiben eines Nachrichten-Streams 1999
Nachrichtenstream aktualisieren 2001
Löschen eines Nachrichten-Streams 2005
Weitere Informationen finden Sie auch unter 2007
Exportieren Sie Konfigurationen für unterstützte Cloud-Ziele 2007
Durchführen von Machine Learning-Inferenzen 2025
Wie funktioniert AWS IoT Greengrass ML-Inferenz 2025
Was ist in AWS IoT Greengrass Version 2 anders? 2027
Voraussetzungen 2027
Unterstützte Modellquellen 2028
Unterstützte Laufzeiten 2028
Komponenten für maschinelles Lernen 2029
Verwenden Sie SageMaker AI Edge Manager 2037
Funktionsweise 2038
Voraussetzungen 2039
Beginnen Sie mit SageMaker AI Edge Manager 2041
Passen Sie Ihre Komponenten für maschinelles Lernen an 2041
Ändern Sie die Konfiguration einer öffentlichen Inferenzkomponente 2042
Verwenden Sie ein benutzerdefiniertes Modell mit der Beispiel-Inferenzkomponente 2044
Erstellen Sie benutzerdefinierte Komponenten für maschinelles Lernen 2048
Erstellen Sie eine benutzerdefinierte Inferenzkomponente 2051
Fehlerbehebung 2058
Bibliothek konnte nicht abgerufen werden 2059

Cannot open shared object file ... 2060
Error: ModuleNotFoundError: No module named '<library>' ... 2060
Es wurde kein CUDA-fähiges Gerät erkannt ... 2062
Keine solche Datei oder kein solches Verzeichnis ... 2062
RuntimeError: module compiled against API version 0xf but this version of NumPy is <version> ... 2063
picamera.exc.PiCameraError: Camera is not enabled ... 2063
Speicherfehler ... 2064
Speicherplatzfehler ... 2064
Timeout-Fehler ... 2064
Verwalten Sie Kerngeräte mit AWS Systems Manager ... 2065
Den Systems Manager Agent installieren ... 2066
Schritt 1: Ausführen allgemeiner Systems Manager-Einrichtungsschritte ... 2066
Schritt 2: Erstellen Sie eine IAM-Servicerolle für Systems Manager ... 2067
Schritt 3: Fügen Sie der Token-Exchange-Rolle Berechtigungen hinzu ... 2067
Schritt 4: Bereitstellen der Systems Manager Agent-Komponente ... 2072
Schritt 5: Überprüfen Sie die Registrierung des Kerngeräts mit Systems Manager ... 2075
Deinstallieren Sie den Systems Manager Agent ... 2076
Schritt 1: Das Kerngerät vom Systems Manager abmelden ... 2077
Schritt 2: Deinstallieren Sie die Systems Manager Agent-Komponente ... 2077
Schritt 3: Deinstallieren Sie die Systems Manager Agent-Software ... 2078
Sicherheit ... 2079
Datenschutz ... 2080
Datenverschlüsselung ... 2081
Integration von Hardware-Sicherheit ... 2083
Geräteauthentifizierung und -autorisierung ... 2096
X.509-Zertifikate ... 2097
AWS IoT Richtlinien ... 2098
Aktualisieren Sie die AWS IoT Richtlinie eines Kerngeräts ... 2104
Minimale AWS IoT Richtlinie ... 2109
Minimale AWS IoT Richtlinie zur Unterstützung von Client-Geräten ... 2112
Minimale AWS IoT Richtlinie für Client-Geräte ... 2114
Identity and Access Management ... 2116
Zielgruppe ... 2116
Authentifizierung mit Identitäten ... 2117
Verwalten des Zugriffs mit Richtlinien ... 2118

Weitere Informationen finden Sie auch unter 2120
Wie AWS IoT Greengrass funktioniert mit IAM 2120
Beispiele für identitätsbasierte Richtlinien 2125
Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS 2127
Minimale IAM-Richtlinie für das Installationsprogramm zur Bereitstellung von Ressourcen 2133
Servicerolle bei Greengrass 2137
AWS verwaltete Richtlinien 2146
Serviceübergreifende Confused-Deputy-Prävention 2152
Problembehandlung bei Identitäts- und Zugriffsproblemen 2153
Geräteverkehr über einen Proxy oder eine Firewall zulassen 2155
Endpunkte für den Basisbetrieb 2156
Endpunkte für die Installation mit automatischer Bereitstellung 2161
Endpunkte für AWS von -bereitgestellte Komponenten 2163
Compliance-Validierung 2163
FIPS-Endpunkte 2164
Aktivieren Sie FIPS-Endpunkte bei der Bereitstellung 2165
Installieren Sie Nucleus mit FIPS-Endpunkten mit manueller Ressourcenbereitstellung 2167
Installieren Sie FIPS-Endpunkte mit Flottenbereitstellung 2202
Installieren Sie FIPS-Endpunkte mit auto Ressourcenbereitstellung 2221
Komponenten von Erstanbietern zur FIPS-Konformität 2239
Ausfallsicherheit 2240
Sicherheit der Infrastruktur 2241
Konfigurations- und Schwachstellenanalyse 2242
Integrität des Codes 2242
VPC-Endpunkte (AWS PrivateLink) 2244
Überlegungen zu AWS IoT Greengrass VPC-Endpunkten 2245
Erstellen Sie einen VPC-Schnittstellen-Endpunkt für den Betrieb auf AWS IoT Greengrass Steuerungsebene 2245
Erstellen einer VPC-Endpunktrichtlinie für AWS IoT Greengrass 2246
Betreiben Sie ein AWS IoT Greengrass Kerngerät in VPC 2246
Bewährte Methoden für die Gewährleistung der Sicherheit 2253
Erteilen von Mindestberechtigungen 2253
Anmeldeinformationen in Greengrass-Komponenten nicht hartcodieren 2253
Keine Protokollierung sensibler Informationen 2253
Synchronisieren der internen Uhr Ihres Geräts 2254
Empfehlungen für die Cipher Suite 2254

Weitere Informationen finden Sie auch unter 2254
AWS IoT Device Tester Für AWS IoT Greengrass V2 verwenden 2255
AWS IoT Greengrass Qualifizierungssuite 2255
Benutzerdefinierte Testsuiten 2256
Unterstützte Versionen 2256
Aktuelle IDT-Version für V2 AWS IoT Greengrass 2257
Frühere IDT-Versionen für AWS IoT Greengrass 2257
Nicht unterstützte Versionen von für V2 AWS IoT Device TesterAWS IoT Greengrass 2258
Laden Sie IDT für AWS IoT Greengrass V2 herunter 2264
Laden Sie IDT manuell herunter 2264
Laden Sie IDT programmgesteuert herunter 2265
Verwenden Sie IDT, um die AWS IoT Greengrass Qualification Suite auszuführen 2271
Test-Suite-Versionen 2272
Beschreibung der Testgruppen 2272
Voraussetzungen 2275
Konfigurieren Sie Ihr Gerät für die Ausführung von IDT-Tests 2298
IDT-Einstellungen konfigurieren 2309
Führen Sie die AWS IoT Greengrass Qualifizierungssuite aus 2327
Verstehen von Ergebnissen und Protokollen 2331
Verwenden Sie IDT, um Ihre eigenen Testsuiten zu entwickeln und auszuführen 2334
Herunterladen der neuesten Version von IDT für AWS IoT Greengrass 2276
Arbeitsablauf bei der Erstellung einer Testsuite 2335
Tutorial: IDT-Beispiel-Testsuite erstellen und ausführen 2336
Tutorial: Entwickeln Sie eine einfache IDT-Testsuite 2342
Erstellen Sie Konfigurationsdateien für die IDT-Testsuite 2351
Konfigurieren Sie den IDT-Testorchestrator 2359
Konfigurieren Sie die IDT-Zustandsmaschine 2367
Ausführbare IDT-Testfalldateien erstellen 2392
Verwenden Sie den IDT-Kontext 2400
Einstellungen für Testläufer konfigurieren 2405
Debuggen und Ausführen benutzerdefinierter Testsuiten 2416
Überprüfen Sie die IDT-Testergebnisse und -protokolle 2420
IDT-Nutzungsmetriken 2426
Problembehandlung IDT für V2 AWS IoT Greengrass 2433
Wo kann man nach Fehlern suchen 2434
IDT für AWS IoT Greengrass V2-Fehler beheben 2435

Unterstützungsrichtlinie AWS IoT Device Tester für AWS IoT Greengrass 2443
IoT-Lösungen auf Basis von Greengrass 2444
Eurotech 2444
Fehlerbehebung 2445
Protokolle der AWS IoT Greengrass Kernsoftware und der Komponenten anzeigen 2445
AWS IoT Greengrass Kernprobleme mit der Software 2445
ThrottlingException von der ListDeployments API 2447
Das Kerngerät konnte nicht eingerichtet werden 2447
Die AWS IoT Greengrass Core-Software konnte nicht als Systemdienst gestartet werden . 2447
Nucleus kann nicht als Systemdienst eingerichtet werden 2448
Es konnte keine Verbindung hergestellt werden AWS IoT Core 2448
Fehler: Nicht genügend Arbeitsspeicher 2449
Greengrass CLI kann nicht installiert werden 2449
User root is not allowed to execute 2449
com.aws.greengrass.lifecyclemanager.GenericExternalService: Could not determine user/group to run with 2450
Failed to map segment from shared object: operation not permitted 2450
Der Windows-Dienst konnte nicht eingerichtet werden 2451
com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager 2451
com.aws.greengrass.deployment.IotJobsHelper: No connection available during subscribing to Iot Jobs descriptions topic. Will retry in sometime 2452
software.amazon.awssdk.services.iam.model.IamException: The security token included in the request is invalid 2452
software.amazon.awssdk.services.iot.model.IotException: User: <user> is not authorized to perform: iot:GetPolicy 2453
Error: com.aws.greengrass.shadowmanager.sync.model.FullShadowSyncRequest: Could not execute cloud shadow get request 2453
Operation aws.greengrass#<operation> is not supported by Greengrass 2454
java.io.FileNotFoundException: <stream-manager-store-root-dir>/stream_manager_metadata_store (Permission denied) 2455
com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: Private key or certificate with label <label> does not exist 2455
software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: User: <user> is not authorized to perform: secretsmanager:GetSecretValue on resource: <arn> . 2456
software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: Access to KMS is not allowed 2457

java.lang.NoClassDefFoundError: com/aws/greengrass/security/CryptoKeySpi 2457
com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: CKR_OPERATION_NOT_INITIALIZED 2457
Greengrass core device stuck on nucleus v2.12.3 2458
Greengrass nucleus v2.14.0 systemd template issue 2460
AWS IoT Greengrass Cloud-Probleme 2460
An error occurred (AccessDeniedException) when calling the CreateComponentVersion operation: User: arn:aws:iam::123456789012:user/<username> is not authorized to perform: null 2461
Invalid Input: Encountered following errors in Artifacts: {<s3ArtifactUri> = Specified artifact resource cannot be accessed} 2461
INACTIVE deployment status 2461
Hauptprobleme bei der Gerätebereitstellung 2462
Error: com.aws.greengrass.componentmanager.exceptions.PackageDownloadException: Failed to download artifact 2463
Error: com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchException: Integrity check for downloaded artifact failed. Probably due to file corruption. 2464
Error: com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException: Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements> 2465
software.amazon.awssdk.services.greengrassv2data.model.ResourceNotFoundException: The latest version of Component <componentName> doesn't claim platform <coreDevicePlatform> compatibility 2466
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 2466
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 2467
Info: com.aws.greengrass.deployment.exceptions.RetryableDeploymentDocumentDownloadException: Greengrass Cloud Service returned an error when getting full deployment configuration 2468

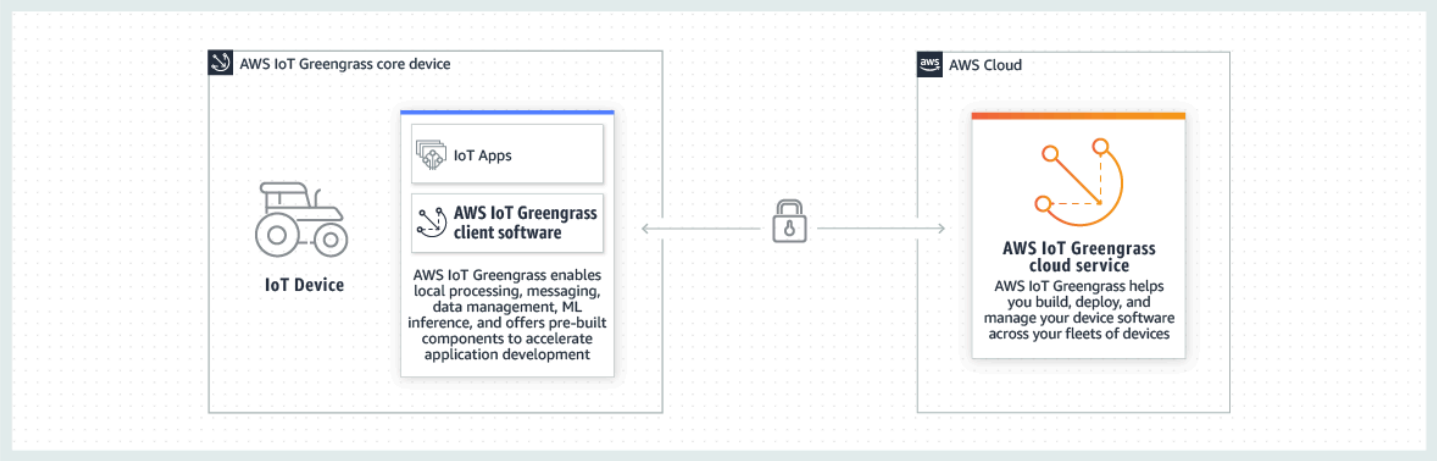
Warn: com.aws.greengrass.deployment.DeploymentService: Failed to get thing group hierarchy 2468
Info: com.aws.greengrass.deployment.DeploymentDocumentDownloader: Calling Greengrass cloud to get full deployment configuration 2469
Caused by: software.amazon.awssdk.services.greengrassv2data.model.GreengrassV2DataException: null (Service: GreengrassV2Data, Status Code: 403, Request ID: <some_request_id>, Extended Request ID: null) 2469
Probleme mit den wichtigsten Gerätekomponenten 2469
Warn: '<command>' is not recognized as an internal or external command 2470
Python-Skript protokolliert keine Nachrichten 2471
Die Komponentenkonfiguration wird nicht aktualisiert, wenn die Standardkonfiguration geändert wird 2472
awsiot.greengrasscoreipc.model.UnauthorizedError 2473
com.aws.greengrass.authorization.exceptions.AuthorizationException: Duplicate policy ID "<id>" for principal "<componentList>" 2474
com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 400) 2474
com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 403) 2476
com.aws.greengrass.tes.CredentialsProviderError: Could not load credentials from any providers 2477
Received error when attempting to retrieve ECS metadata: Could not connect to the endpoint URL: "<tokenExchangeServiceEndpoint>" 2477
copyFrom: <configurationPath> is already a container, not a leaf 2478
com.aws.greengrass.componentmanager.plugins.docker.exceptions.DockerLoginException: Error logging into the registry using credentials - 'The stub received bad data.' 2478
java.io.IOException: Cannot run program "cmd" ...: [LogonUser] The password for this account has expired. 2479
aws.greengrass.StreamManager: Instant exceeds minimum or maximum instant 2480
Probleme mit den Lambda-Funktionskomponenten des Kerngeräts 2481
The following cgroup subsystems are not mounted: devices, memory 2481
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> 2482
Die Version der Komponente wurde eingestellt 2482
Probleme mit der Greengrass-CLI 2483

java.lang.RuntimeException: Unable to create ipc client 2483
AWS CLI Probleme 2483
Error: Invalid choice: 'greengrassv2' 2483
Detaillierte Fehlercodes für die Bereitstellung 2484
Berechtigungsfehler 2486
Fehler bei der Anfrage 2487
Fehler bei der Rezeptur der Komponente 2490
AWS Komponentenfehler, Benutzerkomponentenfehler, Komponentenfehler 2492
Gerätefehler 2493
Abhängigkeitsfehler 2495
HTTP-Fehler 2496
Netzwerkfehler 2497
Nucleus-Fehler 2497
Serverfehler 2499
Fehler beim Cloud-Dienst 2499
Generische Fehler 2500
Unbekannter Fehler 2501
Detaillierte Statuscodes für Komponenten 2502
Markieren Ihrer -Ressourcen mit Tags (Markierungen) 2505
Verwenden von Tags in AWS IoT Greengrass V2 2505
Markieren Sie mit dem AWS-Managementkonsole 2505
Taggen Sie mit der AWS IoT Greengrass V2 API 2506
Verwenden von Tags mit IAM-Richtlinien 2507
AWS CloudFormation Ressourcen 2509
AWS IoT Greengrass und CloudFormation Vorlagen 2509
ComponentVersion Beispiel für eine Vorlage 2509
Beispiel für eine Bereitstellungsvorlage 2510
Erfahren Sie mehr über CloudFormation 2511
Open-Source-Software 2512
Dokumentverlauf 2513
AWS Glossar 2581
..... mmdlxxxii

Was ist AWS IoT Greengrass?

AWS IoT Greengrass ist ein Open-Source-Edge-Runtime- und Cloud-Dienst für das Internet der Dinge (IoT), mit dem Sie IoT-Anwendungen auf Ihren Geräten erstellen, bereitstellen und verwalten können. Sie können AWS IoT Greengrass damit Software entwickeln, die es Ihren Geräten ermöglicht, lokal auf die von ihnen generierten Daten zu reagieren, Prognosen auf der Grundlage von Modellen für maschinelles Lernen auszuführen und Gerätedaten zu filtern und zu aggregieren. AWS IoT Greengrass ermöglicht es Ihren Geräten, Daten näher am Ort ihrer Entstehung zu sammeln und zu analysieren, selbstständig auf lokale Ereignisse zu reagieren und sicher mit anderen Geräten im lokalen Netzwerk zu kommunizieren. Greengrass-Geräte können auch sicher mit dem kommunizieren AWS IoT Core und IoT-Daten dorthin exportieren. AWS Cloud Sie können Edge-Anwendungen mithilfe von vorgefertigten Softwaremodulen, sogenannten Komponenten, erstellen, die Ihre Edge-Geräte mit AWS Diensten oder Diensten von Drittanbietern verbinden können. AWS IoT Greengrass Sie können es auch verwenden AWS IoT Greengrass , um Ihre Software mithilfe von Lambda-Funktionen, Docker-Containern, systemeigenen Betriebssystemprozessen oder benutzerdefinierten Laufzeiten Ihrer Wahl zu verpacken und auszuführen.

Das folgende Beispiel zeigt, wie ein AWS IoT Greengrass Gerät mit dem interagiert. AWS Cloud



Neue Features

AWS IoT Greengrass V2 führt neue Funktionen 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 Erstnutzer von AWS IoT Greengrass

Wenn Sie noch nicht mit dem Thema AWS IoT Greengrass vertraut sind, empfehlen wir Ihnen, den folgenden Abschnitt zu lesen:

- Wie AWS IoT Greengrass funktioniert

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

Für bestehende Benutzer von AWS IoT Greengrass V1

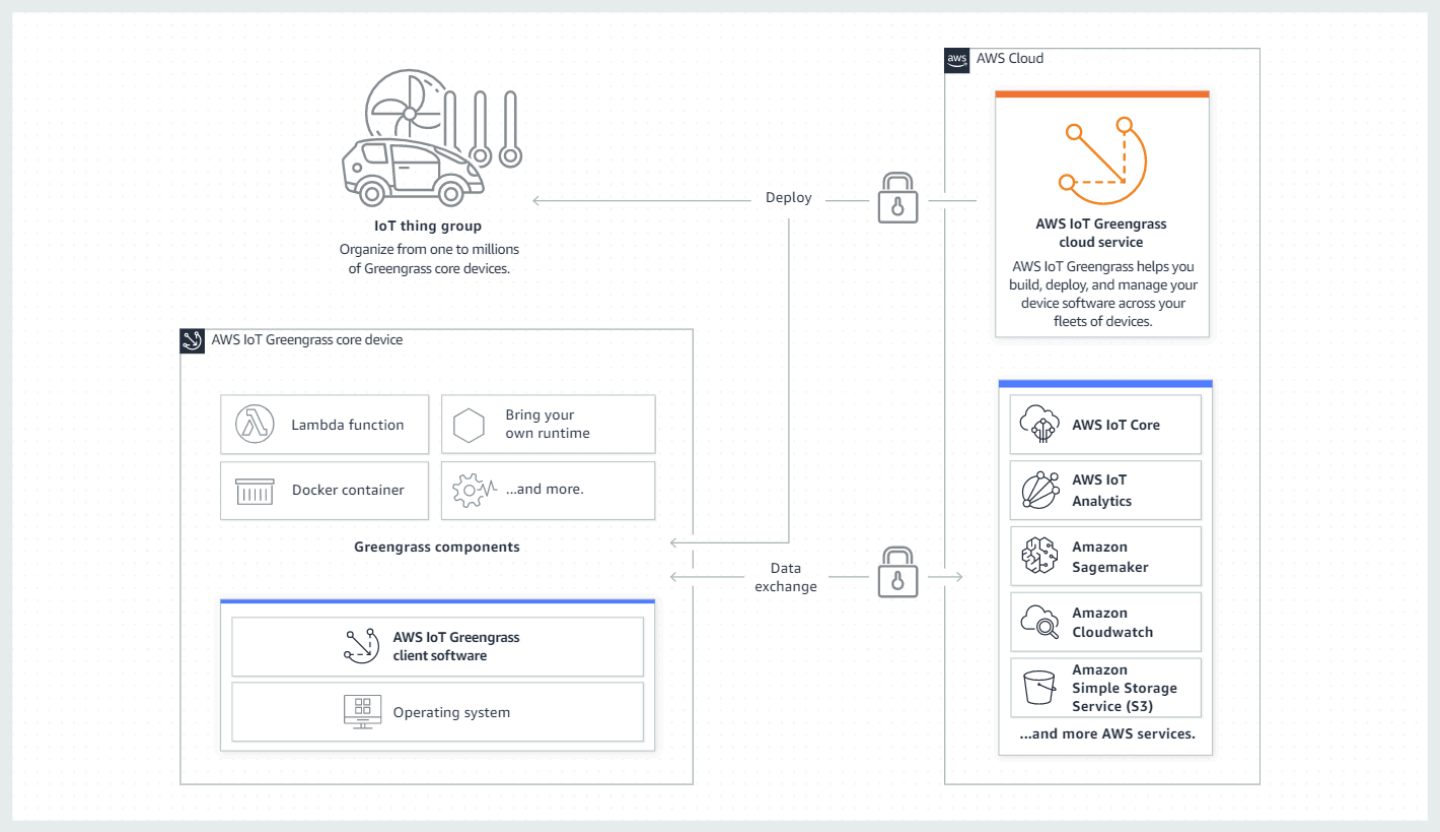
Hinweis zum Ende des Supports: Am 1. Oktober 2026 AWS endet der Support für AWS IoT Greengrass Version 1. Nach dem 1. Oktober 2026 können Sie nicht mehr auf die AWS IoT Greengrass V1 Konsole oder AWS IoT Greengrass V1 die Ressourcen zugreifen. Weitere Informationen zum Wechsel von Version 1 zu Version 2 finden Sie unterVon AWS IoT Greengrass Version 1 migrieren.

Wie AWS IoT Greengrass funktioniert

Die AWS IoT Greengrass Client-Software, auch AWS IoT Greengrass Core-Software genannt, läuft auf Windows- und Linux-basierten Distributionen wie Ubuntu oder Raspberry Pi OS für Geräte mit ARM- oder x86-Architekturen. Mit können Sie Geräte so programmieren AWS IoT Greengrass, dass sie lokal auf die von ihnen generierten Daten reagieren, Vorhersagen auf der Grundlage von Modellen für maschinelles Lernen ausführen und Gerätedaten filtern und aggregieren. AWS IoT Greengrass ermöglicht die lokale Ausführung von AWS Lambda Funktionen, Docker-Containern, systemeigenen 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 Greengrass Komponenten ermöglichen es Ihnen, eine Verbindung zu AWS Diensten und Drittanbieteranwendungen am Netzwerkrand herzustellen. Nachdem Sie Ihre IoT-Anwendungen entwickelt haben, AWS IoT Greengrass können Sie diese Anwendungen auf Ihrer Geräteflotte vor Ort remote bereitstellen, konfigurieren und verwalten.

Das folgende Beispiel zeigt, wie ein AWS IoT Greengrass Gerät mit dem AWS IoT Greengrass Cloud-Dienst und anderen AWS Diensten in der AWS Cloud interagiert.



Schlüsselkonzepte für AWS IoT Greengrass

Die folgenden Konzepte sind für das Verständnis und die Anwendung von grundlegender Bedeutung AWS IoT Greengrass:

AWS IoT Sache

Ein AWS IoT Ding ist eine Repräsentation eines bestimmten Geräts oder einer logischen Einheit. Informationen über eine Sache 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 eine AWS IoT-Sache. Sie können mehrere Kerngeräte zu Dinggruppen hinzufügen AWS IoT , 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 eine AWS IoT Sache. Das Kerngerät kann Daten von Client-Geräten, die eine Verbindung zu ihm herstellen, verarbeiten, filtern und aggregieren. Sie können das Core-Gerät so konfigurieren, dass es MQTT-Nachrichten zwischen Client-Geräten, dem AWS IoT Core Cloud-Dienst und Greengrass-Komponenten weiterleitet. Weitere Informationen finden Sie unter Interagieren Sie mit lokalen IoT-Geräten.

Client-Geräte können FreeRTOS ausführen oder die AWS IoT Device SDKoder Greengrass Discovery API verwenden, um Informationen über Kerngeräte abzurufen, mit denen sie sich verbinden können.

Greengrass-Komponente

Ein Softwaremodul, das auf einem Greengrass-Core-Gerät bereitgestellt wird und auf diesem ausgeführt wird. Die gesamte Software, mit der entwickelt und bereitgestellt wird, AWS IoT Greengrass wird als Komponente modelliert. AWS IoT Greengrass stellt vorgefertigte öffentliche Komponenten bereit, die Features und Funktionen 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 sie mithilfe des AWS IoT Greengrass Cloud-Dienstes auf einzelnen oder mehreren Core-Geräten bereitstellen. Sie können eine benutzerdefinierte Komponente erstellen und diese Komponente auf einem Kerngerät bereitstellen. Wenn Sie dies tun, lädt das Kerngerät die folgenden Ressourcen herunter, um die Komponente auszuführen:

- Rezept: Eine JSON- oder YAML-Datei, die das Softwaremodul beschreibt, indem sie Komponentendetails, Konfiguration und Parameter definiert.
- Artifact: Der Quellcode, die Binärdateien oder Skripts, die die Software definieren, die auf Ihrem Gerät ausgeführt werden soll. Sie können Artefakte von Grund auf neu erstellen, oder Sie können eine Komponente mithilfe einer Lambda-Funktion, eines Docker-Containers oder einer benutzerdefinierten Laufzeit erstellen.
- Abhängigkeit: Die Beziehung zwischen Komponenten, die es Ihnen ermöglicht, automatische Updates oder Neustarts abhängiger Komponenten zu erzwingen. Beispielsweise können Sie eine Komponente für die sichere Nachrichtenverarbeitung von einer Verschlüsselungskomponente abhängig machen. Dadurch wird sichergestellt, dass bei allen Aktualisierungen der Verschlüsselungskomponente die Nachrichtenverarbeitungskomponente automatisch aktualisiert und neu gestartet wird.

Weitere Informationen erhalten Sie unter AWS-mitgelieferte Komponenten und AWS IoT Greengrass Komponenten entwickeln.

Bereitstellung

Der Prozess zum Senden von Komponenten und zum Anwenden der gewünschten Komponentenkonfiguration auf ein Zielgerät, bei dem es sich um ein einzelnes Greengrass-Core-Gerät oder eine Gruppe von Greengrass-Core-Geräten 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 bestehende Bereitstellung klonen, um eine neue Bereitstellung zu erstellen, die dieselben Komponenten verwendet, aber für ein anderes 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 AWS IoT Greengrass Komponenten auf Geräten bereitstellen.

AWS IoT Greengrass Kernsoftware

Ab Version 2.14 stehen zwei alternative Implementierungen der Gerätelaufzeit zur AWS IoT Greengrass Verfügung, eine ausführbare Datei, die als Nucleus bezeichnet wird. Der erste und bisher einzige Nucleus ist in Java implementiert. Diese Option bietet die größte Portabilität zwischen Architekturen und Betriebssystemen. Sie ist jedoch auch von der Java Virtual Machine abhängig, was zu einem großen Speicherbedarf führt.

Der zweite, neu hinzugefügte Nucleus ist in C implementiert. Durch diese Wahl wird der Speicherbedarf erheblich reduziert. Es erfordert jedoch eine separate Verteilung (oder Kompilierung aus dem Quellcode) für verschiedene Zielarchitekturen und Betriebssysteme. Wenn es notwendig ist, die beiden zu unterscheiden, werden wir die erste Implementierung als Greengrass Nucleus und die letztere als Greengrass Nucleus Lite bezeichnen.

- Optionale Komponenten: Diese konfigurierbaren Komponenten werden von Ihren Edge-Geräten bereitgestellt AWS IoT Greengrass und ermöglichen zusätzliche Funktionen auf diesen Geräten. Je nach Ihren Anforderungen können Sie die optionalen Komponenten auswählen, die Sie auf Ihrem Gerät bereitstellen möchten, z. B. Datenstreaming, lokale Inferenz für maschinelles Lernen oder eine lokale Befehlszeilenschnittstelle. Weitere Informationen finden Sie unter AWS-mitgelieferte Komponenten.

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

Funktionen von AWS IoT Greengrass

AWS IoT Greengrass Version 2 besteht aus den folgenden Elementen:

- Softwareverteilungen
  - Die Greengrass Nucleus-Komponente, bei der es sich um die Mindestinstallation der AWS IoT Greengrass Core-Software handelt. Diese Komponente verwaltet die Bereitstellung, Orchestrierung und das Lebenszyklusmanagement von Greengrass-Komponenten.
  - Zusätzliche, optional AWS bereitgestellte Komponenten, die sich in Dienste, Protokolle und Software integrieren lassen.
  - Greengrass-Entwicklungstools, mit denen Sie benutzerdefinierte Greengrass-Komponenten erstellen, testen, erstellen, veröffentlichen und bereitstellen können.
  - Die AWS IoT Device SDK, die IPC-Bibliothek (Interprocess Communication) für benutzerdefinierte Greengrass-Komponenten und die Greengrass Discovery-Bibliothek für Client-Geräte enthält.
  - Das Stream Manager SDK, mit dem Sie Datenstreams auf Kerngeräten verwalten können.
- Cloud-Dienst
  - AWS IoT Greengrass V2 API
  - AWS IoT Greengrass V2 Konsole

AWS IoT Greengrass Kernsoftware

Sie können die AWS IoT Greengrass Core-Software, die auf Ihren Edge-Geräten ausgeführt wird, für folgende Zwecke verwenden:

- Verarbeiten Sie Datenströme auf dem lokalen Gerät mit automatischen Exporten in die AWS Cloud. Weitere Informationen finden Sie unter Datenströme auf Greengrass-Kerngeräten verwalten.
- Support MQTT-Messaging zwischen AWS IoT und Komponenten. Weitere Informationen finden Sie unter MQTT-Nachrichten veröffentlichen/abonnieren AWS IoT Core.
- Interagieren Sie mit lokalen Geräten, die sich über MQTT verbinden und kommunizieren. Weitere Informationen finden Sie unter Interagieren Sie mit lokalen IoT-Geräten.
- Support lokales Veröffentlichen und Abonnieren von Nachrichten 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 AWS IoT Greengrass Komponenten auf Geräten bereitstellen.
- Verwalten Sie die Lebenszyklen von Komponenten, z. B. mit Unterstützung für Installations- und Ausführungsskripte. Weitere Informationen finden Sie unter AWS IoT Greengrass Referenz zum Komponenten-Rezept.
- Führen Sie sichere over-the-air (OTA) -Softwareupdates der AWS IoT Greengrass Core-Software und der benutzerdefinierten Komponenten durch. Weitere Informationen erhalten Sie unter Aktualisieren Sie die AWS IoT Greengrass Core-Software (OTA) und AWS IoT Greengrass Komponenten auf Geräten bereitstellen.
- Sorgen Sie für eine sichere, verschlüsselte Speicherung lokaler Geheimnisse 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-Kerngeräte, indem AWS IoT Greengrass APIs Sie kontinuierliche Softwarebereitstellungen erstellen. Weitere Informationen finden Sie unter AWS IoT Greengrass Komponenten auf Geräten bereitstellen.

Einige Funktionen werden nur auf bestimmten Plattformen unterstützt. Weitere Informationen finden Sie unter Kompatibilität der Greengrass-Funktionen.

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

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

Kompatibilität der Greengrass-Funktionen

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 zur Einrichtung von Greengrass-Core-Geräten finden Sie unterEinrichtung von AWS IoT Greengrass Kerngeräten.

Messaging

Feature	Linux	Windows	Greengrass Nucleus Lite (Linux)
Tauschen Sie MQTT-Nachrichten zwischen Komponenten AWS IoT aus	Ja	Ja	Ja (außer für MQTT5 Erweiterungen)
Tauschen Sie lokale publish/subscribe Nachrichten zwischen Komponenten aus	Ja	Ja	Ja
Interagieren Sie mit lokalen IoT-Geräten über MQTT	Ja	Ja	Nein
Interagieren Sie mit lokalen Modbus-RTU-Geräten mithilfe der Modbus-RTU-Komponente	Ja	Nein	Nein

Sicherheit

Feature	Linux	Windows	Greengrass Lite (Linux)
Sichere Verbindungen mit Geräteauthentifizierung und Autorisierung	Ja	Ja	Ja

Feature	Linux	Windows	Greengrass Lite (Linux)
Stellen Sie sichere, verschlüsselte Geheimnisse bereit und greifen Sie auf sie zu AWS Secrets Manager	Ja	Ja	Nein
Verwenden Sie ein Hardware-Sicherheitsmodul (HSM), um den privaten Schlüssel und das Zertifikat des Geräts sicher zu speichern	Ja	Nein	Ja
Überwachen Sie Kerngeräte mit AWS IoT Device Defender	Ja	Ja	Nein
Verwenden Sie AWS Anmeldeinformationen, um mit AWS Diensten zu interagieren	Ja	Ja	Ja

Installation

Feature	Linux	Windows	Greengrass Lite (Linux)
Installation AWS IoT Greengrass mit automatischer Bereitstellung	Ja	Ja	Nein
Installation AWS IoT Greengrass mit manueller Bereitstellung	Ja	Ja	Ja
Installation AWS IoT Greengrass mit AWS IoT Flottenbereitstellung	Ja	Ja	Ja
Installation AWS IoT Greengrass mit benutzerdefinierter Bereitstellung	Ja	Ja	Ja
AWS IoT Greengrass In einem Docker-Container mit einem vorgefertigten Docker-Image ausführen	Ja	Nein	Nein

> **Note**
>
> AWS IoT Greengrass kann in einem systemd-fähigen Docker-Container installiert und ausgeführt werden.

Fernwartung und Updates

Feature	Linux	Windows	Greengass Lite (Linux)
Führen Sie sichere over-the-air (OTA) Softwareupdates durch	Ja	Ja	Ja
Verwalten Sie Kerngeräte mit AWS Systems Manager	Ja	Nein	Nein
Stellen Sie mit AWS IoT sicherem Tunneling eine Connect zu Kerngerät en her	Ja	Nein	Ja

Machine Learning

Feature	Linux	Windows	Greengrass Lite (Linux)
Führen Sie mit Amazon SageMaker AI Edge Manager Inferenzen für maschinelles Lernen durch	Ja	Ja	Nein
Führen Sie mit Amazon Lookout for Vision Inferenz für	Ja	Nein	Nein

Feature	Linux	Windows	Greengrass Lite (Linux)
maschinelles Lernen durch			
Führen Sie mit DLR Inferenzen für maschinelles Lernen durch	Ja	Ja	Nein
Führen Sie Inferenzen für maschinelles Lernen durch mit TensorFlow	Ja	Ja	Nein

Funktionen der Komponenten

Feature	Linux	Windows	Greengrass Lite (Linux)
Lambda-Funktionen bereitstellen und aufrufen	Ja	Nein	Nein
Führen Sie Docker-Container in Komponenten aus	Ja	Ja	Ja
Verarbeiten und exportieren Sie umfangreiche Datenströme mit dem Stream Manager	Ja	Ja	Ja

Feature	Linux	Windows	Greengrass Lite (Linux)
Verwalten Sie die Lebenszyklen von Komponenten mit Lebenszyklusskripten	Ja	Ja	Ja
Interagieren Sie mit Geräteschatten	Ja	Ja	Nein
Logs auf Amazon CloudWatch Logs hochladen	Ja	Ja	Ja
Laden Sie mithilfe der CloudWatch Metrikkomponente Daten in Amazon CloudWatch Metrics hoch	Ja	Ja	Ja
Veröffentlichen Sie Nachrichten mithilfe der Amazon SNS-Komponente in Amazon Simple Notification Service	Ja	Nein	Nein

Feature	Linux	Windows	Greengrass Lite (Linux)
Veröffentlichen Sie Daten mithilfe des Stream-Managers in Amazon Kinesis Data Streams-Lieferdatenströmen	Ja	Ja	Ja
Veröffentlichen Sie Daten mithilfe der Firehose-Komponente in Amazon Data Firehose-Lieferdatenströmen	Ja	Nein	Nein
Erfassen Sie Systemtelemetrietriemetriken in Echtzeit und reagieren Sie darauf	Ja	Ja	Nein
Konfigurieren Sie die Systemressourcenlimits für Komponentenprozesse	Ja	Nein	Nein
Unterbrechen Sie die Komponentenprozesse und setzen Sie sie fort	Ja	Nein	Nein

Feature	Linux	Windows	Greengrass Lite (Linux)
Integrieren Sie in die AWS IoT SiteWise Verwendung der AWS IoT SiteWise Komponenten	Ja	Ja	Nein
Veröffentlichen Sie Videostreams mit der Komponente Edge Connector für Kinesis Video Streams in Amazon Kinesis Video Streams	Ja	Nein	Nein

Entwicklung von Komponenten

Feature	Linux	Windows	Greengrass Lite (Linux)
Entwickeln Sie Komponenten lokal auf Kerngeräten	Ja	Ja	Ja
Interagieren Sie mit einem Core-Gerät über die AWS IoT Greengrass CLI	Ja	Ja	Nein
Interagieren Sie mit einem Core-Gerä t über die lokale Debug-Konsole	Ja	Ja	Nein

Feature	Linux	Windows	Greengrass Lite (Linux)
Verwenden Sie AWS IoT Device SDK for Python in benutzerdefinierten Komponenten	Ja	Ja	Ja
Verwenden Sie AWS IoT Device SDK für C++ in benutzerdefinierten Komponenten	Ja	Ja	Ja
Verwenden Sie AWS IoT Device SDK für Java in benutzerdefinierten Komponenten	Ja	Ja	Ja

Gerätezertifizierung

Feature	Linux	Windows	Greengrass Lite (Linux)
Verwenden Sie AWS IoT Device Tester für AWS IoT Greengrass V2 die Validierung von IoT-Geräten	Ja	Ja	Nein

Wählen Sie Ihre AWS IoT Greengrass Nucleus-Laufzeit

Ab Version 2.14.0 AWS IoT Greengrass bietet es zwei alternative Implementierungen seiner Gerätelaufzeit, eine ausführbare Datei, die als Nucleus bekannt ist. Trotz ihrer

Implementierungsunterschiede sind beide Laufzeiten mit dem AWS IoT Greengrass Service kompatibel APIs und ermöglichen es Ihnen, vom Greengrass SDK bereitgestellte Komponenten bereitzustellen AWS oder benutzerdefinierte Komponenten zu entwickeln. Es ist auch möglich, Geräte zu kombinieren und je nach Bedarf beide Arten von Nukleus innerhalb derselben Flotte zu verwenden.

Um jedoch die gewünschte Portabilität oder die spezifischen Vorteile der Speichereinsparung zu erreichen, müssen Sie unbedingt sicherstellen, dass der Nucleus, den Sie auf Ihren Greengrass-Geräten einsetzen, mit den Komponenten kompatibel ist, die Sie verwenden möchten, um die Entwicklung Ihrer AWS IoT Lösungen zu beschleunigen. Weitere Informationen zur Komponentenkompatibilität finden Sie unter Komponenten.

Letztlich hängt die Wahl zwischen den beiden Greengrass-Laufzeitoptionen von Ihrem spezifischen Anwendungsfall, den Geräteeinschränkungen, den Funktionsanforderungen und dem Betriebssystem ab.

Grüngraskern

AWS IoT Greengrass Nucleus ist die Runtime mit vollem Funktionsumfang, mit der Sie AWS IoT Greengrass auf einer Vielzahl von Geräten laufen können, darunter Gateways, Server und Edge-Geräte mit mehr Rechenressourcen. Erwägen Sie die Wahl von Greengrass Nucleus, wenn:

- Rechenressourcen: Ihr Gerät verfügt über ausreichend Rechenressourcen, wie z. B. mehr als 128 MB RAM und einen relativ leistungsstarken Prozessor (z. B. mehr als 1 GHz Uhr).
- Vollständige Betriebssystemunterstützung ist erforderlich: Greengrass Nucleus unterstützt die meisten Betriebssysteme (einschließlich der meisten Linux-Distributionen und Windows).
- Kompatibilität der Komponenten: Greengrass Nucleus bietet die umfassendste Kompatibilität mit bestehenden Komponenten, die vom AWS IoT Serviceteam und den Partnern veröffentlicht wurden.

Greengrass Nucleus Lite

AWS IoT Greengrass Nucleus Lite ist eine schlanke Open-Source-Runtime, mit der Sie AWS IoT Greengrass auf Geräten mit beschränkten Ressourcen arbeiten können. Dies kann für kostengünstige Einplatinencomputer mit umfangreichen Anwendungen wie Smart-Home-Hubs, intelligenten Energiezählern, intelligenten Fahrzeugen, Edge-KI und Robotik nützlich sein. Erwägen Sie, Greengrass Nucleus Lite zu wählen, wenn Ihre Geräte:

- Eingeschränkte Ressourcen: Ihr Gerät verfügt über begrenzte Ressourcen wie RAM-Speicher (512 MB oder weniger), Speicherplatz (FLASH) oder einen Prozessor mit geringer Leistung (weniger als 1). GHz
- Eingeschränkte Abhängigkeit: Die Softwareplattform Ihres Geräteanbieters unterstützt weder Java noch die spezifische JVM, die von Greengrass Nucleus benötigt wird.
- Betriebssystem: Auf Ihren Geräten läuft eine Linux-Distribution, die Systemd unterstützt (zum Beispiel: Ubuntu, Yocto).

Aktuelle Einschränkungen von Greengrass Nucleus Lite

Wie in AWS IoT Greengrass v2.14.0 enthalten, bietet Greengrass Nucleus Lite (v.2.0.0) Runtime einen Teil der Funktionen, die von Greengrass Nucleus (v2.14.0) verfügbar sind.

Der AWS IoT Greengrass IPC-Mechanismus (Interprocess Communication) ermöglicht es Komponenten, mit dem Greengrass-Kern zu kommunizieren. Die Lightweight-Version des Nucleus unterstützt die folgende Untergruppe:

Feature	Verfügbarkeit
SubscribeToTopic	Available (Verfügbar)
PublishToTopic	Available (Verfügbar)
PublishToIoTCore	Available (Verfügbar)
SubscribeToIoTCore	Available (Verfügbar)
UpdateState	Derzeit nicht verfügbar
SubscribeToComponentUpdates	Derzeit nicht verfügbar
DeferComponentUpdate	Derzeit nicht verfügbar
GetConfiguration	Available (Verfügbar)
UpdateConfiguration	Available (Verfügbar)
SubscribeToConfigurationUpdate	Available (Verfügbar)

Feature	Verfügbarkeit
SubscribeToValidateConfigurationUpdates	Derzeit nicht verfügbar.
SendConfigurationValidityReport	Derzeit nicht verfügbar.
GetSecretValue	Derzeit nicht verfügbar.
PutComponentMetric	Derzeit nicht verfügbar
GetComponentDetails	Derzeit nicht verfügbar
RestartComponent	Derzeit nicht verfügbar
StopComponent	Derzeit nicht verfügbar
CreateLocalDeployment	Available (Verfügbar)
CancelLocalDeployment	Derzeit nicht verfügbar
GetLocalDeploymentStatus	Derzeit nicht verfügbar
ListLocalDeployments	Derzeit nicht verfügbar
ListComponents	Derzeit nicht verfügbar
ValidateAuthorizationToken	Available (Verfügbar)
CreateDebugPassword	Derzeit nicht verfügbar
PauseComponent	Derzeit nicht verfügbar
ResumeComponent	Derzeit nicht verfügbar
GetThingShadow	Derzeit nicht verfügbar
UpdateThingShadow	Derzeit nicht verfügbar
DeleteThingShadow	Derzeit nicht verfügbar
ListNamedShadowsForThing	Derzeit nicht verfügbar

Feature	Verfügbarkeit
SubscribeToCertificateUpdates	Derzeit nicht verfügbar
VerifyClientDeviceIdentity	Derzeit nicht verfügbar
GetClientDeviceAuthToken	Derzeit nicht verfügbar
AuthorizeClientDeviceAction	Derzeit nicht verfügbar

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:

- AWS IoT Greengrass Nucleus Lite — AWS IoT Greengrass Nucleus-Versionen 2.14.0 und höher bieten AWS IoT Greengrass Nucleus Lite. Greengrass Nucleus Lite ist eine schlanke Open-Source-Runtime, mit der Sie Greengrass auf Geräten mit eingeschränkten Ressourcen ausführen können. Dies kann für kostengünstige Einplatinencomputer mit umfangreichen Anwendungen wie Smart-Home-Hubs, intelligenten Energiezählern, intelligenten Fahrzeugen, Edge-KI und Robotik nützlich sein. Weitere Informationen finden Sie unter Nucleus lite von Greengrass.
- Von Publisher unterstützte Komponenten — bietet jetzt auch vom Publisher unterstützte Komponenten. AWS IoT Greengrass Diese Komponenten werden von Drittanbietern entwickelt, angeboten und gewartet. Weitere Informationen finden Sie unter Von Publishern 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 Sie ein AWS IoT Greengrass Kerngerät in 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: Softwareupdate AWS IoT Greengrass Core v2.16.1 am 23. Dezember 2025
- Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.16.0 am 6. November 2025
- Veröffentlichung: AWS IoT Greengrass Core v2.15.1-Softwareupdate am 6. November 2025
- Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.15.0 am 3. Juli 2025
- Veröffentlichung: AWS IoT Greengrass Core v2.14.3-Softwareupdate am 11. April 2025
- Veröffentlichung: AWS IoT Greengrass Core v2.14.2-Softwareupdate am 27. März 2025
- Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.14.1 am 7. Februar 2025
- Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.14.0 am 16. Dezember 2024
- Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.13.0 am 26. August 2024
- 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 AWS IoT Greengrass Core v2.12.1 am 8. Dezember 2023
- Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.12.0 am 7. November 2023
- Veröffentlichung: AWS IoT Greengrass Core v2.11.3-Softwareupdate am 18. Oktober 2023
- Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.11.2 am 9. August 2023
- Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.11.1 am 21. Juli 2023
- Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.11.0 am 28. Juni 2023
- Veröffentlichung: AWS IoT Greengrass Core v2.10.3-Softwareupdate am 21. Juni 2023
- Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.10.2 am 5. Juni 2023
- Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.10.1 am 11. Mai 2023
- Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.10.0 am 9. Mai 2023
- Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.9.6 am 20. April 2023
- Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.9.5 am 30. März 2023

- Veröffentlichung: AWS IoT Greengrass Core v2.9.4 Softwareupdate am 24. Februar 2023
- Veröffentlichung: AWS IoT Greengrass Core v2.9.3 Softwareupdate am 01. Februar 2023
- Veröffentlichung: AWS IoT Greengrass Core v2.9.2 Softwareupdate am 22. Dezember 2022
- Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.9.1 am 18. November 2022
- Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.9.0 am 15. November 2022
- Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.8.1 am 13. Oktober 2022
- Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.8.0 am 7. Oktober 2022
- Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.7.0 am 28. Juli 2022
- Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.6.0 am 27. Juni 2022
- Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.5.6 am 31. Mai 2022
- Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.5.5 am 6. April 2022
- Veröffentlichung: AWS IoT Greengrass Core v2.5.4 Softwareupdate am 23. März 2022
- Veröffentlichung: AWS IoT Greengrass Core v2.5.3-Softwareupdate am 6. Januar 2022
- Veröffentlichung: AWS IoT Greengrass Core v2.5.2 Softwareupdate am 3. Dezember 2021
- Veröffentlichung: AWS IoT Greengrass Core v2.5.1 Softwareupdate am 23. November 2021
- Veröffentlichung: AWS IoT Greengrass Core v2.5.0-Softwareupdate am 12. November 2021
- Veröffentlichung: AWS IoT Greengrass Core v2.4.0-Softwareupdate am 3. August 2021
- Veröffentlichung: AWS IoT Greengrass Core v2.3.0-Softwareupdate am 29. Juni 2021
- Veröffentlichung: AWS IoT Greengrass Core v2.2.0-Softwareupdate am 18. Juni 2021
- Veröffentlichung: AWS IoT Greengrass Core v2.1.0-Softwareupdate am 26. April 2021
- Veröffentlichung: AWS IoT Greengrass Core v2.0.5 Softwareupdate am 09. März 2021
- Veröffentlichung: AWS IoT Greengrass Core v2.0.4 Softwareupdate am 04. Februar 2021

Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.16.1 am 23. Dezember 2025

Diese Version enthält Version 2.16.1 der Greengrass Nucleus-Komponente und Version 2.16.1 der Greengrass CLI-Komponente.

Veröffentlichungsdatum: 23. Dezember 2025

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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.16.1 von Greengrass Nucleus ist verfügbar. Fehlerkorrekturen und Verbesserungen • Fügt die Konfiguration für Wiederholungsintervalle für Anmeldein formationen nach Ausfällen des Token-Exchange-Dienstes hinzu.
Greengrass CLI	Version 2.16.1 der Greengrass CLI ist verfügbar. Version für die Version Greengrass Nucleus v2.16.1 aktualisiert.

Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.16.0 am 6. November 2025

Diese Version enthält Version 2.16.0 der Greengrass Nucleus-Komponente und Updates der bereitgestellten Komponenten. AWS

Veröffentlichungsdatum: 6. November 2025

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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.16.0 von Greengrass Nucleus ist verfügbar.

Komponente	Details
	Neue Features • Nucleus unterstützt jetzt cgroups V2. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem Nucleus veraltete Bereitstellungen nicht bereinigte, wenn eine Bereitstellung storniert wurde.
Greengrass Nucleus Lite	Version 2.3.0 des Greengrass Nucleus Lite ist verfügbar. Neue Features • Support für die Verwendung von TPM 2.0 für die IoT Core MQTT-Auto risierung. • Die apt-Beispielpakete unterstützen jetzt mehr Betriebssysteme: Ubuntu 22.04, Ubuntu 24.04, Debian 12 und Debian 13. • RestartComponent IPC wird jetzt unterstützt. Fehlerkorrekturen und Verbesserungen • Lokale Bereitstellungen benötigen keinen Internetzugang mehr. • GetConfiguration wurde aktualisiert, um dem Laufzeitverhalten von Greengrass Nucleus zu entsprechen. (Bahnbrechende Veränderung) • Allgemeine Fehlerbehebungen und Verbesserungen.
Schattenm anager	Version 2.3.12 des Shadow Managers ist verfügbar. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem Bereitstellungen blockiert wurden, wenn mehr als 1024 Geräteschatten konfiguriert wurden.
Protokoll manager	Version 2.3.11 des Log-Managers ist verfügbar. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem die Laufzeitkonfiguration von Log Manager mit veralteten Informationen zu hochgeladenen Protokolldateien unbegrenzt wuchs.

Komponente	Details
Systempro tokollweiterleitun g	Version 2.1.0 des System Log Forwarders ist verfügbar. Fehlerkorrekturen und Verbesserungen • Aktualisiert die Rezeptur der Komponenten, um den Greengrass-Kern ordnungsgemäß zu unterstützen. • Die Protokollausgabe für den Fall, dass keine Protokolle zum Hochladen vorhanden sind, wurde verbessert. • Allgemeine Fehlerbehebungen und Verbesserungen.

Veröffentlichung: AWS IoT Greengrass Core v2.15.1-Softwareupdate am 6. November 2025

Diese Version enthält Version 2.15.1 der Greengrass Nucleus-Komponente und Version 2.15.1 der Greengrass CLI-Komponente.

Veröffentlichungsdatum: 6. November 2025

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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern Version 2.15.1	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem Nucleus die Komponentenprozesse nach dem 30-sekündigen Shutdown-Timeout während Bootstrap-Bereitst ellungen nicht bereinigen konnte.
Greengrass CLI Version 2.15.1	Die Version wurde für die Version Greengrass Nucleus v2.15.1 aktualisiert.

Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.15.0 am 3. Juli 2025

Diese Version enthält Version 2.15.0 der Greengrass Nucleus-Komponente und Updates der bereitgestellten Komponenten. AWS

Veröffentlichungsdatum: 3. Juli 2025

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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.15.0 von Greengrass Nucleus ist verfügbar. Neue Features • Fügt eine Telemetriefunktion hinzu, um Hostsysteminformationen wie CPU- und Betriebssystemdetails einzubeziehen. Weitere Informationen finden Sie unter Telemetrie-Metriken. • Fügt die `deploymentConfigurationTimeSource` Konfiguration hinzu. Weitere Informationen finden Sie unter Greengrass Nucleus-Konfiguration. Fehlerkorrekturen und Verbesserungen • Nucleus priorisiert jetzt standardmäßig die Verwendung lokaler Versionen und Artefakte, wodurch die Konsistenz der Bereitstellung verbessert und externe Abhängigkeiten reduziert werden. Benutzer können dieses Verhalten außer Kraft setzen, indem sie im Bereitstellungsdokument alternative Einstellungen angeben.

Komponente	Details
	• Fügt ein neues Warnprotokoll hinzu, wenn ein bereitgestellter Windows-Benutzer nicht dem normalerweise von Windows akzeptierten Zeichensatz entspricht.
Greengrass CLI	Version 2.15.0 der Greengrass CLI ist verfügbar. Fehlerkorrekturen und Verbesserungen • Fügt einen Fix hinzu, sodass die Komponente nicht neu installiert wird, wenn sich die Authentifizierungskonfiguration ändert, sondern trotzdem neu gestartet wird. • Allgemeine Fehlerbehebungen und Verbesserungen.
Greengrass Nucleus Lite	Version 2.2.0 von Greengrass Nucleus Lite ist verfügbar. Neue Features • Fügt Unterstützung für Container-Image-Artefakte hinzu. URIs Fehlerkorrekturen und Verbesserungen • Allgemeine Fehlerbehebungen und Verbesserungen.
Protokoll manager	Version 2.3.10 des Log-Managers ist verfügbar. Neue Features • Fügt einen neuen Konfigurationsschlüssel (`updateToTlogIntervalSec`) hinzu, um die Häufigkeit zu steuern, mit der Details von Log-Upload-Ereignissen im lokalen Transaktionslog () gespeichert werden. `config.tlog` Fehlerkorrekturen und Verbesserungen • Verbessert den Protokollmanager, um den Cloudwatch-Client bei Socket-Verbindungsfehlern zu aktualisieren.

Komponente	Details
Geheimer Manager	Version 2.2.6 des Secret Managers ist verfügbar. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem der Secret Manager aufgrund eines langsamen oder nicht reagierenden Trusted Platform Module kein Geheimnis erhält.

Veröffentlichung: AWS IoT Greengrass Core v2.14.3-Softwareupdate am 11. April 2025

Diese Version enthält Version 2.14.3 der Greengrass Nucleus-Komponente und Updates der bereitgestellten Komponenten. AWS

Veröffentlichungsdatum: 11. April 2025

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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.14.3 von Greengrass Nucleus ist verfügbar. Fehlerkorrekturen und Verbesserungen • Ermöglicht den Neustart des Token-Austauschdienstes bei Änderungen der Portkonfiguration. • Behebt ein Problem, bei dem der Flottenstatusdienst keine Meldungen zur Änderung des Komponentenstatus für Lambdas sendet, wenn diese nicht ausgelöst wurden. • Behebt ein Problem, bei dem Komponenten nicht ordnungsgemäß heruntergefahren werden, wenn eine neue Komponentenversion bereitgestellt wird. • Behebt ein Problem, bei dem die Injection-Hooks integrierter Plugins zweimal ausgeführt werden, was zu zusätzlichen Lebenszyklusereign issen und doppelter Protokollierung führt. • Verbessert die Protokollierung des Komponentenlebenszyklus auf Windows-Geräten.
Greengrass CLI	Version 2.14.3 der Greengrass CLI ist verfügbar. Fehlerkorrekturen und Verbesserungen • Die Version wurde für die Version 2.14.3 von Greengrass Nucleus aktualisiert.
Geheimer Manager	Version 2.2.4 des Secret Managers ist verfügbar. Fehlerkorrekturen und Verbesserungen • Reduziert die Häufigkeit von Schreibvorgängen in den lokalen geheimen Speicher. Secret Manager schreibt jetzt nur noch in den lokalen Speicher, wenn Geheimnisse aktualisiert werden.

Komponente	Details
EMAX MQTT-Broker	Version 2.0.3 des EMQX MQTT-Brokers ist verfügbar. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem EMQX unter Windows nicht gestartet wird, wenn der Pfad Leerzeichen enthält.

Veröffentlichung: AWS IoT Greengrass Core v2.14.2-Softwareupdate am 27. März 2025

Diese Version enthält Version 2.14.2 der Greengrass Nucleus-Komponente und Updates der bereitgestellten Komponenten. AWS

Veröffentlichungsdatum: 27. März 2025

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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.14.2 von Greengrass Nucleus ist verfügbar. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem ein HTTP-Client nicht mit gegenseitiger Authentifizierung konfiguriert ist.
Greengrass CLI	Version 2.14.2 der Greengrass CLI ist verfügbar. Fehlerkorrekturen und Verbesserungen • Die Version wurde für die Version 2.14.2 von Greengrass Nucleus aktualisiert.

Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.14.1 am 7. Februar 2025

Diese Version enthält Version 2.14.1 der Greengrass Nucleus-Komponente und Updates der bereitgestellten Komponenten. AWS

Veröffentlichungsdatum: 7. Februar 2025

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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.14.1 von Greengrass Nucleus ist verfügbar. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem Komponenten bei Neuinstallationen von Greengrass nicht richtig gestoppt wurden.
Greengrass CLI	Version 2.14.1 der Greengrass CLI ist verfügbar. Fehlerkorrekturen und Verbesserungen • Die Version wurde für die Version 2.14.1 von Greengrass Nucleus aktualisiert.
Sicheres Tunneling	Version 1.1.1 der Secure Tunneling-Komponente ist verfügbar. Fehlerkorrekturen und Verbesserungen • Fügt eine Konfiguration zur Unterstützung von Greengrass Nucleus Lite hinzu.

Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.14.0 am 16. Dezember 2024

Diese Version enthält Version 2.14.0 der Greengrass Nucleus-Komponente und neue AWS IoT Greengrass Nucleus Lite-Updates. AWS IoT Greengrass Nucleus Lite ist eine neue Runtime, die für AWS IoT Greengrass Version 2 verfügbar ist. Sie bietet eine Alternative mit reduziertem Speicherbedarf. Dies ist eine gute Option für Geräte mit begrenzten Ressourcen. Es implementiert einen Teil der Nucleus-Funktionalität mit verbesserter Funktionskompatibilität, die für future Versionen geplant ist. Der Quellcode ist ab sofort auf Github verfügbar. Mit der Nucleus Lite-Runtime können Sie:

- Stellen Sie Komponenten auf Greengrass-Kerngeräten bereit. Verwenden Sie dasselbe Rezeptformat, obwohl einige erweiterte Funktionen möglicherweise noch nicht verfügbar sind.
- Anwendungen, die als Greengrass-Komponenten bereitgestellt werden, können das Gerät verwenden SDKs , um auf den unterstützten Greengrass-IPC zuzugreifen APIs, z. B.: AWS IoT Core MQTT-Zugriff, lokaler Pub/Sub- und Greengrass-Konfigurationszugriff. Eine Liste der unterstützten IPCs finden Sie in der Kompatibilitätstabelle. APIs
- Einige AWS verwaltete Komponenten wurden für die Unterstützung von Nucleus Lite aktualisiert. Eine Liste vorhandener kompatibler Komponenten finden Sie unter den AWS bereitgestellten Komponenten.

Neue Funktionen:

- Benötigt weniger Arbeitsspeicher und Festplattenspeicher (weniger als 5 MB RAM und weniger als 5 MB Speicher).
- Die Komponenten sind in den Service Manager des Hostsystems integriert (systemd für derzeit unterstützte Linux-Plattformen).

Dinge, auf die Sie achten sollten:

- AWS IoT Greengrass Bei Nucleus Lite-Rezepten wird Groß- und Kleinschreibung beachtet. Stellen Sie sicher, dass das richtige (Schlüssel-) Gehäuse wie in der https://docs.aws.amazon.com/greengrass/v2/developerguide/component-recipe-reference.html Rezeptreferenz verwendet wird.

- Die Nucleus Lite-Runtime unterstützt Dinggruppen-Bereitstellungen und unterstützt noch nicht den Bereitstellungszieltyp (einzelnes) Core-Gerät. Verwenden Sie für die Bereitstellung auf einem einzelnen Greengrass-Gerät eine Dinggruppe, in der nur dieses eine Gerät enthalten ist.
- Die Nucleus Lite-Runtime verwendet begrenzte Speicherressourcen. Funktionen, die je nach Nutzung in der klassischen Runtime skaliert werden, können aufgrund der Überschreitung der auf Lite verfügbaren Ressourcen fehlschlagen. Dies beinhaltet eine aktuelle Beschränkung auf maximal 50 MQTT-Abonnements gleichzeitig und Höchstgrenzen für die Größe und Bereitstellung von Rezeptdateien. Einige dieser Grenzwerte können zur Kompilierzeit konfiguriert werden, wenn Sie die Lite-Runtime selbst kompilieren.
- Die Nucleus Lite-Runtime ist nicht im Lieferumfang von Java enthalten. Um Komponenten verwenden zu können, die Java erfordern, muss Java auf dem System bereits installiert sein, oder es kann eine Komponente zur Installation von Java verwendet werden.
- Wir empfehlen, die Nucleus Lite-Runtime aus dem Quellcode zu kompilieren und Ihren eigenen Build zu verwenden, der auf Ihr System zugeschnitten ist. Für Yocto-Systeme ist eine Ebene verfügbar, um die Nucleus Lite-Runtime in Ihr System-Image zu integrieren.
- Derzeit geht Nucleus Lite von einem Linux-System aus, das Systemd verwendet, oder von einem Container-Image, das Systemd verwendet.
- Sie können Docker-Container zwar mit Rezeptskripten verwalten, von Greengrass verwaltete Container-Artefakte sind jedoch noch nicht verfügbar.
- Die Nucleus Lite-Runtime bietet noch keine Unterstützung für Schlüssel, die in einem PKCS11 Modul gespeichert sind. Wenn Ihr Anwendungsfall Schlüssel erfordert, die auf einem sicheren Element gespeichert sind, kann die klassische Runtime diesen Anwendungsfall derzeit unterstützen. Stellen Sie sicher, dass die Produktionsgeräte die vollständige Festplattenverschlüsselung verwenden, um zu verhindern, dass Ihre Geräteanmeldeinformationen durchsickern.

Neben der Einführung von Nucleus Lite veröffentlichen wir auch Nucleus v2.14.0. Dieses Update bringt erhebliche Verbesserungen für den bestehenden Greengrass-Kern.

Wichtigste Funktionen und Verbesserungen:

- Die neue Unterstützung von Dual-Stack-Endpunkten ermöglicht die IPv6 Netzwerkkommunikation.
- Verbesserte Widerstandsfähigkeit gegen Nucleus-Neustartfehler und Verzeichnisbeschädigung.
- Speicherlecks beim Schließen von PubSub IPC-Abonnements wurden behoben.

Veröffentlichungsdatum: 16. Dezember 2024

Öffentliche Komponenten-Updates

In der folgenden Tabelle sind die AWS bereitgestellten Komponenten aufgeführt, 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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Greengrass Nucleus Lite	Version 2.0.0 von Greengrass Nucleus Lite ist verfügbar. Neue Features • Benötigt weniger Arbeitsspeicher und Festplattenspeicher (weniger als 5 MB RAM und weniger als 5 MB Speicher). • Die Komponenten sind in den Service Manager des Hostsystems integriert (systemd für derzeit unterstützte Linux-Plattformen).
Grüngraskern	Version 2.14.0 von Greengrass Nucleus ist verfügbar.

Komponente	Details
	Neue Features • Die neue Unterstützung von Dual-Stack-Endpunkten ermöglicht die Netzwerkkommunikation. IPv6 • Verbesserte Widerstandsfähigkeit gegen Nucleus-Neustartfehler und Verzeichnisbeschädigung. Fehlerkorrekturen und Verbesserungen • Speicherlecks beim Schließen von PubSub IPC-Abonnements wurden behoben. • Behebt den Ausführungslebenszyklus der Komponente, bei der sie aufgrund eines Start-Timeouts in den Status ERRORED übergeht, wenn die Skipif-Bedingung erfüllt ist. • Behebt ein Problem, bei dem das Core-Gerät keine Verbindung herstelle n kann, AWS IoT Core wenn die TLS-Richtlinie auf _1_3_2022_10 gesetzt ist. TLS13
Greengrass CLI	Version 2.14.0 der Greengrass CLI ist verfügbar. Fehlerkorrekturen und Verbesserungen • Überprüfen Sie den Zielparameter für die Bereitstellung im CLI-Befehl.
Stream-Manager	Version 2.14.0 des Stream-Managers ist verfügbar. Neue Features • Fügt einen neuen Konfigurationsschlüssel für das Start-Timeout hinzu. Der Standardwert ist 120 Sekunden. • Fügen Sie Rezeptunterstützungen für Greengrass Nucleus Lite hinzu.
MQTT 5-Broker (EMQX)	Version 2.0.2 des MQTT 5-Brokers (EMQX) ist verfügbar. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem EMQX gestartet wird, bevor die Komponente für die Client-Geräteauthentifizierung bereit ist.

Komponente	Details
Lambda-Laufzeitkomponente	Version 2.0.9 der Komponente Lambda Runtimes ist verfügbar. Fehlerkorrekturen und Verbesserungen • Behebt eine Syntaxwarnung mit Python 3.12
Lambda-Manager-Komponente	Version 2.3.5 der Lambda-Manager-Komponente ist verfügbar. Fehlerkorrekturen und Verbesserungen • Verbessert die Leistung, indem epoll anstelle von no verwendet wird, sofern verfügbar
Secret Manager-Komponente	Version 2.2.2 der Secret Manager-Komponente ist verfügbar. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem der Secret Manager die mit teilweisen ARNS konfigurierten Geheimnisse nicht herunterlädt.
Komponente für sicheres Tunneling	Version 1.1.0 der Komponente Secure Tunneling ist verfügbar. Neue Features • Rezeptstützen für Greengrass Nucleus Lite hinzufügen
CloudWatch Komponente für Metriken	Version 1.1.0 der CloudWatch Metrikkomponente ist verfügbar. Neue Features • Rezeptstützen für Greengrass Nucleus Lite hinzufügen

Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.13.0 am 26. August 2024

Diese Version stellt Version 2.13.0 der Greengrass Nucleus-Komponente bereit.

Veröffentlichungsdatum: 26. August 2024

Öffentliche Komponenten-Updates

In der folgenden Tabelle sind die AWS bereitgestellten Komponenten aufgeführt, 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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.13.0 von Greengrass Nucleus ist verfügbar. Neue Features • Support den FIPS-Endpunkt in Nucleus. Weitere Informationen finden Sie unter FIPS-Endpunkte. Fehlerkorrekturen und Verbesserungen • Bereitstellungsverbesserungen abbrechen — Bereitstellungen können jetzt abgebrochen werden, während eine neue Konfiguration zusammengeführt wird und auf den Start der Dienste gewartet wird.
Stream-Manager	Version 2.1.13 der Stream Manager-Komponente ist verfügbar. Fehlerkorrekturen und Verbesserungen • Support Sie den FIPS-Endpunkt in AWS IoT SiteWise

Komponente	Details
Geheimer Manager	Version 2.2.0 der Secret Manager-Komponente ist verfügbar. Neue Features • Integriert die Unterstützung für die regelmäßige Aktualisierung konfiguri erter Geheimnisse über einen neuen Komponentenkonfigurationssc hlüssel. • Fügt Unterstützung für einen neuen Anforderungsparameter in der GetSecretValueIPC-Anfrage hinzu, um die Geheimnisse pro Anfrage zu aktualisieren
IP-Detektor	Version 2.2.0 der IP-Detektorkomponente ist verfügbar. Neue Features • Fügt Unterstützung für IPv6 hinzu. Sie können es jetzt IPv6 für lokales Messaging verwenden.
Authentifizierung auf dem Client-Gerät	Version 2.5.1 der Client-Geräteauthentifizierung ist verfügbar. Fehlerkorrekturen und Verbesserungen • Allgemeine Fehler und Korrekturen. • Unterstützt den FIPS-Endpunkt.
Lokale Debug-Konsole	Version 2.4.3 der lokalen Debug-Konsole ist verfügbar. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem STREAM_MANAGER_EXPORTER_MAX _BANDWIDTH fälschlicherweise in Mpbs statt in Bytes/Sekunde angezeigt wurde.

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 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 Sie die 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 des Jetson Nano, beim Start zu einem Absturz führt.
Greengrass CLI	Version 2.12.6 der Greengrass CLI ist verfügbar.

Komponente	Details
	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 Sie die 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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.12.4 von Greengrass Nucleus ist verfügbar. 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 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 Sie die 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.

Komponente	Details
	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.
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 Nucleus-Komponente und Updates der bereitgestellten Komponenten. AWS

Veröffentlichungsdatum: 15. Februar 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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.12.2 von Greengrass Nucleus ist verfügbar.

Komponente	Details
	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem alte Logs nicht richtig bereinigt wurden. • Allgemeine Fehlerbehebungen und Verbesserungen.
Schattenmanager	Version 2.3.6 der Shadow Manager-Komponente ist verfügbar. 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.
Lambda-Launcher	Version 2.0.13 der Lambda-Launcher-Komponente ist verfügbar. Fehlerkorrekturen und Verbesserungen Allgemeine Fehlerbehebungen und Verbesserungen.
Festplatten-Spooler	Version 1.0.3 der Disk Spooler-Komponente ist verfügbar. Fehlerkorrekturen und Verbesserungen Verbessert die Leistung durch die Wiederverwendung von Datenbankverbindungen.

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

Diese Version enthält Version 2.12.1 der Greengrass Nucleus-Komponente und Updates der bereitgestellten Komponenten. AWS

Veröffentlichungsdatum: 8. Dezember 2023

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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.12.1 von Greengrass Nucleus ist verfügbar. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem der Nucleus MQTT-Abonnements für Bereitstellungsthemen duplizieren kann, was zu zusätzlicher Protokollierung und MQTT-Veröffentlichung führt.
Authentifizierung auf dem Client-Gerät	Version 2.4.5 der Komponente für die Authentifizierung auf Clientgeräten ist verfügbar. Neue Features Integriert die Unterstützung für Platzhalterpräfixe zur Auswahl von Dingnamen mit dem Parameter. `selectionRule`

Komponente	Details
	Fehlerkorrekturen und Verbesserungen Behebt ein Problem, bei dem Zertifikate in bestimmten Fällen nicht mit neuen Verbindungsinformationen aktualisiert werden.
Festplatten-Spooler	Version 1.0.2 der Disk Spooler-Komponente ist verfügbar. Fehlerkorrekturen und Verbesserungen Behebt ein Problem, bei dem das Feld im MQTT-Nachrichtenformat in bestimmten Fällen nicht dauerhaft angezeigt wird.
MQTT-Brücke	Version 2.3.1 der Disk Spooler-Komponente ist verfügbar. Fehlerkorrekturen und Verbesserungen Behebt ein Problem, bei dem der lokale MQTT-Client in eine Trennschleife gerät.
Stream-Manager	Version 2.1.12 der Stream Manager-Komponente ist verfügbar. Fehlerkorrekturen und Verbesserungen Aktualisiert die Reihenfolge, in der Anmeldeinformationen verwendet werden, sodass Greengrass-Anmeldeinformationen für AWS Servicean fragen bevorzugt werden.

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

Diese Version enthält Version 2.12.0 der Greengrass Nucleus-Komponente und Updates der bereitgestellten Komponenten. AWS

Veröffentlichungsdatum: 7. November 2023

Höhepunkte der Veröffentlichung

- Bootstrap on Rollback — stellt AWS IoT Greengrass jetzt einen Greengrass-Nukleus-Konfigurationsparameter bereit, der aufgerufen wird. `BootstrapOnRollback` Mit dieser Funktion können Sie die Bootstrap-Lebenszyklusschritte als Teil einer Rollback-Bereitstellung ausführen.

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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.12.0 von Greengrass Nucleus ist verfügbar. Neue Features • Ermöglicht es Ihnen, die Bootstrap-Lebenszyklusschritte als Teil einer Rollback-Bereitstellung auszuführen.

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

Diese Version stellt Version 2.11.3 der Greengrass Nucleus-Komponente bereit.

Veröffentlichungsdatum: 18. Oktober 2023

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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.11.3 von Greengrass Nucleus ist verfügbar.

Komponente	Details
	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.
Lambda-Manager	Version 2.3.1 der Lambda-Manager-Komponente ist verfügbar. Fehlerkorrekturen und Verbesserungen • Passt die Protokollebenen für bestimmte Fehler an.
Lokale Debug-Konsole	Version 2.4.0 der Lambda-Manager-Komponente ist verfügbar. Neue Features • Fügt die Stream Manager-Debugging-Konsole hinzu.
Protokollmanager	Version 2.3.6 der Log Manager-Komponente ist verfügbar. Fehlerkorrekturen und Verbesserungen • Passt die Protokollebenen für bestimmte Fehler an.
Schattenmanager	Version 2.3.4 der Shadow Manager-Komponente ist verfügbar. Fehlerkorrekturen und Verbesserungen • Integriert die Unterstützung für Dokumente mit Nullstatus und leeren Schattenzuständen.

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

Diese Version stellt Version 2.11.2 der Greengrass Nucleus-Komponente bereit.

Veröffentlichungsdatum: 9. August 2023

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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.11.2 von Greengrass Nucleus ist verfügbar. 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.

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

Diese Version stellt Version 2.11.1 der Greengrass Nucleus-Komponente bereit.

Veröffentlichungsdatum: 21. Juli 2023

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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.11.1 von Greengrass Nucleus ist verfügbar.

Komponente	Details
	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.
Lambda-Manager	Version 2.2.11 des Lambda-Managers ist verfügbar. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem die LegacySubscriptionRouter Konfiguration nicht aktualisiert wird, wenn sich die Lambda-Konfiguration ändert.

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

Diese Version stellt Version 2.11.0 der Greengrass Nucleus-Komponente bereit.

Veröffentlichungsdatum: 28. Juni 2023

Höhepunkte der Veröffentlichung

- Persistenter Festplatten-Spooler — bietet AWS IoT Greengrass jetzt eine persistente Spooler-Implementierung für Nachrichten, auf die von Greengrass-Kerngeräten gespoolt wird. AWS IoT Core Diese Komponente speichert diese ausgehenden Nachrichten auf der Festplatte. Weitere Informationen finden Sie unter Festplattenspooler.
- Verbesserungen bei der lokalen Bereitstellung — Sie können jetzt lokale Bereitstellungen stornieren, Richtlinien für die Behandlung fehlgeschlagener Bereitstellungen festlegen und sich den detaillierten Status der Bereitstellung anzeigen lassen.
- Verbesserungen der Protokollierungsgeschwindigkeit — Die Geschwindigkeit beim Hochladen von Protokollen für die Log-Manager-Komponente wurde verbessert.

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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.11.0 von Greengrass Nucleus ist verfügbar. 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.
Greengrass CLI	Version 2.11.0 der Greengrass CLI ist verfügbar. Neue Features • Ermöglicht es Ihnen, eine lokale Bereitstellung abzubrechen.

Komponente	Details
	• Ermöglicht die Konfiguration einer Fehlerbehandlungsrichtlinie für eine lokale Bereitstellung. • Verbessert die detaillierte Berichterstattung zum Bereitstellungsstatus.
Festplatten-Spooler	Version 1.0.0 der Disk Spooler-Komponente ist verfügbar. • Die Festplatten-Spooler-Komponente ermöglicht die persistente Speicherung von Nachrichten, die von Greengrass-Core-Geräten an gesendet werden. AWS IoT Core
Protokoll manager	Version 2.3.5 der Log Manager-Komponente ist verfügbar. Verbesserungen Verbessert die Geschwindigkeit beim Hochladen von Protokollen.

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

Diese Version stellt Version 2.10.3 der Greengrass Nucleus-Komponente bereit.

Veröffentlichungsdatum: 21. Juni 2023

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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.10.3 von Greengrass Nucleus ist verfügbar. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem Greengrass keine Bereitstellungsben achrichtigungen abonniert, wenn der PKCS #11 -Anbieter verwendet wird.

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

Diese Version stellt Version 2.10.2 der Greengrass Nucleus-Komponente bereit.

Veröffentlichungsdatum: 5. Juni 2023

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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.10.2 von Greengrass Nucleus ist verfügbar. 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.
Authentifizierung auf dem Client-Gerät	Version 2.4.2 der Komponente für die Authentifizierung auf Clientgeräten ist verfügbar. Neue Features Fügt eine neue `startupTimeoutSeconds` Konfigurationsoption hinzu.
Lambda-Manager	Version 2.2.9 der Lambda-Manager-Komponente ist verfügbar.

Komponente	Details
	Fehlerkorrekturen und Verbesserungen Behebt ein Problem, bei dem die Portnummer aufgrund einer schiefen Uhr beschädigt ist.
Protokoll manager	Version 2.3.4 der Log Manager-Komponente ist verfügbar. Fehlerkorrekturen und Verbesserungen • Fügt Unterstützung für das Setzen des `periodicUploadIntervalSec` Parameters auf Bruchwerte hinzu. Das Minimum ist 1 Mikrosekunde. • Behebt ein Problem, bei dem der Protokollmanager die CloudWatch `putLogEvents` Grenzwerte nicht einhält.
MQTT 3.1-Broker (Moquette)	Version 2.3.3 der Komponente MQTT 3.1 Broker (Moquette) ist verfügbar. Neue Features Fügt eine neue Konfigurationsoption hinzu. `startupTimeoutSeconds`
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. Fehlerkorrekturen und Verbesserungen Behebt ein Problem, bei dem der Stream-Manager die Proxykonfiguration nicht korrekt lesen kann.

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

Diese Version stellt Version 2.10.1 der Greengrass Nucleus-Komponente bereit.

Veröffentlichungsdatum: 11. Mai 2023

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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.10.1 von Greengrass Nucleus ist verfügbar. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, das bei bestimmten ARMv8 Prozessoren, einschlie ßlich des Jetson Nano, zu einem Absturz beim Start führen konnte. • Greengrass schließt den Standard einer Komponente nicht mehr, dadurch wird das Verhalten auf das Verhalten vor 2.10.0 zurückgesetzt
Stream-Manager	Version 2.1.6 des neuen Stream-Managers ist verfügbar.

Komponente	Details
	Fehlerkorrekturen und Verbesserungen Behebt ein Problem, das bei bestimmten ARMv8 Prozessoren, einschließlich des Jetson Nano, zu einem Absturz beim Start führen konnte.

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

Diese Version enthält Version 2.10.0 der Greengrass Nucleus-Komponente und Updates der bereitgestellten Komponenten. AWS

Veröffentlichungsdatum: 9. Mai 2023

Höhepunkte der Veröffentlichung

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

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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.10.0 von Greengrass Nucleus ist verfügbar. Neue Features • Fügt `interpolateComponentConfiguration` Unterstützung für den leeren regulären Ausdruck hinzu. Greengrass interpoliert jetzt vom Root-Konfigurationsobjekt aus. • Fügt Unterstützung für hinzu. MQTT5 • Fügt einen Mechanismus hinzu, mit dem Plugin-Komponenten schnell und ohne Scannen geladen werden können. • 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`

Komponente	Details
	• 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.
Schattenm anager	Version 2.3.2 des neuen Shadow Managers ist verfügbar. Fehlerkorrekturen und Verbesserungen Behebt ein Problem, bei dem der Shadow Manager in den `BROKEN` Status wechselt, wenn die lokale Shadow-Datenbank beschädigt ist.

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

Diese Version stellt Version 2.9.6 der Greengrass Nucleus-Komponente bereit.

Veröffentlichungsdatum: 20. April 2023

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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.9.6 von Greengrass Nucleus ist verfügbar. 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.

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

Diese Version stellt Version 2.9.5 der Greengrass Nucleus-Komponente bereit.

Veröffentlichungsdatum: 30. März 2023

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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.9.5 von Greengrass Nucleus ist verfügbar. 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.

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

Diese Version stellt Version 2.9.4 der Greengrass Nucleus-Komponente bereit.

Veröffentlichungsdatum: 24. Februar 2023

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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.9.4 von Greengrass Nucleus ist verfügbar.

Komponente	Details
	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.

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

Diese Version stellt Version 2.9.3 der Greengrass Nucleus-Komponente bereit.

Veröffentlichungsdatum: 01. Februar 2023

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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.9.3 von Greengrass Nucleus ist verfügbar. Fehlerkorrekturen und Verbesserungen • Stellt sicher, dass MQTT-Clients 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.

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

Diese Version stellt Version 2.9.2 der Greengrass Nucleus-Komponente bereit.

Veröffentlichungsdatum: 22. Dezember 2022

Einzelheiten zur Veröffentlichung

- Öffentliche Komponenten-Updates

Öffentliche Komponenten-Updates

In der folgenden Tabelle sind die AWS bereitgestellten Komponenten aufgeführt, 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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.9.2 von Greengrass Nucleus ist verfügbar. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem die Konfiguration `interpolateComponentConfiguration` nicht für eine laufende Bereitstellung gilt. • Verwendet OSHI, um alle untergeordneten Prozesse aufzulisten.

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

Diese Version enthält Version 2.9.1 der Greengrass Nucleus-Komponente und Updates der bereitgestellten Komponenten. AWS

Veröffentlichungsdatum: 18. November 2022

Höhepunkte der Veröffentlichung

- Protokollmanager — Der Protokollmanager 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

Einzelheiten zur Veröffentlichung

- Öffentliche Komponenten-Updates

Öffentliche Komponenten-Updates

In der folgenden Tabelle sind die AWS bereitgestellten Komponenten aufgeführt, 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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.9.1 von Greengrass Nucleus ist verfügbar.

Komponente	Details
	Fehlerkorrekturen und Verbesserungen • Fügt einen Fix hinzu, bei dem Greengrass neu startet, wenn eine Bereitstellung eine Plugin-Komponente entfernt.
Protokoll manager	Version 2.3.0 des neuen Log-Managers ist verfügbar. **Note** Wir empfehlen, dass Sie auf Greengrass Nucleus 2.9.1 aktualisieren, wenn Sie auf Log Manager 2.3.0 aktualisieren. Neue Features • Reduziert Protokollverzögerungen, indem aktive Protokolldateien verarbeitet und direkt hochgeladen werden, anstatt darauf zu warten, dass neue Dateien rotiert werden. Fehlerkorrekturen und Verbesserungen • Verbessert die Unterstützung der Protokollrotation beim Rotieren von Dateien mit einem eindeutigen Namen. • Zusätzliche kleinere Korrekturen und Verbesserungen.

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

Diese Version enthält Version 2.9.0 der Greengrass Nucleus-Komponente und Updates der bereitgestellten Komponenten. AWS

Veröffentlichungsdatum: 15. November 2022

Höhepunkte der Veröffentlichung

- 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 zu beheben. Mit jeder Unterbereitstellung kann eine andere Konfiguration einer erfolglosen Bereitstellung auf einer kleineren Teilmenge von Geräten getestet werden. Weitere Informationen finden Sie unter Unterbereitstellungen erstellen.

Einzelheiten zur Veröffentlichung

- Öffentliche Komponenten-Updates

Öffentliche Komponenten-Updates

In der folgenden Tabelle sind die AWS bereitgestellten Komponenten aufgeführt, 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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.9.0 von Greengrass Nucleus ist verfügbar.

Komponente	Details
	Neue Features • Integriert die Möglichkeit, Unterbereitstellungen zu erstellen, die Bereitstellungen mit einer kleineren Teilmenge von Geräten wiederhol en. 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.
Authentifizierung auf dem Client-Gerät	Version 2.3.0 der Komponente für die Authentifizierung auf Clientgeräten ist verfügbar. Neue Features • Integriert die Unterstützung für die Offline-Authentifizierung von Client-Geräten. Mit dieser Funktion können Client-Geräte weiterhin eine Verbindung zum Core-Gerät herstellen, wenn das Core-Gerät nicht mit dem Internet verbunden ist. • Integriert die Unterstützung für vom Kunden bereitgestellte Zertifizi erungsstellen (CA). Ihr Kerngerät verwendet eine vom Kunden bereitges tellte CA als Stammzertifikat, um MQTT-Brokerzertifikate zu generieren.
MQTT 5-Broker (EMQX)	Version 1.2.0 der Komponente ist verfügbar. MQTT 5-Broker (EMAX) Neue Features Fügt Unterstützung für Zertifikatsketten hinzu.
Moquette MQTT-Broker	Version 2.3.0 der neuen Moquette MQTT-Broker-Komponente ist verfügbar. Neue Features Integriert die Unterstützung für Zertifikatsketten.

Komponente	Details
Geheimer Manager	Version 2.1.4 des neuen Secret Managers ist verfügbar. Fehlerkorrekturen und Verbesserungen Behebt ein Problem, bei dem zwischengespeicherte Geheimnisse entfernt wurden, wenn Secret Manager bereitgestellt und Greengrass Nucleus neu gestartet wurde.
Stream-Manager	Version 2.1.2 des neuen Stream-Managers ist verfügbar. Fehlerkorrekturen und Verbesserungen Behebt ein Problem unter Windows-Betriebssystemen, die eine andere Sprache als Englisch verwenden.

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

Diese Version stellt Version 2.8.1 der Greengrass Nucleus-Komponente bereit.

Veröffentlichungsdatum: 13. Oktober 2022

> **Note**
>
> Wenn Sie Greengrass Nucleus Version 2.8.0 verwenden, empfehlen wir dringend, auf Greengrass Nucleus Version 2.8.1 zu aktualisieren.

Einzelheiten zur Veröffentlichung

- Öffentliche Komponenten-Updates

Öffentliche Komponenten-Updates

In der folgenden Tabelle sind die AWS bereitgestellten Komponenten aufgeführt, 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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.8.1 von Greengrass Nucleus ist verfügbar. 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.

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

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

Veröffentlichungsdatum: 7. Oktober 2022

Höhepunkte der Veröffentlichung

- Fehlercodes bei der Bereitstellung — Der Greengrass Nucleus meldet jetzt eine Antwort auf den Bereitstellungsstatus, die detaillierte Fehlercodes enthält, wenn eine Komponentenbereitstellung nicht abgeschlossen werden kann. Weitere Informationen finden Sie unter Detaillierte Fehlercodes für die Bereitstellung.
- Fehlerstatus von Komponenten — Der Greengrass-Kern meldet jetzt eine Antwort auf den Komponentenstatus, die detaillierte Fehlerstatusinformationen enthält, wenn eine Komponente in den `BROKEN` Status oder wechselt. `ERRORED` Weitere Informationen finden Sie unter Detaillierte Statuscodes für Komponenten.

Einzelheiten zur Veröffentlichung

- Öffentliche Komponenten-Updates

Öffentliche Komponenten-Updates

In der folgenden Tabelle sind die AWS bereitgestellten Komponenten aufgeführt, 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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.8.0 von Greengrass Nucleus ist verfügbar. 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 Fehlercodes für die Bereitstellung. • 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 Statuscodes für Komponenten. • 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.
MQTT 5-Broker (EMQX)	Version 1.1.0 der Komponente ist verfügbar. MQTT 5-Broker (EMAX)

Komponente	Details
	Neue Features • Integriert die Unterstützung für EMQX-Konfigurationen, einschließlich Broker-Optionen und Plug-ins. Fehlerkorrekturen und Verbesserungen • Aktualisiert EMQX auf Version 4.4.9.

Veröffentlichung: Softwareupdate AWS IoT Greengrass Core v2.7.0 am 28. Juli 2022

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

Veröffentlichungsdatum: 28. Juli 2022

Höhepunkte der Veröffentlichung

- 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 Kerngeräte hochladen. Weitere Informationen finden Sie unter Erfassung von Telemetriedaten zur Systemintegrität von AWS IoT Greengrass Kerngeräten.
- Benutzerdefinierte Zertifizierungsstelle (CA) — Client-Zertifikate, die von einer benutzerdefinierten Zertifizierungsstelle signiert wurden, bei der die CA nicht registriert ist AWS IoT, werden jetzt unterstützt. Weitere Informationen finden Sie unter Verwenden Sie ein Gerätezertifikat, das von einer privaten Zertifizierungsstelle signiert wurde.

Einzelheiten zur Veröffentlichung

- Öffentliche Komponenten-Updates

Öffentliche Komponenten-Updates

In der folgenden Tabelle sind die AWS bereitgestellten Komponenten aufgeführt, 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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.7.0 von Greengrass Nucleus ist verfügbar. 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.

Komponente	Details
	• 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 bei der Festlegung der SubscribeToComponentUpdates Bereitstellungs-ID bei Ereignissen, bei denen der Nucleus 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, sodass automatisch Telemetriedaten an Amazon EventBridge gesendet werden. Weitere Informationen finden Sie unter Erfassung von Telemetriedaten zur Systemintegrität von AWS IoT Greengrass Kerngeräten. Für diese Funktion ist Version 2.7.0 oder höher der Greengrass Nucleus-Komponente erforderlich. • Die Version wurde für die Version 2.7.0 von Greengrass Nucleus 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. 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.

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

Diese Version enthält Version 2.6.0 der Greengrass Nucleus-Komponente, neue von ihnen AWS bereitgestellte Komponenten und Updates für von -bereitgestellte Komponenten. AWS

Veröffentlichungsdatum: 27. Juni 2022

Höhepunkte der Veröffentlichung

- Platzhalter in Themen zum lokalen Publizieren/Abonnieren — Sie können jetzt MQTT-Platzhalter verwenden, wenn Sie Themen zum lokalen Publizieren/Abonnieren abonnieren. Weitere Informationen erhalten Sie unter Lokale Nachrichten veröffentlichen/abonnieren und SubscribeToTopic.
- Unterstützung für Client-Geräteshadow — Sie können jetzt mit Client-Geräteschatten in benutzerdefinierten Komponenten interagieren und Client-Geräteschatten mit synchronisieren. AWS IoT Core Weitere Informationen finden Sie unter Interagieren Sie mit Schatten auf Client-Geräten und synchronisieren Sie sie.
- Lokale MQTT 5-Unterstützung für Client-Geräte — Sie können jetzt den EMQX MQTT 5-Broker einsetzen, um MQTT 5-Funktionen bei der Kommunikation zwischen Client-Geräten und einem Core-Gerät zu verwenden. Weitere Informationen erhalten Sie unter MQTT 5-Broker (EMAX) und Client-Geräte mit Core-Geräten Connect.
- Rezeptvariablen in Komponentenkonfigurationen — Sie können jetzt spezifische 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 erhalten Sie unter Rezeptvariablen und Verwenden Sie Rezeptvariablen bei Merge-Updates.
- Platzhalter in IPC-Autorisierungsrichtlinien — Sie können jetzt * Platzhalter verwenden, um einer beliebigen Kombination von Zeichen in IPC-Autorisierungsrichtlinien (Interprocess Communication) zu entsprechen. 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 zur Verwaltung lokaler Bereitstellungen und Komponenten — Sie können jetzt benutzerdefinierte Komponenten entwickeln, die lokale Bereitstellungen verwalten und Komponentendetails anzeigen. Weitere Informationen finden Sie unter IPC: Lokale Bereitstellungen und Komponenten verwalten.

- IPC-Operationen zur Authentifizierung und Autorisierung von Client-Geräten — Sie können diese Operationen jetzt verwenden, um eine benutzerdefinierte lokale Broker-Komponente zu erstellen. Weitere Informationen finden Sie unter IPC: Client-Geräte authentifizieren und autorisieren.

Einzelheiten zur Veröffentlichung

- Öffentliche Komponenten-Updates

Öffentliche Komponenten-Updates

In der folgenden Tabelle sind die AWS bereitgestellten Komponenten aufgeführt, 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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.6.0 von Greengrass Nucleus ist verfügbar. Neue Features • Fügt Unterstützung für MQTT-Platzhalter hinzu, wenn Sie lokale Themen zum Publizieren/Abonnieren abonnieren. Weitere Informationen

Komponente	Details
	erhalten Sie unter Lokale Nachrichten veröffentlichen/abonnieren und SubscribeToTopic. • Fügt Unterstützung für Rezeptvariablen in Komponentenkonfigu rationen 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 interpolateComponentConfigu rationKonfigurationsoption auf`true`. Weitere Informationen erhalten Sie unter Rezeptvariablen und Verwenden Sie Rezeptvariablen bei Merge-Updates. • Fügt vollständige Unterstützung für den * Platzhalter in Autorisie rungsrichtlinien 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 Autorisierungsrich tlinien. • Integriert die Unterstützung für benutzerdefinierte Komponenten zum Aufrufen von IPC-Operationen, 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 • Behebt ein Problem, bei dem abhängige Komponenten in bestimmte n Szenarien nicht reagierten, wenn ihre festen Abhängigkeiten neu gestartet wurden oder ihren Status änderten. • Verbessert die Fehlermeldungen, die das Kerngerät an den AWS IoT Greengrass Cloud-Dienst meldet, wenn eine Bereitstellung fehlschlägt.

Komponente	Details
	• 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.
MQTT 5-Broker (EMQX)	Version 1.0.0 der neuen EMQX MQTT 5-Broker-Komponente 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 mithilfe der MQTT 5-Funktionen mit einem Kerngerät zu kommunizieren.
Schattenmanager	Version 2.2.0 der Shadow Manager-Komponente ist verfügbar. Neue Features • Fügt Unterstützung für den lokalen Shadow-Service über den lokalen publish/subscribe interface. You can now communicate with the local publish/subscribe Message Broker zu Shadow-MQTT-Themen hinzu, um Schatten auf dem Kerngerät abzurufen, zu aktualisieren und zu löschen. Mit dieser Funktion können Sie Client-Geräte mit dem lokalen Shadow-Dienst 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.

Komponente	Details
Authentifizierung auf dem Client-Gerät	Version 2.2.0 der Komponente für die Authentifizierung auf Clientgeräten ist verfügbar. 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.
MQTT-Brücke	Version 2.2.0 der MQTT-Bridge-Komponente ist verfügbar. Neue Features • Fügt Unterstützung für MQTT-Themen-Platzhalter (#und+) hinzu, 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.

Komponente	Details
Greengrass CLI	Version 2.6.0 der Greengrass CLI ist verfügbar. 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.

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

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

Veröffentlichungsdatum: 31. Mai 2022

Einzelheiten zur Veröffentlichung

- Öffentliche Komponenten-Updates

Öffentliche Komponenten-Updates

In der folgenden Tabelle sind die AWS bereitgestellten Komponenten aufgeführt, 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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.5.6 von Greengrass Nucleus ist verfügbar. 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.
Protokoll manager	Version 2.2.4 der Log Manager-Komponente ist verfügbar. Fehlerkorrekturen und Verbesserungen • Verbessert die Stabilität beim Umgang mit ungültigen Konfigurationen. • Zusätzliche kleinere Korrekturen und Verbesserungen.

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

Diese Version stellt Version 2.5.5 der Greengrass Nucleus-Komponente bereit.

Veröffentlichungsdatum: 6. April 2022

Einzelheiten zur Veröffentlichung

- Öffentliche Komponenten-Updates

Öffentliche Komponenten-Updates

In der folgenden Tabelle sind die AWS bereitgestellten Komponenten aufgeführt, 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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.5.5 von Greengrass Nucleus ist verfügbar.

Komponente	Details
	Neue Features • Fügt die `GG_ROOT_CA_PATH` Umgebungsvariable für Komponenten hinzu, sodass Sie in benutzerdefinierten Komponenten auf das Stammzertifikat der Zertifizierungsstelle (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 Ressourcenbereitstellung 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.

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

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

Veröffentlichungsdatum: 23. März 2022

Einzelheiten zur Veröffentlichung

- Öffentliche Komponenten-Updates

Öffentliche Komponenten-Updates

In der folgenden Tabelle sind die AWS bereitgestellten Komponenten aufgeführt, 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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.5.4 von Greengrass Nucleus ist verfügbar. Fehlerkorrekturen und Verbesserungen • Allgemeine Fehlerbehebungen und Verbesserungen.
Lambda-Launcher	Version 2.0.10 der Lambda-Launcher-Komponente ist verfügbar. Fehlerkorrekturen und Verbesserungen • Allgemeine Fehlerbehebungen und Verbesserungen.

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

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

Veröffentlichungsdatum: 6. Januar 2022

Höhepunkte der Veröffentlichung

- Integration der Hardwaresicherheit — Sie können die AWS IoT Greengrass Core-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.

Einzelheiten zur Veröffentlichung

- Öffentliche Komponenten-Updates

Öffentliche Komponenten-Updates

In der folgenden Tabelle sind die AWS bereitgestellten Komponenten aufgeführt, 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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.5.3 von Greengrass Nucleus ist verfügbar.

Komponente	Details
	Neue Features • Fügt Unterstützung für die Integration von Hardwaresicherheit 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.
PKCS #11 - Anbieter	Version 2.0.0 der PKCS #11 -Provider-Komponente 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: AWS IoT Greengrass Core v2.5.2 Softwareupdate am 3. Dezember 2021

Diese Version stellt Version 2.5.2 der Greengrass Nucleus-Komponente bereit.

Veröffentlichungsdatum: 3. Dezember 2021

Einzelheiten zur Veröffentlichung

- Öffentliche Komponenten-Updates

Öffentliche Komponenten-Updates

In der folgenden Tabelle sind die AWS bereitgestellten Komponenten aufgeführt, 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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.5.2 von Greengrass Nucleus ist verfügbar. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem der Windows-Dienst nach dem Greengras s Nucleus-Update nicht erneut gestartet werden kann, nachdem Sie ihn gestoppt oder das Gerät neu gestartet haben.
AWS IoT Device Defender	Version 3.0.1 der AWS IoT Device DefenderKomponente ist verfügbar. Diese Version der AWS IoT Device Defender Komponente erwartet andere Konfigurationsparameter als Version 2.x. Wenn Sie für Version 2.x eine nicht standardmäßige Konfiguration verwenden und ein Upgrade von Version 2.x auf Version 3.x durchführen möchten, müssen Sie die Konfiguration der Komponente aktualisieren. Weitere Informationen finden Sie unter Komponent enkonfiguration.AWS IoT Device Defender Neue Features • Fügt Unterstützung für Kerngeräte hinzu, auf denen Windows ausgeführ t wird.

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

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

Diese Version stellt Version 2.5.1 der Greengrass Nucleus-Komponente bereit.

Veröffentlichungsdatum: 23. November 2021

Einzelheiten zur Veröffentlichung

- Öffentliche Komponenten-Updates

Öffentliche Komponenten-Updates

In der folgenden Tabelle sind die AWS bereitgestellten Komponenten aufgeführt, 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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.5.1 von Greengrass Nucleus ist 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 AWS IoT Greengrass Komponenten auf Geräten bereitstellen. • 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.

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

Diese Version enthält Version 2.5.0 der Greengrass Nucleus-Komponente, neue von ihnen AWS bereitgestellte Komponenten und Updates für von -bereitgestellte Komponenten. AWS

Veröffentlichungsdatum: 12. November 2021

Höhepunkte der Veröffentlichung

- Windows-Geräteunterstützung — Sie können die AWS IoT Greengrass Core-Software jetzt auf Geräten mit Windows-Betriebssystemen ausführen. Weitere Informationen finden Sie unter Kompatibilität der Greengrass-Funktionen.
- Neues Verhalten beim Entfernen von Dinggruppen — Sie können jetzt ein Kerngerät aus einer Dinggruppe entfernen, um die Komponenten dieser Dinggruppe bei der nächsten Bereitstellung auf diesem Gerät zu entfernen.

  > **Important**
  >
  > 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.

- Unterstützung für Hardwaresicherheit — Sie können die AWS IoT Greengrass Core-Software jetzt so konfigurieren, dass sie ein Hardware-Sicherheitsmodul (HSM) verwendet, sodass Sie den privaten Schlüssel und das Zertifikat des Geräts sicher speichern können. Weitere Informationen finden Sie unter Integration von Hardware-Sicherheit.
- HTTPS-Proxyunterstützung — Sie können die AWS IoT Greengrass Core-Software jetzt so konfigurieren, dass sie eine Verbindung über HTTPS-Proxys herstellt. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy.

Einzelheiten zur Veröffentlichung

- Updates zur Plattformunterstützung
- Öffentliche Komponenten-Updates

Updates zur Plattformunterstützung

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

Plattform	Details
	• Windows 10 • Windows Server 2019 Weitere Informationen finden Sie unter Kompatibilität der Greengrass-Funktionen.

Öffentliche Komponenten-Updates

In der folgenden Tabelle sind die AWS bereitgestellten Komponenten aufgeführt, 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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.5.0 von Greengrass Nucleus ist verfügbar. Neue Features • Fügt Unterstützung für Kerngeräte hinzu, auf denen Windows ausgeführt wird.

Komponente	Details
	• 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 Kerngerät s 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. Mit diesem Parameter können Sie den Standardbenutzer angeben, 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. • Behebt die IPC-Autorisierung für On-Demand-Lambda-Funktionsk omponenten.

Komponente	Details
	• Verbessert die Protokollmeldungen 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.
Greengrass CLI	Version 2.5.0 der Greengrass CLI ist verfügbar. Neue Features • Fügt Unterstützung für Kerngeräte hinzu, auf denen Windows ausgeführ t wird. • Fügt den neuen `AuthorizedWindowsGroups` Konfigurationspara meter 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.

Komponente	Details
CloudWatch Metriken	Version 3.0.0 der CloudWatchMetrikkomponente 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 Version 2.x auf Version 3.x durchführen möchten, müssen Sie die Konfiguration der Komponente aktualisieren. Weitere Informationen finden Sie unter Konfiguration der Metrikkomponenten. CloudWatch Neue Features • Integriert die Unterstützung für Kerngeräte, auf denen Windows ausgeführt wird. • Ändert den Komponententyp von einer Lambda-Komponente in eine generische Komponente. Diese Komponente ist jetzt nicht mehr von der älteren Abonnement-Router-Komponente abhängig, 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. Fehlerkorrekturen und Verbesserungen Integriert die Unterstützung für doppelte Zeitstempel in Eingabedaten.

Komponente	Details
Lambda-Manager	Version 2.2.0 der Lambda-Manager-Komponente ist verfügbar. 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

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

Diese Version enthält Version 2.4.0 der Greengrass Nucleus-Komponente, neue von ihnen AWS bereitgestellte Komponenten und Updates der von -bereitgestellten Komponenten. AWS

Veröffentlichungsdatum: 3. August 2021

Höhepunkte der Veröffentlichung

- Systemressourcenlimits — Die Greengrass Nucleus-Komponente unterstützt jetzt Systemressourcenlimits. 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.
- Komponenten pausieren/fortsetzen — Der Greengrass Nucleus unterstützt jetzt das Anhalten und Wiederaufnehmen von Komponenten. Sie können die IPC-Bibliothek (Interprocess Communication) verwenden, um benutzerdefinierte Komponenten zu entwickeln, die die Prozesse anderer Komponenten anhalten und wieder aufnehmen. Weitere Informationen erhalten Sie unter PauseComponent und ResumeComponent.
- Mit AWS IoT Fleet Provisioning installieren — Verwenden Sie das neue AWS IoT Fleet Provisioning-Plugin, um die AWS IoT Greengrass Core-Software auf Geräten zu installieren, die eine Verbindung herstellen, um benötigte Ressourcen bereitzustellen. AWS IoT AWS Geräte verwenden für die Bereitstellung ein Antragszertifikat. Sie können das Antragszertifikat bei der

Herstellung auf Geräten einbetten, sodass jedes Gerät die Bereitstellung durchführen 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 Provisioning-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 individuellen Anwendungsfall einzurichten. Weitere Informationen finden Sie unter Installieren Sie die AWS IoT Greengrass Core-Software mit benutzerdefinierter Ressourcenbereitstellung.

Einzelheiten zur Veröffentlichung

- Öffentliche Komponenten-Updates

Öffentliche Komponenten-Updates

In der folgenden Tabelle sind die AWS bereitgestellten Komponenten aufgeführt, 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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.4.0 von Greengrass Nucleus ist verfügbar. 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 erhalten 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.

Komponente	Details
	• Zusätzliche kleinere Korrekturen und Verbesserungen. Weitere Informationen finden Sie in den Veröffentlichungen unter 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 der einzelnen Komponenten auf dem Kerngerät verwenden können. Weitere Informationen finden Sie unter Konfigurieren Sie die Systemressourcenlimits für Komponenten und dem Befehl deployment create.
AWS IoT Bereitstellung der Flotte nach Schadensfall	Das Plug-in „ AWS IoT Fleet Provisioning by Claim“ 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 her, AWS IoT um die erforderlichen AWS Ressourcen bereitzustellen und Gerätezertifikate herunterzuladen, um sie für den regulären Betrieb zu verwenden.

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

Diese Version stellt Version 2.3.0 der Greengrass Nucleus-Komponente bereit.

Veröffentlichungsdatum: 29. Juni 2021

Höhepunkte der Veröffentlichung

- Unterstützung umfangreicher Konfigurationen — Die Nucleus-Komponente von Greengrass unterstützt jetzt Bereitstellungsdokumente bis zu 10 MB. Sie können jetzt größere Konfigurationsupdates für Greengrass-Komponenten bereitstellen.

> **Note**
>
> 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.

Einzelheiten zur Veröffentlichung

- Öffentliche Komponenten-Updates

Öffentliche Komponenten-Updates

In der folgenden Tabelle sind die AWS bereitgestellten Komponenten aufgeführt, 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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.3.0 von Greengrass Nucleus ist verfügbar. Neue Features • Integriert die Unterstützung für Bereitstellungskonfigurationsdokumente bis zu 10 MB, bis zu 7 KB (für Bereitstellungen, die auf Dinge abzielen) oder 31 KB (für Bereitstellungen, die auf Objektgruppen 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.

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

Diese Version enthält Version 2.2.0 der Greengrass Nucleus-Komponente, neue von ihnen AWS bereitgestellte Komponenten und Updates für von -bereitgestellte Komponenten. AWS

Veröffentlichungsdatum: 18. Juni 2021

Höhepunkte der Veröffentlichung

- Unterstützung für Client-Geräte — Die neu AWS bereitgestellten Client-Gerätekomponenten ermöglichen es Ihnen, Client-Geräte mithilfe von Cloud Discovery mit Ihren Kerngeräten zu verbinden. In Greengrass-Komponenten können Sie Client-Geräte mit Client-Geräten synchronisieren AWS IoT Core und mit diesen interagieren. Weitere Informationen finden Sie unter Interagieren Sie mit lokalen IoT-Geräten.
- Lokaler Shadow-Service — Die neue Shadow-Manager-Komponente aktiviert den lokalen Shadow-Service auf Ihren Kerngeräten. Sie können diesen Shadow-Service verwenden, um offline mit lokalen Shadows zu interagieren, indem Sie die Greengrass Interprocess Communication (IPC) -Bibliotheken in der verwenden. AWS IoT Device SDK Sie können die Shadow-Manager-Komponente auch verwenden, um lokale Shadow-Zustände mit zu synchronisieren. AWS IoT Core Weitere Informationen finden Sie unter Interagiere mit Geräteschatten.

Einzelheiten zur Veröffentlichung

- Öffentliche Komponenten-Updates

Öffentliche Komponenten-Updates

In der folgenden Tabelle sind die AWS bereitgestellten Komponenten aufgeführt, 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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.2.0 von Greengrass Nucleus ist verfügbar. 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.
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 lokales Schattenmanagement mithilfe von IPC hinzu. • Fügt Unterstützung für die Shadow-Synchronisierung mit AWS IoT Core hinzu.
Authentifizierung auf dem Client-Gerät	Version 2.0.0 der neuen Authentifizierungskomponente für Client-Geräte ist verfügbar. Neue Features • Integriert die Unterstützung für Greengrass-Client-Geräte, bei denen es sich um lokale IoT-Geräte handelt, die über MQTT eine Verbindung zu einem Kerngerät herstellen. • Integriert die Unterstützung für die Authentifizierung und Autorisierung von Client-Geräten und deren MQTT-Aktionen.
Moquette MQTT-Broker	Version 2.0.0 der neuen Moquette MQTT-Broker-Komponente ist verfügbar.

Komponente	Details
	Neue Features • Integriert die Unterstützung für einen lokalen Moquette MQTT-Broker, der die Kommunikation mit Client-Geräten abwickelt.
MQTT-Brücke	Version 2.0.0 der neuen MQTT-Bridge-Komponente ist verfügbar. Neue Features • Integriert die Unterstützung für die Weiterleitung von Nachrichten zwischen dem lokalen MQTT-Broker, dem lokalen Publish/Subscribe-Broker von Greengrass und dem MQTT-Broker. AWS IoT Core
IP-Detektor	Version 2.0.0 der neuen IP-Detektorkomponente ist verfügbar. Neue Features • Fügt Unterstützung hinzu, um die lokalen MQTT-Broker-Endpunkte eines Kerngeräts an den AWS IoT Greengrass Cloud-Dienst zu melden, damit Client-Geräte eine Verbindung herstellen können.
Protokoll manager	Version 2.1.1 der Log Manager-Komponente ist verfügbar. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem die Konfiguration des Systemprotokolls in bestimmten Fällen nicht aktualisiert wurde.
DLR-Objek terkennung	Version 2.1.2 der DLR-Objekterkennung ist verfügbar. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem mit der Bildskalierung, das zu ungenauen Begrenzungsrahmen in den Beispielinferenzergebnissen der DLR-Objek terkennung führte.

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

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

Diese Version enthält Version 2.1.0 der Greengrass Nucleus-Komponente und von Updates AWS bereitgestellte Komponenten.

Veröffentlichungsdatum: 26. April 2021

Höhepunkte der Veröffentlichung

- Integration von Docker Hub und Amazon Elastic Container Registry (Amazon ECR) — Mit der neuen Docker Application Manager-Komponente können Sie öffentliche oder private Images von Amazon ECR herunterladen. Sie können diese Komponente auch verwenden, um öffentliche Images von Docker Hub und herunterzuladen. AWS 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 für die Ausführung AWS IoT Greengrass in einem Docker-Container verwenden, der Amazon Linux 2 als Basisbetriebssystem verwendet. Sie können das AWS IoT Greengrass Dockerfile auch verwenden, um Ihr eigenes Greengrass-Image zu erstellen. Weitere Informationen finden Sie unter Führen Sie die AWS IoT Greengrass Core-Software in einem Docker-Container aus.
- Support für zusätzliche Frameworks und Plattformen für maschinelles Lernen — Sie können Beispielkomponenten für maschinelles Lernen bereitstellen, die vortrainierte Modelle verwenden, um die Probenbildklassifizierung und Objekterkennung mit TensorFlow Lite 2.5.0 und DLR 1.6.0 durchzuführen. Diese Version erweitert auch die Unterstützung von Beispielen für maschinelles Lernen für Armv8 () -Geräte. AArch64 Weitere Informationen finden Sie unter Durchführen von Machine Learning-Inferenzen.

Einzelheiten zur Veröffentlichung

- Updates zur Plattformunterstützung
- Öffentliche Komponenten-Updates

Updates zur Plattformunterstützung

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 auf einer anderen Plattformarchitektur läuft. AWS IoT Greengrass Docker-Image AWS IoT Greengrass stellt ein vorgefertigtes Docker-Image bereit, 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 Führen Sie die AWS IoT Greengrass Core-Software in einem Docker-Container aus.

Öffentliche Komponenten-Updates

In der folgenden Tabelle sind die AWS bereitgestellten Komponenten aufgeführt, 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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.1.0 von Greengrass Nucleus 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 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.

Komponente	Details
	• Verbessert die Auflösung von Komponentenabhängigkeiten, sodass eine Bereitstellung fehlschlägt, die eine zirkuläre Abhängigkeit enthält. • Behebt ein Problem, das die erneute Bereitstellung einer Plugin-Komponente verhinderte, 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.
Docker-Anwendungsmanager	Version 2.0.0 der neuen Docker-Anwendungsmanager-Komponente ist verfügbar. Neue Features • Verwaltet Anmeldeinformationen zum Herunterladen von Bildern aus privaten Repositorys in Amazon ECR. • Lädt öffentliche Bilder von Amazon ECR, Docker Hub und herunter. AWS Marketplace
Lambda-Launcher	Version 2.0.4 der Lambda-Launcher-Komponente ist verfügbar. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem die Komponente nicht korrekt `AddGroupOwner` an den Lambda-Funktionscontainer übergeben wird.

Komponente	Details
Legacy-Abonnement-Router	Version 2.1.0 der älteren Abonnement-Router-Komponente ist verfügbar. Fehlerkorrekturen und Verbesserungen • Integriert die Unterstützung für die Angabe von Komponentennamen anstelle von ARNs für `source` und`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. Fehlerkorrekturen und Verbesserungen • Sie können Flashbar-Meldungen im Konfigurationseditor verwerfen.
Protokollmanager	Version 2.1.0 der Log Manager-Komponente ist verfügbar. Fehlerkorrekturen und Verbesserungen • Verwenden Sie Standardwerte für `logFileDirectoryPath` Greengrass-Komponenten`logFileRegex` , die auf Standardausgabe (stdout) und Standardfehler (stderr) drucken, und die für diese funktionieren. • Leiten Sie den Datenverkehr korrekt über einen konfigurierten Netzwerk-Proxy weiter, wenn Sie Protokolle in Logs hochladen. CloudWatch • Behandelt Doppelpunkte (`:`) in Protokolldatenstromnamen korrekt. CloudWatch Log-Log-Stream-Namen unterstützen keine Doppelpunkte. • Vereinfachen Sie die Namen von Protokolldatenströmen, indem Sie die Namen der Dinggruppen aus dem Protokollstream entfernen. • Entfernen Sie eine Fehlerprotokollmeldung, die bei normalem Verhalten ausgegeben wird.

Komponente	Details
DLR-Bildklassifizierung	Version 2.1.1 der DLR-Komponente zur Bildklassifizierung ist verfügbar. Neue Features • Verwenden Sie Deep Learning Runtime v1.6.0. • Unterstützung für die Klassifizierung von Beispielbildern auf Armv8 (AArch64) -Plattformen 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.

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 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.

Komponente	Details
Modellspeicher für die DLR-Bildklassifizierung	Version 2.1.1 der DLR-Komponente zum Speichern von Bildklassifizierungsmodellen ist verfügbar. Neue Features • Fügen Sie ein Beispiel für ein ResNet -50-Bildklassifizierungsmodell für Armv8 () -Plattformen hinzu. AArch64 Dies erweitert die Unterstützung für maschinelles Lernen für Greengrass-Kerngeräte, auf denen NVIDIA Jetson ausgeführt wird, wie z. B. den Jetson Nano.
Modellspeicher für DLR-Objekterkennung	Version 2.1.1 der DLR-Komponente für den Modellspeicher zur Objekterkennung ist verfügbar. Neue Features • Fügen Sie ein Beispiel für ein YOLOv3 Objekterkennungsmodell für Armv8 () AArch64 -Plattformen hinzu. Dies erweitert die Unterstützung für maschinelles Lernen für Greengrass-Kerngeräte, auf denen NVIDIA Jetson ausgeführt wird, wie z. B. den Jetson Nano.
DLR-Installationsprogramm	Version 1.6.1 der DLR-Komponente ist verfügbar. Neue Features • Installieren Sie Deep Learning Runtime v1.6.0 und die zugehörigen Abhängigkeiten. • Unterstützung für die Installation von DLR auf Armv8 () AArch64 -Plattformen hinzugefügt. Dies erweitert die Unterstützung für maschinelles 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 Konfigurationsänderungen anzuwenden. • Zusätzliche kleinere Bugfixes und Verbesserungen.

Komponente	Details
TensorFlow Lite-Bildklassifizierung	Version 2.1.0 der neuen TensorFlow Lite-Komponente zur Bildklassifizierung ist verfügbar. Neue Features • Unterstützung für die Inferenz zur Klassifizierung von Beispielbildern mithilfe von TensorFlow Lite hinzugefügt.
TensorFlow Lite-Objekterkennung	Version 2.1.0 der neuen TensorFlow Lite-Komponente zur Objekterkennung ist verfügbar. Neue Features • Unterstützung für Beispiel-Inferenzen zur Objekterkennung mit TensorFlow Lite hinzugefügt.
TensorFlow Lite-Modellspeicher zur Bildklassifizierung	Version 2.1.0 der neuen TensorFlow Lite-Komponente „Image Classification Model Store“ ist verfügbar. Neue Features • Stellen Sie mit Lite ein vortrainiertes MobileNet quantisiertes v1-Modell für die Inferenz zur Klassifizierung von Probenbildern bereit. TensorFlow
TensorFlow Lite-Modellspeicher zur Objekterkennung	Version 2.1.0 der neuen TensorFlow Lite-Komponente für den Objektspeicher zur Objekterkennung ist verfügbar. Neue Features • Stellen Sie ein vortrainiertes MobileNet SSD-Modell (Single Shot Detection) bereit, das anhand des COCO-Datensatzes trainiert wurde, um mithilfe von Lite anhand eines Beispiels eine Inferenz zur Objekterkennung zu erhalten. TensorFlow
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: AWS IoT Greengrass Core v2.0.5 Softwareupdate am 09. März 2021

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 China Regionen.

Veröffentlichungsdatum: 09. März 2021

Öffentliche Komponenten-Updates

In der folgenden Tabelle sind die AWS bereitgestellten Komponenten aufgeführt, 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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.0.5 von Greengrass Nucleus ist verfügbar. Fehlerkorrekturen und Verbesserungen • Leitet den Datenverkehr beim Herunterladen von AWS bereitgestellten Komponenten korrekt über einen konfigurierten Netzwerk-Proxy weiter.

Komponente	Details
	• Verwenden Sie den richtigen Endpunkt der Greengrass-Datenebene in AWS China Regionen.

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

Diese Version stellt Version 2.0.4 der Greengrass Nucleus-Komponente bereit. Sie enthält den neuen `greengrassDataPlanePort` 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`

Veröffentlichungsdatum: 04. Februar 2021

Öffentliche Komponenten-Updates

In der folgenden Tabelle sind die AWS bereitgestellten Komponenten aufgeführt, 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 Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Details
Grüngraskern	Version 2.0.4 von Greengrass Nucleus ist verfügbar. 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 vorhanden ist. 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.

Von AWS IoT Greengrass Version 1 migrieren

AWS IoT Greengrass Version 2 ist eine Hauptversion der AWS IoT Greengrass Core-Software und der Konsole. APIs AWS IoT Greengrass V2 führt mehrere Verbesserungen ein AWS IoT Greengrass V1, wie z. B. modulare Anwendungen, Bereitstellungen auf großen Geräteflotten und Unterstützung für zusätzliche Plattformen.

> **Note**
>
> Hinweis zum Ende des Supports: Am 1. Oktober 2026 AWS endet der Support für. AWS IoT Greengrass Version 1 Nach dem 1. Oktober 2026 können Sie nicht mehr auf die AWS IoT Greengrass V1 Konsole oder AWS IoT Greengrass V1 die Ressourcen zugreifen.

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

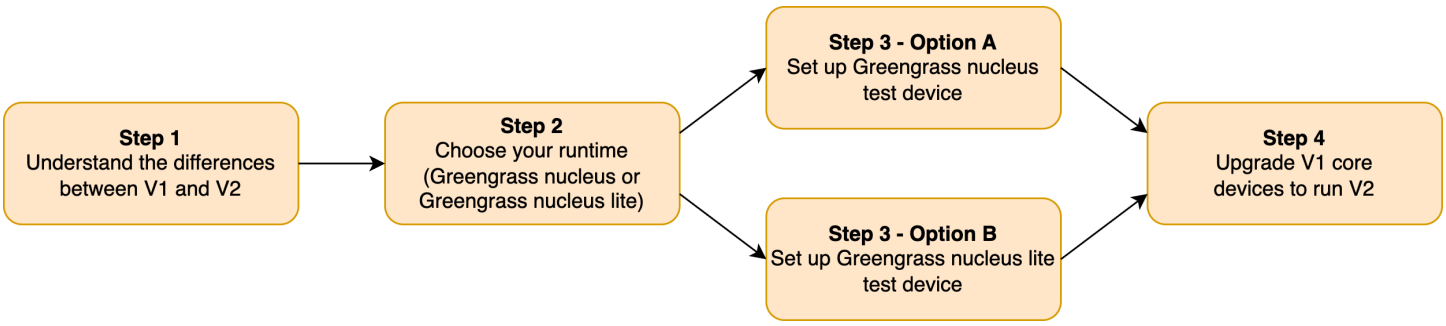
Überblick über die Migration

Auf einer höheren Ebene können Sie das folgende Verfahren verwenden, um Kerngeräte von auf AWS IoT Greengrass V1 zu aktualisieren AWS IoT Greengrass V2.

Vor der Migration wählen Sie zwischen zwei Runtime-Optionen:

- Greengrass Nucleus (geringerer Migrationsaufwand, volle Funktionsunterstützung)
- Greengrass Nucleus Lite (höherer Migrationsaufwand, konzipiert für Geräte mit begrenzten Ressourcen).

Das genaue Verfahren, das Sie befolgen, hängt von Ihren Geräteressourcen, den erforderlichen Funktionen und den spezifischen Umgebungsanforderungen ab.



1. Verstehen Sie die Unterschiede zwischen V1 und V2

   AWS IoT Greengrass V2 führt neue grundlegende Konzepte für Geräteflotten und einsatzfähige Software ein, und V2 vereinfacht mehrere Konzepte aus Version 1.

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

2. Wählen Sie Ihre Laufzeit (Greengrass Nucleus oder Greengrass Nucleus Lite)

   Entscheiden Sie sich je nach Geräteressourcen und Funktionsanforderungen zwischen Greengrass Nucleus oder Greengrass Nucleus Lite:

   - Greengrass-Kernpfad: Geringerer Migrationsaufwand. Lambda-Funktionen können als Lambda-Komponenten mit minimalen Codeänderungen importiert werden. Unterstützt V1-Funktionen (lokaler Shadow-Service, Client-Geräte, Konnektoren).
   - Greengrass Nucleus Lite Path: Höherer Migrationsaufwand. Lambda-Funktionen müssen in generische Komponenten umgewandelt werden, sodass Codeänderungen erforderlich sind, um das AWS IoT Device SDK AWS IoT Greengrass V2/Component SDK anstelle des AWS IoT Greengrass Core SDK zu verwenden. Unterstützt keine lokalen Shadow-Dienste, Client-Geräte oder Konnektoren.

3. Richten Sie ein neues Gerät ein, um V1-Anwendungen auf V2 zu testen

   Um das Risiko für Ihre Geräte in der Produktion zu minimieren, erstellen Sie ein neues Gerät, um Ihre V1-Anwendungen auf V2 zu testen. Wählen Sie den Einrichtungsleitfaden auf der Grundlage Ihrer Laufzeitauswahl aus:

   - Option A — Greengrass Nucleus Runtime: Richten Sie ein neues Gerät ein, um V1-Anwendungen auf V2 zu testen. Importieren Sie Lambda-Funktionen als Lambda-Komponenten mit minimalen Codeänderungen.
   - Option B — Greengrass Nucleus Lite-Laufzeit: Richten Sie ein neues Gerät ein, um V1-Anwendungen auf V2 zu testen (Greengrass Nucleus Lite). Konvertieren Sie Lambda-Funktionen in generische Komponenten mit AWS IoT Device SDK.

4. Aktualisieren Sie V1-Kerngeräte, um V2 auszuführen

   Aktualisieren Sie nach dem Testen auf einem neuen Gerät Ihre vorhandenen V1-Core-Geräte, sodass sie die AWS IoT Greengrass Core-Software v2.x und AWS IoT Greengrass V2 die

zugehörigen Komponenten ausführen können. Um eine Geräteflotte von V1 auf 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 einsatzfähige 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 AWS IoT Greengrass V1 definieren Lambda-Funktionen die Software, die auf Kerngeräten 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 containerisierten Lambda-Laufzeitumgebung ausführt, definieren Sie Container-Parameter wie Speicherlimits.	Bei Komponenten handelt AWS IoT Greengrass V2 es sich um Softwaremodule, die auf Kerngeräten ausgeführt werden. • Jede Komponente hat ein Rezept, das die Metadaten, Parameter, Abhängigkeiten und Skripts der Komponente definiert, die in 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 Komponente auf einem Kerngerät

Konzept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
		bereitstellen, lädt das Kerngerät das Komponent enrezept und die Artefakte herunter, um die Komponent e auszuführen. Sie können Ihre V1-Lambda -Funktionen als Komponent en importieren, die in einer Lambda-Laufzeitumgebung in ausgeführt werden. AWS IoT Greengrass V2 Wenn Sie die Lambda-Funktion importier en, geben Sie die Abonnemen ts, lokalen Ressourcen und Containerparameter für die Funktion an. Weitere Informati onen finden Sie unter Schritt 2: AWS IoT Greengrass V2 Komponenten für die Migration AWS IoT Greengrass V1 von Anwendungen erstellen und bereitstellen. Weitere Informationen zum Erstellen benutzerdefinierte r Komponenten finden Sie unterAWS IoT Greengrass Komponenten entwickeln.

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

Konzept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
		entwickeln. Weitere Informationen finden Sie unter AWS IoT Greengrass Komponenten erstellen.

Konzept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
AWS IoT Greengrass Kernsoftware	Bei AWS IoT Greengrass V1 der AWS IoT Greengras s Core-Software handelt es sich um ein einziges Paket, das die Software und all ihre Funktionen enthält. Das Edge-Gerät, auf dem Sie die AWS IoT Greengrass Core-Soft ware installieren, wird als Greengrass-Core bezeichnet.	Die AWS IoT Greengrass Core-Software ist modular aufgebaut, sodass Sie auswählen können, was installiert werden soll, um den Speicherbedarf zu kontrolli eren. AWS IoT Greengrass V2 • Die Greengrass Nucleus-Komponente ist die mindestens erforderliche Installation der AWS IoT Greengrass Core-Software. Das Edge-Gerät, auf dem Sie den Nucleus installieren, wird als Greengrass-Core-Gerät bezeichnet. • Der Nucleus kümmert sich um die Bereitstellung, Orchestrierung und das Lebenszyklusmanagement anderer Komponenten auf dem Core-Gerät. • Funktionen wie Stream Manager, Secret Manager und Log Manager sind Komponenten, die Sie nur einsetzen, wenn Sie diese Funktionen benötigen . Weitere Informationen finden Sie unter AWS-mitge lieferte Komponenten.

Konzept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Konnektoren	Bei AWS IoT Greengrass V1 Konnektoren handelt es sich um vorgefertigte Module, die Sie auf AWS IoT Greengrass V1 Kerngeräten bereitstellen, um mit der lokalen Infrastruktur, Geräteprotokollen und anderen Cloud-Diensten zu interagieren. AWS	In AWS IoT Greengrass V2, AWS stellt Greengrass-Komponenten bereit, die die Funktionalität implementieren, die von Konnektoren in V1 bereitgestellt wird. Die folgenden AWS IoT Greengrass V2 Komponenten bieten die Funktionalität des Greengrass V1-Connectors: • CloudWatch Metriken-Komponente • AWS IoT Device Defender Komponente • Firehose-Komponente • Komponente des Modbus-RTU-Protokolladapters • Amazon SNS SNS-Komponente Weitere Informationen finden Sie unter AWS-mitgelieferte 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 Kerngerät 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 und Anwendungen auf dem Kerngerät weiterzuleiten. AWS IoT Core	In AWS IoT Greengrass V2 werden verbundene Geräte als Greengrass-Client-Geräte bezeichnet. • Sie verknüpfen Client-Geräte mit Kerngeräten, 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 Einrichtung erstellen müssen, um ein Client-Gerät hinzuzufügen oder zu entfernen. • Um Nachrichten zwischen Client-Geräten und Greengrass-Komponenten weiterzuleiten AWS IoT Core, können Sie eine optionale MQTT-Bridge-Komponente konfigurieren. AWS IoT Greengrass V1 Sowohl AWS IoT Greengrass V2 in als auch können Geräte FreeRTOS ausführen oder die AWS IoT Device SDKoder Greengrass Discovery API verwenden, um Informationen

Konzept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
		über Kerngeräte zu erhalten, mit denen sie sich verbinden können. Die Greengrass Discovery API ist abwärtsko mpatibel. Wenn Sie also Client-Geräte haben, die eine Verbindung zu einem V1-Core-Gerät herstellen, können Sie diese mit einem V2-Core-Gerät verbinden, ohne ihren Code zu ändern. Weitere Informationen zu Client-Geräten finden Sie unter. Interagieren Sie 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 ausgeführt werden, so konfiguriert werden, dass sie auf Volumes und Geräte im Dateisystem des Kerngeräts zugreifen. Diese Dateisystemressourcen werden als lokale Ressourcen bezeichnet.	In können Sie Komponenten ausführen AWS IoT Greengrass V2, bei denen es sich um Lambda-Funktionen, Docker-Container oder native Betriebssystemprozesse oder benutzerdefinierte Laufzeiten handelt. • Wenn Sie eine containerisierte Lambda-Funktion als Komponente importieren, müssen Sie die lokalen Ressourcen angeben, die die Funktion verwendet. • Nicht containerisierte Lambda-Funktionen und Nicht-Lambda-Komponenten können direkt mit lokalen Ressourcen auf Kerngeräten arbeiten, sodass Sie die lokalen Ressourcen, die die Komponente verwendet, nicht angeben müssen.

Konzept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Lokaler Shadow-Dienst	In AWS IoT Greengrass V1 ist der lokale Shadow-Dienst standardmäßig aktiviert und unterstützt nur unbenannte klassische Shadows. 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 Shadow-Dienst AWS IoT Greengrass V2, indem Sie die Shadow-Manager-Komponente bereitstellen. • Sie können die AWS IoT Device SDK V2 in Lambda-Funktionen und benutzerdefinierten Komponenten verwenden, um mit Schatten auf Ihren Geräten zu interagieren. • Der lokale Shadow-Dienst unterstützt benannte Schatten. • Mit dem lokalen Shadow-Dienst können Sie Schatten löschen und gelöschte Schatten mit synchronisieren AWS IoT Core. Weitere Informationen finden Sie unter Interagiere 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 Shadow-Service zu spezifizieren. Abonnements geben an, wo Lambda-Funktionen Ereignisnachrichten empfangen, um sie als Funktionsnutzlasten zu verwenden.	In geben Sie Kommunikationskanäle an AWS IoT Greengrass V2, ohne Abonnements zu verwenden.
		• Komponenten verwalten ihre eigenen Kommunikationskanäle, um mit lokalen Publish/Subscribe-Nachrichten, AWS IoT Core MQTT-Nachrichten und dem lokalen Shadow-Dienst zu interagieren.
		  • Um eine Komponente zu entwickeln, die auf Nachrichten von einer anderen Komponente oder dem AWS IoT Core MQTT-Broker reagiert, können Sie IPC-Schnittstellen (Interprocess Communication) für lokales Publish/Subscribe-Messaging und MQTT-Messaging verwenden.AWS IoT Core
		  • Um eine Komponente zu entwickeln, die mit dem lokalen Shadow-Dienst interagiert, können Sie die IPC-Schnittstelle für den lokalen Shadow-Dienst verwenden.

Konzept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
		• In der Komponent enkonfiguration definiere n Sie Autorisierungsrich tlinien, um die Themen und lokalen Shadows anzugeben, zu deren Verwendung die Komponente berechtigt ist. • Um Kommunikationskanä le zwischen Client-Ge räten, dem lokalen Publish/ Subscribe-Broker und dem MQTT-Broker zu konfigurieren, konfiguri eren und implementieren Sie die AWS IoT Core MQTT-Bridge-Kompon ente. Die MQTT-Bridge-Komponente ermöglich t es Ihnen, mit Client-Ge räten in Komponenten zu interagieren und Nachricht en zwischen Client-Geräten und weiterzuleiten. AWS IoT Core

Konzept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Zugriff auf andere AWS-Services	In AWS IoT Greengrass V1 fügen Sie einer Greengrass-Gruppe eine AWS Identity and Access Management (IAM-) Rolle, die so genannte Gruppenrolle, hinzu. Die Gruppenrolle definiert die Berechtigungen, die Lambda-Funktionen und AWS IoT Greengrass -Features auf dem Kerngerät dieser Gruppe für den Zugriff AWS-Services verwenden.	In AWS IoT Greengrass V2 fügen Sie einem Greengrass-Core-Gerät einen AWS IoT Rollenalias hinzu. Der Rollenalias verweist auf eine IAM-Rolle, die Token-Exchange-Rolle genannt wird. Die Token-Austauschrolle definiert die Berechtigungen, die Greengrass-Komponenten auf dem Kerngerät für den Zugriff AWS-Services verwenden. Weitere Informationen finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

Wählen Sie Ihre Laufzeit (Greengrass Nucleus oder Greengrass Nucleus Lite)

Die Wahl zwischen Greengrass Nucleus und Greengrass Nucleus Lite hängt von Ihren Geräteressourcen und den Funktionen ab, die Ihre Lambda-Funktionen verwenden. Sehen Sie sich die Kompatibilitätsmatrix für die Ereignisquellen in der folgenden Tabelle an und ermitteln Sie anhand des Entscheidungsflussdiagramms, welche Laufzeit für Ihre Migration geeignet ist. Einen detaillierten Vergleich der Funktionen von Greengrass Nucleus und Greengrass Nucleus Lite finden Sie unter Wählen Sie Ihre Laufzeit.

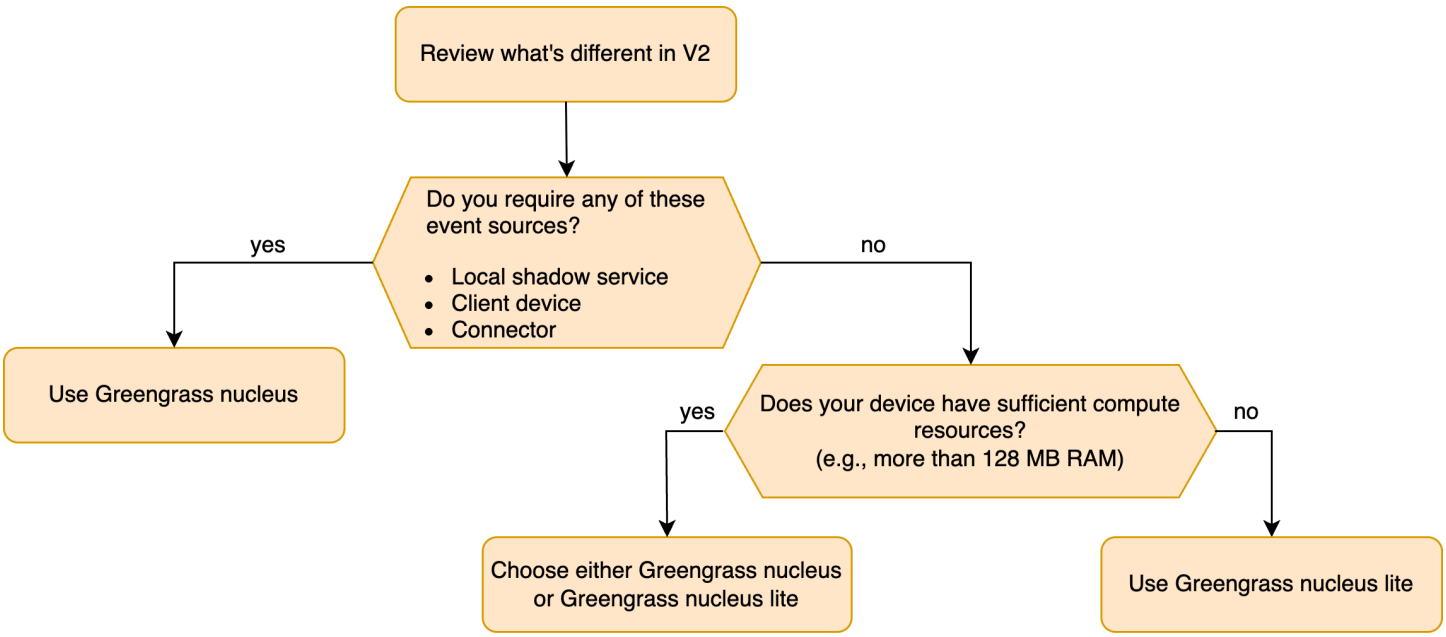
Kompatibilitätsmatrix für die Ereignisquelle

In AWS IoT Greengrass V1 können Lambda-Funktionen mit fünf Arten von Ereignisquellen kommunizieren: anderen Lambda-Funktionen AWS IoT Core, lokalem Schattendienst, Client-Geräten und Konnektoren. Die folgende Tabelle zeigt, welche dieser Ereignisquellen in jeder V2-Laufzeit unterstützt werden.

Hinweis: Die Namen von Ereignisquellen verwenden AWS IoT Greengrass V1 Terminologie. Bei der Migration zu V2 werden Lambda-Funktionen entweder in Lambda-Komponenten (nur in Greengrass Nucleus unterstützt) oder in generische Komponenten (unterstützt sowohl in Greengrass Nucleus als auch in Greengrass Nucleus Lite) umgewandelt.

Ereignisquelle	Grüngraskern	Greengrass Nucleus Lite
Andere Lambda-Funktionen in der Gruppe	✓ (Lambda-Komponenten und generische Komponenten)	✓ (Nur generische Komponenten)
AWS IoT Core Dienst	✓	✓
Lokaler Schattendienst	✓	✗
Client-Gerät	✓	✗
Konnektor	✓	✗

Entscheidungsablauf bei der Auswahl der Laufzeit



Hinweise

1. Informationen zu den Anforderungen und zur Kompatibilität von Greengrass Nucleus Lite finden Sie unter Greengrass Nucleus Lite. Greengrass Nucleus Lite benötigt mindestens 5 MB RAM und ist für Geräte mit begrenzten Ressourcen konzipiert.
2. Der Entscheidungsablauf bietet Anleitungen auf der Grundlage typischer Anwendungsfälle, ist jedoch keine strikte Anforderung. Kunden mit Geräten, die sowohl über begrenzte als auch über ausreichend Ressourcen verfügen, können sich aus Gründen der Benutzerfreundlichkeit dafür entscheiden, eine einzige Runtime für alle Geräte zu verwenden, auch wenn einige Geräte beide Runtime unterstützen könnten.

Nächste Schritte

Nachdem Sie Ihre Laufzeit ausgewählt haben, fahren Sie mit der Einrichtung Ihres Testgeräts fort:

- Für Greengrass Nucleus Runtime: Richten Sie ein neues Gerät ein, um V1-Anwendungen auf V2 zu testen
- Für Greengrass Nucleus Lite Runtime: Richten Sie ein neues Gerät mit Greengrass Nucleus Lite ein

Richten Sie ein neues Gerät ein, um V1-Anwendungen auf V2 zu testen

Um das Risiko für Ihre Geräte in der Produktion zu minimieren, erstellen Sie ein neues Gerät, um Ihre V1-Anwendungen auf V2 zu testen, bevor Sie Ihre Produktionsgeräte aktualisieren.

Wählen Sie auf der Grundlage Ihrer Runtime-Auswahl aus dem vorherigen Schritt eine der folgenden Einrichtungsleitfäden aus:

- Option A — Greengrass Nucleus Runtime: Folgen Sie, Richten Sie ein neues Gerät ein, um V1-Anwendungen auf V2 zu testen wenn Sie sich für Greengrass Nucleus entschieden haben. Mit dieser Option können Sie Lambda-Funktionen als Lambda-Komponenten mit minimalen Codeänderungen importieren und V1-Funktionen wie lokalen Shadow-Service, Client-Geräte und Konnektoren unterstützen.
- Option B — Greengrass Nucleus Lite-Laufzeit: Folgen Sie, Richten Sie ein neues Gerät ein, um V1-Anwendungen auf V2 zu testen (Greengrass Nucleus Lite) wenn Sie sich für Greengrass

Nucleus Lite entschieden haben. Diese Option erfordert die Konvertierung von Lambda-Funktionen in generische Komponenten mithilfe des AWS IoT Device SDK V2- oder AWS IoT Greengrass Komponenten-SDK, ist jedoch für Geräte mit begrenzten Ressourcen optimiert.

Richten Sie ein neues Gerät ein, um V1-Anwendungen auf V2 zu testen

Richten Sie ein neues AWS IoT Greengrass V2 Kerngerät ein, um die AWS bereitgestellten Komponenten und AWS Lambda Funktionen für Ihre AWS IoT Greengrass V1 Anwendungen bereitzustellen und zu testen. Sie können dieses V2-Kerngerät auch verwenden, um zusätzliche benutzerdefinierte Greengrass-Komponenten zu entwickeln und zu testen, die native Prozesse auf Kerngeräten ausführen. Nachdem Sie Ihre Anwendungen auf einem V2-Kerngerät getestet haben, können Sie Ihre vorhandenen V1-Kerngeräte auf V2 aufrüsten und die V2-Komponenten bereitstellen, die Ihre V1-Funktionalität bereitstellen.

Schritt 1: AWS IoT Greengrass V2 Auf einem neuen Gerät installieren

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 Komponenten entwickelt und bereitgestellt werden. In diesem Tutorial wird die automatische Bereitstellung verwendet, um ein Gerät schnell einzurichten. Wenn Sie die AWS IoT Greengrass Core-Software v2.x installieren, geben Sie das `--deploy-dev-tools` Argument für die Bereitstellung der Greengrass-CLI an, sodass 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 mithilfe eines Hardware-Sicherheitsmoduls (HSM), finden Sie unter. Installieren Sie die AWS IoT Greengrass Core-Software

(Optional) Protokollierung bei Amazon CloudWatch Logs aktivieren

Um einem V2-Core-Gerät das Hochladen von Protokollen auf Amazon CloudWatch Logs zu ermöglichen, können Sie die von uns AWS bereitgestellte Log-Manager-Komponente bereitstellen. Sie können CloudWatch Logs verwenden, um Komponentenprotokolle anzuzeigen, sodass Sie ohne Zugriff auf das Dateisystem des Kerngeräts debuggen und Fehler beheben können. Weitere Informationen finden Sie unter AWS IoT Greengrass Protokolle überwachen.

Schritt 2: AWS IoT Greengrass V2 Komponenten für die Migration AWS IoT Greengrass V1 von Anwendungen erstellen und bereitstellen

Sie können die meisten AWS IoT Greengrass V1 Anwendungen auf ausführen AWS IoT Greengrass V2. Sie können Lambda-Funktionen als Komponenten importieren AWS IoT Greengrass V2, auf denen sie ausgeführt werden, und Sie können AWS bereitgestellte Komponenten verwenden, die dieselbe Funktionalität wie AWS IoT Greengrass Konnektoren bieten.

Sie können auch benutzerdefinierte Komponenten entwickeln, um jede Funktion oder Runtime für die Ausführung auf Greengrass-Core-Geräten zu erstellen. Informationen zum lokalen Entwickeln und Testen von Komponenten finden Sie unterAWS IoT Greengrass Komponenten erstellen.

Themen

- V1-Lambda-Funktionen importieren
- Verwenden Sie V1-Anschlüsse
- Ausführen von Docker-Containern
- Connect V1 Greengrass-Geräte
- Aktivieren Sie den lokalen Shadow-Dienst
- Integrieren Sie mit AWS IoT SiteWise

V1-Lambda-Funktionen importieren

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

- Importieren Sie V1-Lambda-Funktionen direkt als Greengrass-Komponenten.
- Aktualisieren Sie Ihre Lambda-Funktionen, um die Greengrass-Bibliotheken in Version 2 zu verwenden, AWS IoT Device SDK 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-Funktion Funktionen wie Stream-Manager oder lokale Geheimnisse verwendet, müssen Sie Abhängigkeiten von den von AWS-bereitgestellten Komponenten definieren, die diese Funktionen verpacken. Wenn Sie die Lambda-Funktionskomponente bereitstellen, umfasst die

Bereitstellung auch die Komponente für jedes Feature, das Sie als Abhängigkeit definieren. In der Bereitstellung können Sie Parameter konfigurieren, z. B. welche Geheimnisse auf dem Kerngerät 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:

- Greifen Sie auf andere Dienste zu AWS

  Wenn Ihre Lambda-Funktion AWS Anmeldeinformationen verwendet, um Anfragen an andere AWS Dienste zu stellen, muss die Token-Austauschrolle des Kerngeräts es dem Kerngerät ermöglichen, 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 dies `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 Kerngeräts muss es dem Kerngerät ermöglichen, auf die AWS Cloud Ziele zuzugreifen, die Sie mit Stream Manager verwenden. Weitere Informationen finden Sie unter Stream-Manager.

- Lokale Geheimnisse

  Wenn Ihre Lambda-Funktion lokale Geheimnisse verwendet, geben Sie diese `aws.greengrass.SecretManager` als Komponentenabhängigkeit an, wenn Sie die Funktion importieren. Wenn Sie die Secret Manager-Komponente bereitstellen, geben Sie die geheimen Ressourcen an, die auf den Ziel-Core-Geräten bereitgestellt werden sollen. Die Token-Austauschfunktion des Kerngeräts muss es dem Kerngerät 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 Erlaubnis erteilt, den GetSecretValue IPC-Vorgang in der V2 zu verwenden. AWS IoT Device SDK

- Lokale Schatten

  Wenn Ihre Lambda-Funktion mit lokalen Schatten interagiert, müssen Sie den Lambda-Funktionscode aktualisieren, um V2 verwenden zu können. AWS IoT Device SDK Sie müssen beim Import der `aws.greengrass.ShadowManager` Funktion auch eine

Komponentenabhängigkeit angeben. Weitere Informationen finden Sie unter Interagiere mit Geräteschatten.

Wenn Sie Ihre Lambda-Funktionskomponente bereitstellen, konfigurieren Sie sie so, dass sie über eine IPC-Autorisierungsrichtlinie verfügt, die die Erlaubnis erteilt, die Schatten-IPC-Operationen in der V2 zu verwenden. AWS IoT Device SDK

- 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 implementieren Sie die ältere Abonnement-Router-Komponente, wenn Sie Ihre Lambda-Funktion bereitstellen. Wenn Sie die ältere Abonnement-Router-Komponente bereitstellen, geben Sie die Abonnements an, die die Lambda-Funktion verwendet.

    > **Note**
    >
    > Die ältere Abonnement-Router-Komponente 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 IPC-Schnittstelle (Interprocess Communication) in AWS IoT Device SDK V2 zu verwenden, müssen Sie die ältere Abonnement-Router-Komponente nicht bereitstellen. Weitere Informationen finden Sie in den folgenden Interprozesskommunikationsdiensten:
    >
    > - 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 publish/subscribe Themen weiterzuleiten, die Sie als Ereignisquellen angeben.
  - Wenn Ihre Lambda-Funktion Nachrichten an lokal verbundene Geräte veröffentlicht, müssen Sie den Lambda-Funktionscode aktualisieren, um AWS IoT Device SDK V2 zum Veröffentlichen lokaler publish/subscribe Nachrichten zu verwenden. Sie müssen auch die MQTT-Bridge-Komponente konfigurieren und bereitstellen, um Nachrichten vom lokalen publish/subscribe Message Broker an die angeschlossenen 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 AWS Lambda Funktionen ausführen.

Verwenden Sie V1-Anschlüsse

Sie können AWS bereitgestellte Komponenten einsetzen, die dieselbe Funktionalität wie einige AWS IoT Greengrass Konnektoren bieten. Wenn Sie die Bereitstellung erstellen, können Sie die Parameter der Konnektoren konfigurieren.

Die folgenden AWS IoT Greengrass V2 Komponenten bieten die Funktionalität des Greengrass V1-Connectors:

- CloudWatch Metriken-Komponente
- AWS IoT Device Defender Komponente
- Firehose-Komponente
- Komponente des Modbus-RTU-Protokolladapters
- Amazon SNS SNS-Komponente

Ausführen von Docker-Containern

AWS IoT Greengrass V2 stellt keine Komponente bereit, die den V1-Connector für die Bereitstellung von Docker-Anwendungen direkt ersetzt. Sie können jedoch die Docker-Anwendungsmanager-Komponente verwenden, um Docker-Images herunterzuladen, und anschließend benutzerdefinierte Komponenten erstellen, die Docker-Container aus den heruntergeladenen Images ausführen. Weitere Informationen erhalten Sie unter Führen Sie einen Docker-Container aus und Docker-Anwendungsmanager.

Connect V1 Greengrass-Geräte

Verbundene Geräte in AWS IoT Greengrass V1 werden als Client-Geräte in AWS IoT Greengrass V2 bezeichnet. 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 deren Anwendungscode zu ändern. Damit Client-Geräte eine Verbindung zu

einem V2-Core-Gerät herstellen können, stellen Sie Greengrass-Komponenten bereit, die die Unterstützung von Client-Geräten ermöglichen, und ordnen Sie die Client-Geräte dem Core-Gerät zu. Um Nachrichten zwischen Client-Geräten, dem AWS IoT Core Cloud-Dienst und Greengrass-Komponenten (einschließlich Lambda-Funktionen) weiterzuleiten, müssen Sie die MQTT-Bridge-Komponente bereitstellen und konfigurieren. Sie können die IP-Detektorkomponente einsetzen, um Verbindungsinformationen automatisch zu erkennen, oder Sie können Endpunkte manuell verwalten. Weitere Informationen finden Sie unter Interagieren Sie mit lokalen IoT-Geräten.

Aktivieren Sie den lokalen Shadow-Dienst

In AWS IoT Greengrass V2 wird der lokale Shadow-Dienst durch die von AWS-bereitgestellte Shadow-Manager-Komponente implementiert. AWS IoT Greengrass V2 beinhaltet auch Unterstützung für benannte Schatten. Damit Ihre Komponenten mit lokalen Shadows interagieren und Shadow-Status mit ihnen synchronisieren können AWS IoT Core, konfigurieren und implementieren Sie die Shadow-Manager-Komponente und verwenden Sie die Shadow-IPC-Operationen in Ihrem Komponentencode. Weitere Informationen finden Sie unter Interagiere mit Geräteschatten.

Integrieren Sie 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 stellt ein Installationsskript bereit, das die AWS IoT SiteWise Komponenten für Sie bereitstellt.

Schritt 3: Testen Sie Ihre Anwendungen AWS IoT Greengrass V2

Nachdem Sie V2-Komponenten erstellt und auf Ihrem neuen V2-Kerngerät bereitgestellt haben, stellen Sie sicher, dass Ihre Anwendungen Ihren Erwartungen entsprechen. Sie können die Protokolle des Geräts überprüfen, um die Standardausgabe- (stdout) und Standardfehlermeldungen (stderr) Ihrer Komponenten einzusehen. Weitere Informationen finden Sie unter AWS IoT Greengrass Protokolle überwachen.

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

Nachdem Sie überprüft haben, dass Ihre Anwendungen auf einem V2-Core-Gerät funktionieren, können Sie die Greengrass-Komponenten Ihrer Anwendung auf anderen Kerngeräten bereitstellen. Wenn Sie benutzerdefinierte Komponenten entwickelt haben, die native Prozesse oder Docker-

Container ausführen, müssen Sie diese Komponenten zunächst im AWS IoT Greengrass Service veröffentlichen, um sie auf anderen Kerngeräten bereitzustellen.

Richten Sie ein neues Gerät ein, um V1-Anwendungen auf V2 zu testen (Greengrass Nucleus Lite)

Richten Sie ein neues Gerät mit Greengrass Nucleus Lite ein, um generische Komponenten zu testen, die Sie für die Migration Ihrer AWS IoT Greengrass V1 Anwendungen auf V2 erstellen. Greengrass Nucleus Lite ist eine leichtgewichtige Runtime, die für Geräte mit beschränkten Ressourcen optimiert ist. Sie können dieses Gerät verwenden, um benutzerdefinierte Greengrass-Komponenten zu entwickeln und zu testen, die native Prozesse ausführen. Nachdem Sie Ihre Anwendungen auf einem Greengrass Nucleus Lite-Gerät getestet haben, können Sie die V2-Komponenten auf anderen Geräten bereitstellen, auf denen Greengrass Nucleus Lite ausgeführt wird, oder Ihre vorhandenen V1-Kerngeräte auf V2 aufrüsten.

Schritt 1: Installieren Sie Greengrass Nucleus Lite auf einem neuen Gerät

Installieren Sie den Greengrass Nucleus Lite auf einem neuen Gerät. Folgen Sie der Greengrass Nucleus Lite-Installationsanleitung, um ein Gerät einzurichten.

> **Note**
>
> Greengrass Nucleus Lite unterstützt derzeit keine lokalen Shadow-Dienste, Client-Geräte oder Konnektoren. Stellen Sie sicher, dass Ihre V1-Anwendungen nicht auf diese Funktionen angewiesen sind, bevor Sie mit diesem Handbuch fortfahren.

Schritt 2: Generische Komponenten zur Migration von AWS IoT Greengrass V1 Lambda-Funktionen erstellen und bereitstellen

Um die Funktionalität Ihrer AWS IoT Greengrass V1 Lambda-Funktionen auf Greengrass Nucleus Lite zu replizieren, müssen Sie sie in generische Komponenten konvertieren. Dazu müssen Sie Ihren Lambda-Funktionscode so umschreiben, dass er das AWS IoT Device SDK V2- oder AWS IoT Greengrass Komponenten-SDK anstelle des AWS IoT Greengrass Core-SDK verwendet.

In der folgenden Tabelle sind die in der V2-Version SDKs verwendeten Komponenten in diesem Handbuch aufgeführt:

SDK	Mindestversion
AWS IoT Device SDK für Python v2	v1.11.3
AWS IoT Device SDK für Java v2	v1.9.3
AWS IoT Device SDK für v2 JavaScript	v1.12.0
AWS IoT Greengrass Komponenten-SDK (C/C++) (derzeit in der Vorschauversion)	v0.4.0

Die folgenden Beispiele zeigen Lambda-Funktionen, die das AWS IoT Greengrass V1 Core SDK und ihre entsprechenden generischen Komponenten verwenden, mit Komponentencode, Rezepten und Build-Anweisungen in mehreren Programmiersprachen für zwei Hauptszenarien:

- Lokale Kommunikation — Komponenten, die über lokales Pub/Sub mit anderen Komponenten auf demselben Gerät kommunizieren
- Cloud-Kommunikation — Komponenten, die mit AWS IoT Core oder anderen Diensten kommunizieren AWS

Szenario 1: Lokale Kommunikation (Herausgeber → Prozessor → Abonnent)

Dieses Szenario demonstriert die Konvertierung einer V1-Lambda-Funktion, die lokale pub/sub Kommunikation verwendet, in eine generische V2-Komponente.

Anwendungsarchitektur

Dieses Beispiel umfasst drei Komponenten:

- Herausgeber Lambda veröffentlicht Temperaturdaten
- Prozessor Lambda empfängt Daten und verarbeitet sie
- Prozessor Lambda veröffentlicht das verarbeitete Ergebnis an den Abonnenten Lambda

Das folgende Codebeispiel konzentriert sich auf den Prozessor Lambda, der das Abonnieren und Veröffentlichen von Nachrichten für die lokale Kommunikation demonstriert.

V1-Gruppenabonnements

In ermöglichen AWS IoT Greengrass V1 die folgenden Gruppenabonnements die Kommunikation zwischen Lambda-Funktionen:

Abonnement 1: Herausgeber → Prozessor

- Quelle: Lambda (Herausgeber)
- Ziel: Lambda (Prozessor)
- Thema: Sensoren/Temperatur

Abonnement 2: Prozessor → Abonnent

- Quelle: Lambda (Prozessor)
- Ziel: Lambda (Abonnent)
- Thema: Lambda/Alerts

Prozessor-Lambda-Funktion (V1)

Python

```
import greengrasssdk
import json

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

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

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

    # Process: Check if temperature is high.
    if temperature > 80:
        alert_data = {
```

```
            'sensor_id': sensor_id,
            'temperature': temperature,
            'alert': 'HIGH_TEMPERATURE'
        }

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

        print(f"Alert sent to subscriber Lambda")

    return {'statusCode': 200}
```

Java

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

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

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

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

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

```
        int temperature = temp.intValue();

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

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

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

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

            iotDataClient.publish(publishRequest);

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

        return "Success";
    }
}
```

JavaScript

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

const iotClient = new greengrasssdk.IotData();

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

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

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

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

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

C

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

static aws_greengrass_iot_data_client *iot_client = NULL;

void on_message_received(const char *topic, const uint8_t *payload, size_t
 payload_len, void *user_data) {
    // Parse the incoming message.
```

```
    char *payload_str = strndup((char *)payload, payload_len);
    json_error_t error;
    json_t *event = json_loads(payload_str, 0, &error);
    free(payload_str);

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

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

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

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

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

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

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

        aws_greengrass_iot_data_publish(iot_client, &params);

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

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

```
    json_decref(event);
}

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

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

    aws_greengrass_iot_data_subscribe(iot_client, &subscribe_params);

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

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

    return 0;
}
```

C++

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

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

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

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

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

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

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

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

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

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

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

```
    json_decref(event);
}

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

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

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

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

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

    free(alert_payload);
    json_decref(alert_data);
}

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

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

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

void run() {
```

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

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

    return 0;
}
```

Generische Komponente (V2)

Um dieselbe Funktionalität in zu erreichen AWS IoT Greengrass V2, erstellen Sie eine generische Komponente mit den folgenden Komponenten:

1. Code der Komponente

Python

Voraussetzungen: Bevor Sie diesen Komponentencode verwenden, installieren und überprüfen Sie das AWS IoT Device SDK für Python auf Ihrem Greengrass-Gerät:

```
# Install the SDK
pip3 install awsiotsdk

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

Wenn Sie während der Installation auf Abhängigkeitskonflikte stoßen, versuchen Sie, eine bestimmte Version von zu installieren. AWS IoT Device SDK

Wenn der Bestätigungsbefehl „SDK erfolgreich installiert“ ausgibt, können Sie den Komponentencode verwenden:

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

ipc_client = GreengrassCoreIPCClientV2()

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

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

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

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

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

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

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

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

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

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

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

Java

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

public class TemperatureProcessor {
    private static GreengrassCoreIPCClientV2 ipcClient;

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

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

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

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

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

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

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

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

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

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

```
                    .withPublishMessage(publishMessage);

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

JavaScript

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

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

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

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

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

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

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

            streamingOperation.on('streamError', (error) => {
                console.error('Stream error:', error);
            });

            streamingOperation.on('ended', () => {
```

```
            console.log('Subscription stream ended');
        });

        await streamingOperation.activate();

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

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

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

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

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

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

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

            await this.ipcClient.publishToTopic(publishRequest);
            console.log('Alert sent to AlertHandler component');
        }

    } catch (error) {
```

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

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

C

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

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

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

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

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

    while (true) {
        pthread_mutex_lock(&alert_mutex);
        while (!has_pending_alert) {
```

```
            pthread_cond_wait(&alert_cond, &alert_mutex);
        }

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

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

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

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

    return NULL;
}

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

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

    GgMap map = gg_obj_into_map(payload);

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

```
        return;
    }
    GgBuffer sensor_id = gg_obj_into_buf(*sensor_id_obj);

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

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

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

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

    gg_sdk_init();

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

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

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

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

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

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

    return 0;
}
```

C++

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

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

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

```
};

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

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

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

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

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

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

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

class SensorCallback : public gg::ipc::LocalTopicCallback {
    void operator()(
        std::string_view topic,
        gg::Object payload,
        gg::ipc::Subscription& handle
    ) override {
        (void) topic;
        (void) handle;
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

    return 0;
}
```

2. Erstellen und verpacken Sie die Komponente

Einige Sprachen müssen vor der Bereitstellung erstellt oder gepackt werden.

Python

Python erfordert keine Kompilierung. Die Komponente kann die .py-Datei direkt verwenden.

Java

Um eine ausführbare JAR zu erstellen, in der alle Abhängigkeiten gebündelt sind:

1. Erstellen Sie eine `pom.xml` Datei in Ihrem Projektverzeichnis:

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

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

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

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

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

2. Erstellen Sie das JAR:

```
mvn clean package
```

Dies erstellt, `target/temperature-processor-1.0.0.jar` wobei alle Abhängigkeiten enthalten sind.

3. Laden Sie das JAR zur Bereitstellung in Ihren S3-Bucket hoch.

JavaScript

Um Ihre Komponente Node.js mit allen Abhängigkeiten zu verpacken:

1. Erstellen Sie eine `package.json` Datei:

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

2. Installieren Sie Abhängigkeiten auf Ihrem Entwicklungscomputer:

```
npm install
```

Dadurch wird ein `node_modules` Ordner erstellt, der die AWS AWS IoT Device SDK Version v2 enthält.

3. Package für die Bereitstellung:

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

4. Laden Sie die Zip-Datei zur Bereitstellung in Ihren S3-Bucket hoch.

> **Note**
>
> Auf dem Greengrass-Gerät muss Node.js Runtime installiert sein (Version 14 oder höher). Sie müssen es nicht `npm install` auf dem Greengrass-Core-Gerät ausführen, da das Komponentenartefakt alle Abhängigkeiten im gebündelten `node_modules` Ordner enthält.

C

Voraussetzungen:

Um das SDK und die Komponente zu erstellen, benötigen Sie die folgenden Build-Abhängigkeiten:

- GCC oder Clang
- CMake (mindestens Version 3.22)
- Marke oder Ninja

Installieren Sie die Build-Abhängigkeiten:

Auf Ubuntu/Debian:

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

Auf Amazon Linux:

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

Erstellen Sie eine Datei CMake Lists.txt für Ihre Komponente:

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

set(CMAKE_C_STANDARD 11)

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

# Build your component executable
```

```
add_executable(temperature_processor temperature_processor.c)
target_link_libraries(temperature_processor gg-sdk)
```

Schritte zum Erstellen:

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

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

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

C++

Voraussetzungen:

Um das SDK und die Komponente zu erstellen, benötigen Sie die folgenden Build-Abhängigkeiten:

- GCC oder Clang mit C++20-Unterstützung
- CMake (mindestens Version 3.22)
- Marke oder Ninja

Installieren Sie die Build-Abhängigkeiten:

Auf Ubuntu/Debian:

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

Auf Amazon Linux:

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

Erstellen Sie eine Datei CMake Lists.txt für Ihre Komponente:

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

```
set(CMAKE_CXX_STANDARD 20)

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

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

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

Schritte zum Erstellen:

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

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

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

3. Rezept für Komponenten

Aktualisieren Sie das Array „Ressourcen“ mit den aktuellen Themen, die Ihre Komponente verwendet.

Python

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

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

Java

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

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

JavaScript

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

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

C/C++

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

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

Szenario 2: Cloud-Kommunikation

Dieses Szenario demonstriert die Konvertierung einer V1-Lambda-Funktion, die mit AWS IoT Core kommuniziert, in eine generische V2-Komponente.

Anwendungsarchitektur

In diesem Beispiel wird eine mit der Cloud verbundene Architektur verwendet:

- AWS IoT Core sendet Befehle an das Gerät
- Controller Lambda empfängt Befehle und verarbeitet sie
- Controller Lambda sendet Telemetriedaten zurück an AWS IoT Core

Dieses Beispiel konzentriert sich auf den Controller Lambda, der sowohl das Empfangen von Nachrichten als auch das Veröffentlichen von Nachrichten an AWS IoT Core demonstriert.

V1-Gruppenabonnements

In AWS IoT Greengrass V1 ermöglichen die folgenden Gruppenabonnements die Kommunikation zwischen der Lambda-Funktion und AWS IoT Core:

Abonnement 1: IoT Cloud → Lambda

- Quelle: IoT Cloud
- Ziel: Lambda () DeviceController
- Thema: Befehle/Geräte1

Abonnement 2: Lambda → IoT Cloud

- Quelle: Lambda () DeviceController
- Ziel: IoT Cloud
- Thema: Telemetrie/Gerät1

Lambda-Funktion des Controllers (V1)

Python

```
import greengrasssdk
import json
import time

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

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

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

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

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

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

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

    return {'statusCode': 200}

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

Java

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

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

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

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

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

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

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

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

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

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

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

            iotDataClient.publish(publishRequest);

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

```
        }

        return "Success";
    }

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

JavaScript

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

const iotClient = new greengrasssdk.IotData();

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

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

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

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

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

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

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

C

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

static aws_greengrass_iot_data_client *iot_client = NULL;

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

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

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

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

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

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

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

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

        char *telemetry_payload = json_dumps(telemetry_data, JSON_COMPACT);

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

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

        aws_greengrass_iot_data_publish(iot_client, &params);

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

        free(telemetry_payload);
        json_decref(telemetry_data);
    }

    json_decref(event);
}
```

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

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

    aws_greengrass_iot_data_subscribe(iot_client, &subscribe_params);

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

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

    return 0;
}
```

C++

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

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

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

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

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

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

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

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

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

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

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

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

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

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

            char *telemetry_payload = json_dumps(telemetry_data, JSON_COMPACT);

            // Publish telemetry to IoT Core.
            std::string telemetry_topic = "telemetry/" + std::string(device_id);

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

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

            std::cout << "Telemetry sent to cloud: " << telemetry_payload <<
std::endl;

            free(telemetry_payload);
            json_decref(telemetry_data);
        }

        json_decref(event);
    }

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

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

        std::cout << "Device Controller Lambda started" << std::endl;
        std::cout << "Subscribed to " << topic << std::endl;
```

```
        std::cout << "Waiting for commands from IoT Core..." << std::endl;
    }

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

int main(int argc, char *argv[]) {
    try {
        DeviceController controller;
        controller.subscribe_to_topic("commands/device1");
        controller.run();
    } catch (const std::exception& e) {
        std::cerr << "Error: " << e.what() << std::endl;
        return 1;
    }

    return 0;
}
```

Generische Komponente (V2)

Um dieselbe Funktionalität in zu erreichen AWS IoT Greengrass V2, erstellen Sie eine generische Komponente mit den folgenden Komponenten:

1. Code der Komponente

Python

Voraussetzungen: Bevor Sie diesen Komponentencode verwenden, installieren und überprüfen Sie das AWS IoT Device SDK für Python auf Ihrem Greengrass-Gerät:

```
# Install the SDK
pip3 install awsiotsdk

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

Wenn Sie während der Installation auf Abhängigkeitskonflikte stoßen, versuchen Sie, eine bestimmte Version von zu installieren. AWS IoT Device SDK

Wenn der Bestätigungsbefehl „SDK erfolgreich installiert“ ausgibt, können Sie den Komponentencode verwenden:

```
from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import QOS
import json
import time

ipc_client = GreengrassCoreIPCClientV2()

def on_command(event):
    """
    Receives commands from IoT Core,
    processes them, and sends telemetry back to cloud
    """
    try:
        # Receive command from IoT Core.
        data = json.loads(event.message.payload.decode('utf-8'))
        command = data.get('command')
        device_id = data.get('device_id', 'device1')

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

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

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

            ipc_client.publish_to_iot_core(
                topic_name=f'telemetry/{device_id}',
                qos=QOS.AT_LEAST_ONCE,
                payload=json.dumps(telemetry_data).encode('utf-8')
            )
```

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

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

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

def main():
    print("Device Controller component starting...")

    # Subscribe to commands from IoT Core.
    ipc_client.subscribe_to_iot_core(
        topic_name='commands/device1',
        qos=QOS.AT_LEAST_ONCE,
        on_stream_event=on_command
    )

    print("Subscribed to commands/device1 from IoT Core")
    print("Waiting for commands from cloud...")

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

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

Java

```
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.model.QOS;
import software.amazon.awssdk.aws.greengrass.model.SubscribeToIoTCoreRequest;
import software.amazon.awssdk.aws.greengrass.model.IoTCoreMessage;
import software.amazon.awssdk.aws.greengrass.model.PublishToIoTCoreRequest;
import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class DeviceController {
```

```
    private static GreengrassCoreIPCClientV2 ipcClient;

    public static void main(String[] args) {
        System.out.println("Device Controller component starting...");

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

            SubscribeToIoTCoreRequest subscribeRequest = new
SubscribeToIoTCoreRequest()
                .withTopicName("commands/device1")
                .withQos(QOS.AT_LEAST_ONCE);

            ipcClient.subscribeToIoTCore(
                subscribeRequest,
                DeviceController::onCommand,
                Optional.empty(),
                Optional.empty()
            );

            System.out.println("Subscribed to commands/device1 from IoT Core");
            System.out.println("Waiting for commands from cloud...");

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

    public static void onCommand(IoTCoreMessage message) {
        try {
            String payload = new String(message.getMessage().getPayload(),
StandardCharsets.UTF_8);

            // Simple JSON parsing.
            String command = extractJsonValue(payload, "command");
            String deviceId = extractJsonValue(payload, "device_id");
            if (deviceId == null || deviceId.isEmpty()) {
                deviceId = "device1";
            }
```

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

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

                // Build JSON manually.
                String telemetryJson = String.format(
                    "{\"device_id\":\"%s\",\"status\":\"%s\",\"timestamp\":%.3f}",
                    deviceId, status, System.currentTimeMillis() / 1000.0
                );
                byte[] telemetryBytes =
 telemetryJson.getBytes(StandardCharsets.UTF_8);

                PublishToIoTCoreRequest publishRequest = new
 PublishToIoTCoreRequest()
                    .withTopicName("telemetry/" + deviceId)
                    .withQos(QOS.AT_LEAST_ONCE)
                    .withPayload(telemetryBytes);

                ipcClient.publishToIoTCore(publishRequest);

                System.out.println("Telemetry sent to cloud: " + telemetryJson);
            }
        } catch (Exception e) {
            System.err.println("Error processing command: " + e.getMessage());
        }
    }

    private static String extractJsonValue(String json, String key) {
        Pattern pattern = Pattern.compile("\"" + Pattern.quote(key) + "\"\\s*:\\s*
\"([^\"]+)\"");
        Matcher matcher = pattern.matcher(json);
        return matcher.find() ? matcher.group(1) : null;
    }

    private static String getDeviceStatus() {
        return "online";
    }
}
```

JavaScript

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

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

    async start() {
        console.log('Device Controller component starting...');

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

            const subscribeRequest = {
                topicName: 'commands/device1',
                qos: 1
            };

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

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

            streamingOperation.on('streamError', (error) => {
                console.error('Stream error:', error);
            });

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

            await streamingOperation.activate();

            console.log('Subscribed to commands/device1 from IoT Core');
            console.log('Waiting for commands from cloud...');

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

```
        }
    }

    async onCommand(message) {
        try {
            const payload = message.message.payload.toString('utf-8');
            const data = JSON.parse(payload);

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

            console.log(`Received command from cloud: ${command}`);

            if (command === 'get_status') {
                const status = this.getDeviceStatus();

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

                const telemetryJson = JSON.stringify(telemetryData);

                const publishRequest = {
                    topicName: `telemetry/${deviceId}`,
                    qos: 1,
                    payload: Buffer.from(telemetryJson, 'utf-8')
                };

                await this.ipcClient.publishToIoTCore(publishRequest);
                console.log(`Telemetry sent to cloud: ${telemetryJson}`);
            }

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

    getDeviceStatus() {
        return 'online';
    }
}
```

```
// Start the component.
const controller = new DeviceController();
controller.start();
```

C

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

#define COMMAND_TOPIC "commands/device1"
#define TELEMETRY_TOPIC "telemetry/device1"

typedef struct {
    char device_id[64];
    char command[64];
} CommandData;

static pthread_mutex_t command_mutex = PTHREAD_MUTEX_INITIALIZER;
static pthread_cond_t command_cond = PTHREAD_COND_INITIALIZER;
static CommandData pending_command;
static bool has_pending_command = false;

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

static void *telemetry_publisher_thread(void *arg) {
    (void) arg;

    while (true) {
        pthread_mutex_lock(&command_mutex);
```

```
        while (!has_pending_command) {
            pthread_cond_wait(&command_cond, &command_mutex);
        }

        CommandData cmd = pending_command;
        has_pending_command = false;
        pthread_mutex_unlock(&command_mutex);

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

            // Create telemetry JSON string.
            char telemetry_json[512];
            snprintf(telemetry_json, sizeof(telemetry_json),
                     "{\"device_id\":\"%s\",\"status\":\"%s\",\"timestamp\":%ld}",
                     cmd.device_id, status, time(NULL));

            GgBuffer telemetry_buf = {
                .data = (uint8_t *)telemetry_json,
                .len = strlen(telemetry_json)
            };

            // Publish telemetry to IoT Core.
            GgError ret = ggipc_publish_to_iot_core(GG_STR(TELEMETRY_TOPIC),
 telemetry_buf, 0);

            if (ret != GG_ERR_OK) {
                fprintf(stderr, "Failed to publish telemetry to IoT Core\n");
            } else {
                printf("Telemetry sent to cloud: device_id=%s, status=%s\n",
 cmd.device_id, status);
            }
        }
    }

    return NULL;
}

static void on_cloud_command(
    void *ctx, GgBuffer topic, GgBuffer payload, GgIpcSubscriptionHandle handle
) {
    (void) ctx;
    (void) topic;
```

```
    (void) handle;

    printf("Received command from IoT Core: %.*s\n", (int)payload.len,
payload.data);

    // Parse JSON payload (comes as raw buffer from IoT Core).
    // For simplicity, we'll do basic string parsing.

    // Extract command and device_id from JSON string.
    char payload_str[512];
    snprintf(payload_str, sizeof(payload_str), "%.*s", (int)payload.len,
payload.data);

    // Simple JSON parsing (looking for "command":"get_status").
    char *command_start = strstr(payload_str, "\"command\"");
    char *device_id_start = strstr(payload_str, "\"device_id\"");

    if (command_start) {
        pthread_mutex_lock(&command_mutex);

        char *cmd_value = strstr(command_start, ":");
        if (cmd_value) {
            cmd_value = strchr(cmd_value, '"');
            if (cmd_value) {
                cmd_value++;
                char *cmd_end = strchr(cmd_value, '"');
                if (cmd_end) {
                    size_t cmd_len = cmd_end - cmd_value;
                    if (cmd_len < sizeof(pending_command.command)) {
                        strncpy(pending_command.command, cmd_value, cmd_len);
                        pending_command.command[cmd_len] = '\0';
                    }
                }
            }
        }

        // Extract device_id or use default.
        if (device_id_start) {
            char *dev_value = strstr(device_id_start, ":");
            if (dev_value) {
                dev_value = strchr(dev_value, '"');
                if (dev_value) {
                    dev_value++;
                    char *dev_end = strchr(dev_value, '"');
```

```
                    if (dev_end) {
                        size_t dev_len = dev_end - dev_value;
                        if (dev_len < sizeof(pending_command.device_id)) {
                            strncpy(pending_command.device_id, dev_value, dev_len);
                            pending_command.device_id[dev_len] = '\0';
                        }
                    }
                }
            }
        } else {
            strcpy(pending_command.device_id, "device1");
        }

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

        has_pending_command = true;
        pthread_cond_signal(&command_cond);
        pthread_mutex_unlock(&command_mutex);
    }
}

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

    gg_sdk_init();

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

    // Start telemetry publisher thread.
    pthread_t telemetry_thread;
    if (pthread_create(&telemetry_thread, NULL, telemetry_publisher_thread, NULL) !=
 0) {
        fprintf(stderr, "Failed to create telemetry publisher thread\n");
        exit(1);
    }

    // Subscribe to commands from IoT Core.
    GgIpcSubscriptionHandle handle;
```

```
    ret = ggipc_subscribe_to_iot_core(
        GG_STR(COMMAND_TOPIC),
        0,
        &on_cloud_command,
        NULL,
        &handle
    );

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

    printf("Subscribed to %s\n", COMMAND_TOPIC);
    printf("Waiting for commands from IoT Core...\n");

    while (true) {
        sleep(1);
    }

    return 0;
}
```

C++

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

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

constexpr std::string_view COMMAND_TOPIC = "commands/device1";
constexpr std::string_view TELEMETRY_TOPIC = "telemetry/device1";

struct CommandData {
```

```
    std::string device_id;
    std::string command;
};

static std::mutex command_mutex;
static std::condition_variable command_cv;
static CommandData pending_command;
static bool has_pending_command = false;

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

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

    while (true) {
        std::unique_lock<std::mutex> lock(command_mutex);
        command_cv.wait(lock, [] { return has_pending_command; });

        CommandData cmd = pending_command;
        has_pending_command = false;
        lock.unlock();

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

            // Get current timestamp.
            auto now = std::time(nullptr);

            // Create telemetry JSON payload.
            std::string telemetry_payload = "{\"device_id\":\"" + cmd.device_id +
                                            "\",\"status\":\"" + status +
                                            "\",\"timestamp\":" + std::to_string(now)
 + "}";

            // Publish telemetry to IoT Core.
            gg::Buffer telemetry_buffer(telemetry_payload);
            auto error = client.publish_to_iot_core(TELEMETRY_TOPIC,
 telemetry_buffer);

            if (error) {
```

```
                std::cerr << "Failed to publish telemetry to IoT Core: "
                          << error.message() << std::endl;
            } else {
                std::cout << "Telemetry sent to cloud: device_id=" << cmd.device_id
                          << ", status=" << status << std::endl;
            }
        }
    }
}

class CloudCommandCallback : public gg::ipc::IoTCoreTopicCallback {
    void operator()(
        std::string_view topic,
        gg::Object payload,
        gg::ipc::Subscription& handle
    ) override {
        (void) topic;
        (void) handle;

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

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

        std::cout << "Received command from IoT Core: " << json_str << std::endl;

        // Simple JSON parsing for demo.
        std::string command;
        std::string device_id = "device1";  // Default

        // Extract command.
        size_t cmd_pos = json_str.find("\"command\":");
        if (cmd_pos != std::string::npos) {
            size_t start = json_str.find("\"", cmd_pos + 10) + 1;
            size_t end = json_str.find("\"", start);
            if (end != std::string::npos) {
                command = json_str.substr(start, end - start);
            }
```

```
        }

        // Extract device_id if present.
        size_t dev_pos = json_str.find("\"device_id\":");
        if (dev_pos != std::string::npos) {
            size_t start = json_str.find("\"", dev_pos + 12) + 1;
            size_t end = json_str.find("\"", start);
            if (end != std::string::npos) {
                device_id = json_str.substr(start, end - start);
            }
        }

        if (!command.empty()) {
            std::lock_guard<std::mutex> lock(command_mutex);
            pending_command = {device_id, command};
            has_pending_command = true;
            command_cv.notify_one();

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

int main() {
    // Disable stdout buffering for real-time logging in systemd/Greengrass.
    std::cout.setf(std::ios::unitbuf);

    std::cout << "Device Controller component starting..." << std::endl;

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

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

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

    // Start telemetry publisher thread.
    std::thread telemetry_thread(telemetry_publisher_thread);
    telemetry_thread.detach();
```

```
    // Subscribe to commands from IoT Core.
    static CloudCommandCallback handler;
    error = client.subscribe_to_iot_core(COMMAND_TOPIC, handler);

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

    std::cout << "Subscribed to " << COMMAND_TOPIC << std::endl;
    std::cout << "Waiting for commands from IoT Core..." << std::endl;

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

    return 0;
}
```

2. Erstellen und verpacken Sie die Komponente

Einige Sprachen müssen vor der Bereitstellung erstellt oder gepackt werden.

Python

Python erfordert keine Kompilierung. Die Komponente kann die .py-Datei direkt verwenden.

Java

Um eine ausführbare JAR zu erstellen, in der alle Abhängigkeiten gebündelt sind:

1. Erstellen Sie eine `pom.xml` Datei in Ihrem Projektverzeichnis:

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

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

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

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

    <build>
        <plugins>
            <!-- Maven Shade Plugin - creates a standalone JAR with all
 dependencies included for Greengrass deployment -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>3.2.4</version>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            <transformers>
                                <!-- Set the main class for the executable JAR
 -->
                                <transformer
 implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
                                    <mainClass>DeviceController</mainClass>
```

```
                                </transformer>
                            </transformers>
                            <filters>
                                <!-- Exclude signature files to avoid security
 exceptions -->
                                <filter>
                                    <artifact>*:*</artifact>
                                    <excludes>
                                        <exclude>META-INF/*.SF</exclude>
                                        <exclude>META-INF/*.DSA</exclude>
                                        <exclude>META-INF/*.RSA</exclude>
                                    </excludes>
                                </filter>
                            </filters>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

2. Erstellen Sie das JAR:

```
mvn clean package
```

Dies erstellt, `target/device-controller-1.0.0.jar` wobei alle Abhängigkeiten enthalten sind.

3. Laden Sie das JAR zur Bereitstellung in Ihren S3-Bucket hoch.

JavaScript

Um Ihre Komponente Node.js mit allen Abhängigkeiten zu verpacken:

1. Erstellen Sie eine `package.json` Datei:

```
{
  "name": "device-controller",
  "version": "1.0.0",
  "description": "Device controller component for Greengrass v2",
  "main": "DeviceController.js",
  "dependencies": {
```

```
    "aws-iot-device-sdk-v2": "^1.21.0"
  },
  "engines": {
    "node": ">=14.0.0"
  }
}
```

2. Installieren Sie Abhängigkeiten auf Ihrem Entwicklungscomputer:

```
npm install
```

   Dadurch wird ein `node_modules` Ordner erstellt, der die AWS AWS IoT Device SDK Version v2 enthält.

3. Package für die Bereitstellung:

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

4. Laden Sie die Zip-Datei zur Bereitstellung in Ihren S3-Bucket hoch.

> **Note**
>
> Auf dem Greengrass-Gerät muss Node.js Runtime installiert sein (Version 14 oder höher). Sie müssen es nicht `npm install` auf dem Greengrass-Core-Gerät ausführen, da das Komponentenartefakt alle Abhängigkeiten im gebündelten `node_modules` Ordner enthält.

C

Voraussetzungen:

Um das SDK und die Komponente zu erstellen, benötigen Sie die folgenden Build-Abhängigkeiten:

- GCC oder Clang
- CMake (mindestens Version 3.22)
- Marke oder Ninja

Installieren Sie die Build-Abhängigkeiten:

Auf Ubuntu/Debian:

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

Auf Amazon Linux:

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

Erstellen Sie eine Datei CMake Lists.txt für Ihre Komponente:

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

set(CMAKE_C_STANDARD 11)

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

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

Schritte zum Erstellen:

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

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

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

C++

Voraussetzungen:

Um das SDK und die Komponente zu erstellen, benötigen Sie die folgenden Build-Abhängigkeiten:

- GCC oder Clang mit C++20-Unterstützung
- CMake (mindestens Version 3.22)
- Marke oder Ninja

Installieren Sie die Build-Abhängigkeiten:

Auf Ubuntu/Debian:

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

Auf Amazon Linux:

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

Erstellen Sie eine Datei CMake Lists.txt für Ihre Komponente:

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

set(CMAKE_CXX_STANDARD 20)

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

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

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

Schritte zum Erstellen:

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

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

# The binary 'device_controller' will be in ./build/
```

```
# Upload this binary to S3 for deployment
```

3. Rezept für Komponenten

Aktualisieren Sie das Array „Ressourcen“ mit den aktuellen Themen, die Ihre Komponente verwendet.

Python

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.DeviceController",
  "ComponentVersion": "1.0.0",
  "ComponentType": "aws.greengrass.generic",
  "ComponentDescription": "Receives commands from IoT Core and sends telemetry back
 to cloud",
  "ComponentPublisher": "[Your Company]",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.DeviceController:mqttproxy:1": {
            "policyDescription": "Allows access to subscribe to IoT Core topics",
            "operations": [
              "aws.greengrass#SubscribeToIoTCore"
            ],
            "resources": [
              "commands/device1"
            ]
          },
          "com.example.DeviceController:mqttproxy:2": {
            "policyDescription": "Allows access to publish to IoT Core topics",
            "operations": [
              "aws.greengrass#PublishToIoTCore"
            ],
            "resources": [
              "telemetry/device1"
            ]
          }
        }
      }
    }
```

```
  },
  "ComponentDependencies": {
    "aws.greengrass.TokenExchangeService": {
      "VersionRequirement": ">=2.0.0",
      "DependencyType": "HARD"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "runtime": "*"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/device_controller.py"
      },
      "Artifacts": [
        {
          "Uri": "s3://YOUR-BUCKET/artifacts/com.example.DeviceController/1.0.0/
device_controller.py"
        }
      ]
    }
  ]
}
```

Java

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.DeviceController",
  "ComponentVersion": "1.0.0",
  "ComponentType": "aws.greengrass.generic",
  "ComponentDescription": "Receives commands from IoT Core and sends telemetry back
 to cloud",
  "ComponentPublisher": "[Your Company]",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.DeviceController:mqttproxy:1": {
            "policyDescription": "Allows access to subscribe to IoT Core topics",
            "operations": [
```

```
              "aws.greengrass#SubscribeToIoTCore"
            ],
            "resources": [
              "commands/device1"
            ]
          },
          "com.example.DeviceController:mqttproxy:2": {
            "policyDescription": "Allows access to publish to IoT Core topics",
            "operations": [
              "aws.greengrass#PublishToIoTCore"
            ],
            "resources": [
              "telemetry/device1"
            ]
          }
        }
      }
    },
  },
  "ComponentDependencies": {
    "aws.greengrass.TokenExchangeService": {
      "VersionRequirement": ">=2.0.0",
      "DependencyType": "HARD"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "runtime": "*"
      },
      "Lifecycle": {
        "run": "java -jar {artifacts:path}/DeviceController.jar"
      },
      "Artifacts": [
        {
          "Uri": "s3://YOUR-BUCKET/artifacts/com.example.DeviceController/1.0.0/
DeviceController.jar"
        }
      ]
    }
  ]
}
```

JavaScript

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.DeviceController",
  "ComponentVersion": "1.0.0",
  "ComponentType": "aws.greengrass.generic",
  "ComponentDescription": "Receives commands from IoT Core and sends telemetry back
 to cloud",
  "ComponentPublisher": "[Your Company]",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.DeviceController:mqttproxy:1": {
            "policyDescription": "Allows access to subscribe to command topics from
 IoT Core",
            "operations": [
              "aws.greengrass#SubscribeToIoTCore"
            ],
            "resources": [
              "commands/device1"
            ]
          },
          "com.example.DeviceController:mqttproxy:2": {
            "policyDescription": "Allows access to publish telemetry to IoT Core",
            "operations": [
              "aws.greengrass#PublishToIoTCore"
            ],
            "resources": [
              "telemetry/*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "runtime": "*"
      },
      "Lifecycle": {
```

```
        "run": "cd {artifacts:decompressedPath}/DeviceController && node
 DeviceController.js"
      },
      "Artifacts": [
        {
          "Uri": "s3://YOUR-BUCKET/artifacts/com.example.DeviceController/1.0.0/
DeviceController.zip",
          "Unarchive": "ZIP"
        }
      ]
    }
  ]
}
```

C/C++

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.DeviceController",
  "ComponentVersion": "1.0.0",
  "ComponentType": "aws.greengrass.generic",
  "ComponentDescription": "Receives commands from IoT Core and sends telemetry back
 to cloud",
  "ComponentPublisher": "[Your Company]",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.DeviceController:mqttproxy:1": {
            "policyDescription": "Allows access to subscribe to IoT Core topics",
            "operations": ["aws.greengrass#SubscribeToIoTCore"],
            "resources": ["commands/device1"]
          },
          "com.example.DeviceController:mqttproxy:2": {
            "policyDescription": "Allows access to publish to IoT Core topics",
            "operations": ["aws.greengrass#PublishToIoTCore"],
            "resources": ["telemetry/device1"]
          }
        }
      }
    }
  },
  "Manifests": [
```

```
    {
      "Platform": {
        "os": "linux",
        "runtime": "*"
      },
      "Lifecycle": {
        "run": "{artifacts:path}/device_controller"
      },
      "Artifacts": [
        {
          "Uri": "s3://YOUR-BUCKET/artifacts/com.example.DeviceController/1.0.0/
device_controller"
        }
      ]
    }
  ]
}
```

Aktualisieren Sie Greengrass V1-Kerngeräte auf Greengrass V2

Nachdem Sie sich vergewissert haben, dass Ihre Anwendungen und Komponenten auf einem AWS IoT Greengrass V2 Kerngerät funktionieren, können Sie die AWS IoT Greengrass Core-Software v2.x auf Ihren Geräten installieren, auf denen derzeit Version 1.x ausgeführt wird, z. B. auf Produktionsgeräten. Stellen Sie dann Greengrass V2-Komponenten bereit, um Ihre Greengrass-Anwendungen auf den Geräten auszuführen.

Um eine Geräteflotte von V1 auf V2 aufzurüsten, führen Sie diese Schritte für jedes zu aktualisierende Gerät durch. Sie können Dinggruppen verwenden, um V2-Komponenten auf einer Flotte von Kerngeräten bereitzustellen.

> **Tip**
>
> Wir empfehlen, dass Sie ein Skript erstellen, um den Upgrade-Prozess für eine Flotte von Geräten zu automatisieren. Wenn Sie Ihre Flotte verwalten, können Sie Systems Manager verwenden, um dieses Skript auf jedem Gerät auszuführen, um Ihre Flotte von V1 auf V2 zu aktualisieren. AWS Systems Manager
> Sie können sich an Ihren AWS Enterprise Support-Mitarbeiter wenden, wenn Sie Fragen dazu haben, wie Sie den Upgrade-Prozess am besten automatisieren können.

Schritt 1: Installieren Sie die 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 das Upgrade in weniger Schritten durchzuführen, können Sie die v1.x-Software deinstallieren, bevor Sie die v2.x-Software installieren.

- Upgrade mit minimaler Ausfallzeit

  Für ein Upgrade mit minimalen Ausfallzeiten können Sie beide Versionen der AWS IoT Greengrass Core-Software gleichzeitig installieren. Nachdem Sie die AWS IoT Greengrass Core-Software v2.x installiert und sich vergewissert haben, dass Ihre V2-Anwendungen ordnungsgemäß funktionieren, deinstallieren Sie die AWS IoT Greengrass Core-Software v1.x. Bevor Sie sich für diese Option entscheiden, sollten Sie den zusätzlichen Arbeitsspeicher berücksichtigen, der erforderlich ist, um beide Versionen der AWS IoT Greengrass Core-Software gleichzeitig auszuführen.

Deinstallieren Sie AWS IoT Greengrass Core v1.x, bevor Sie v2.x installieren

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

Um die Core-Software v1.x zu deinstallieren AWS IoT Greengrass

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

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

   ```
   sudo systemctl stop greengrass
   ```

   b. Warten Sie, bis der Dienst beendet wird. Sie können den `list` Befehl verwenden, um den Status des Dienstes zu überprüfen.

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

   c. Deaktivieren Sie den Dienst.

```
sudo systemctl disable greengrass
```

d. Entfernen Sie den Dienst.

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

2. Wenn die AWS IoT Greengrass Core-Software v1.x nicht als Dienst läuft, verwenden Sie den folgenden Befehl, um den Daemon zu beenden. *greengrass-root*Ersetzen Sie es 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 anschließend den Schreibordner zu entfernen.

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

Für Greengrass Nucleus: Sie können die Installationsanweisungen für verwenden AWS IoT Greengrass V2, um den Greengrass Nucleus auf Ihrem Gerät zu installieren.

Für Greengrass Nucleus Lite: Sie können die Installationsanweisungen für Greengrass Nucleus Lite verwenden, um Greengrass Nucleus Lite zu installieren.

> **Tip**
>
> Um die Identität eines Core-Geräts bei der Migration von V1 auf V2 wiederzuverwenden, folgen Sie den Anweisungen zur Installation der AWS IoT Greengrass Core-Software mit manueller Bereitstellung. Entfernen Sie zuerst die V1-Kernsoftware vom Gerät und

verwenden Sie dann das AWS IoT Ding und das Zertifikat des V1-Kerngeräts erneut. Aktualisieren Sie die AWS IoT Richtlinien des Zertifikats, um die Berechtigungen zu gewähren, die für die v2.x-Software erforderlich sind.

Installieren Sie die 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 bereits die AWS IoT Greengrass Core-Software v1.x ausgeführt wird, sollten Sie Folgendes beachten:

- Der AWS IoT Dingname für Ihr V2-Core-Gerät muss eindeutig sein. Verwenden Sie nicht denselben Namen 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 so, dass er einen anderen Port als 8088 verwendet. 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-Systemdienst umzubenennen. Wenn Sie Greengrass als Systemdienst ausführen, müssen Sie einen der folgenden Schritte ausführen, um Konflikte zwischen Systemdienstnamen zu vermeiden:
  - Benennen Sie den Greengrass-Dienst für v1.x um, bevor Sie v2.x installieren.
  - Installieren Sie die AWS IoT Greengrass Core-Software v2.x ohne Systemdienst und konfigurieren Sie die Software dann manuell als Systemdienst mit einem anderen Namen als `greengrass`.

  Um den Greengrass-Dienst für v1.x umzubenennen

  1. Beenden Sie den Dienst AWS IoT Greengrass Core Software v1.x.

     ```
     sudo systemctl stop greengrass
     ```

  2. Warten Sie, bis der Dienst beendet ist. Es kann einige Minuten dauern, bis der Dienst beendet wird. Sie können den `list-units` Befehl verwenden, um zu überprüfen, ob der Dienst beendet wurde.

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

3. Deaktivieren Sie den Dienst.

```
sudo systemctl disable greengrass
```

4. Benennen Sie den Dienst um.

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

5. Laden Sie den Dienst 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 verwenden AWS IoT Greengrass V2, um die Software auf Ihrem Gerät zu installieren.

> **Tip**
>
> Um die Identität eines Core-Geräts bei der Migration von V1 auf V2 wiederzuverwenden, folgen Sie den Anweisungen zur Installation der AWS IoT Greengrass Core-Software mit manueller Bereitstellung. Entfernen Sie zuerst die V1-Kernsoftware vom Gerät und verwenden Sie dann das AWS IoT Ding und das Zertifikat des V1-Kerngeräts erneut. Aktualisieren Sie die AWS IoT Richtlinien des Zertifikats, um die Berechtigungen zu gewähren, die für die v2.x-Software erforderlich sind.

Schritt 2: Stellen Sie AWS IoT Greengrass V2 Komponenten auf den Kerngeräten bereit

Nachdem Sie die AWS IoT Greengrass Core-Software v2.x auf Ihrem Gerät installiert haben, stellen Sie die Komponenten auf der Grundlage der von Ihnen ausgewählten Laufzeit bereit.

Für Greengrass Nucleus:

Erstellen Sie eine Bereitstellung, die die folgenden Ressourcen umfasst. Um Komponenten für eine Flotte ähnlicher Geräte bereitzustellen, erstellen Sie eine Bereitstellung für eine Dinggruppe, die diese Geräte enthält.

- Lambda-Funktionskomponenten, die Sie aus Ihren V1-Lambda-Funktionen erstellt haben. Weitere Informationen finden Sie unter AWS Lambda Funktionen ausführen.
- Wenn Sie V1-Abonnements verwenden, die Router-Komponente für ältere Abonnements.
- Wenn Sie Stream Manager verwenden, die Stream Manager-Komponente. Weitere Informationen finden Sie unter Datenströme auf Greengrass-Kerngeräten verwalten.
- Wenn Sie lokale Geheimnisse verwenden, die Secret Manager-Komponente.
- Wenn Sie V1-Konnektoren verwenden, die von AWS-bereitgestellten Konnektorkomponenten.
- Wenn Sie Docker-Container verwenden, die Docker-Anwendungsmanager-Komponente. Weitere Informationen finden Sie unter Führen Sie einen Docker-Container aus.
- Wenn Sie verbundene Geräte verwenden, unterstützen die Komponenten für Client-Geräte. Sie müssen außerdem die Unterstützung für Client-Geräte aktivieren und die Client-Geräte Ihrem Kerngerät zuordnen. Weitere Informationen finden Sie unter Interagieren Sie mit lokalen IoT-Geräten.
- Wenn Sie Geräteschatten verwenden, die Shadow-Manager-Komponente. Weitere Informationen finden Sie unter Interagiere mit Geräteschatten.
- Wenn Sie Protokolle von Greengrass-Core-Geräten auf Amazon CloudWatch Logs hochladen, die Log-Manager-Komponente. Weitere Informationen finden Sie unter AWS IoT Greengrass Protokolle überwachen.
- Wenn Sie mit integrieren AWS IoT SiteWise, folgen Sie den Anweisungen, um das V2-Kerngerät als AWS IoT SiteWise Gateway einzurichten. AWS IoT SiteWise stellt ein Installationsskript bereit, 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. AWS IoT Greengrass Komponenten auf Geräten bereitstellen

Für Greengrass Nucleus Lite:

Stellen Sie die generischen Komponenten, die Sie in Schritt 2 des Migrationsleitfadens erstellt haben, auf Ihren Greengrass Nucleus Lite-Geräten bereit:

1. Erstellen Sie Ihre Komponenten mithilfe der von Ihnen erstellten Komponentenrezepte
2. Erstellen Sie eine Bereitstellung für Ihre Greengrass Nucleus Lite-Geräte, die Ihre generischen Komponenten enthält
3. Stellen Sie sicher, dass Ihre Komponenten ordnungsgemäß funktionieren

Tutorial: Erste Schritte mit AWS IoT Greengrass V2

Sie können dieses Tutorial für die ersten Schritte abschließen, um die grundlegenden Funktionen von zu erlernen AWS 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-Core-Gerät. Komponenten sind Softwaremodule, die auf Greengrass-Kerngeräten ausgeführt werden.
3. Laden Sie diese Komponente AWS IoT Greengrass V2 in den AWS Cloud hoch.
4. Stellen Sie diese Komponente AWS Cloud auf Ihrem Greengrass-Core-Gerät bereit.

> **Note**
>
> In diesem Tutorial wird beschrieben, wie Sie eine Entwicklungsumgebung einrichten und die Funktionen von AWS IoT Greengrass kennenlernen. Weitere Informationen zum Einrichten und Konfigurieren von Produktionsgeräten finden Sie im Folgenden:
>
> - 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 aufzuwenden.

Themen

- Voraussetzungen
- Schritt 1: Richten Sie ein AWS Konto ein
- Schritt 2: Richten Sie Ihre Umgebung ein
- Schritt 3: Installieren Sie die AWS IoT Greengrass Core-Software
- Schritt 4: Entwickeln und testen Sie eine Komponente auf Ihrem Gerät
- Schritt 5: Erstellen Sie Ihre Komponente im AWS IoT Greengrass Service
- Schritt 6: Stellen Sie Ihre Komponente bereit
- Nächste Schritte

Voraussetzungen

Für dieses Tutorial benötigen Sie Folgendes:

- Ein AWS-Konto. Falls Sie noch keines haben, beachten Sie die Informationen unter Schritt 1: Richten Sie ein AWS Konto ein.
- Die Verwendung einer AWS-Region, die 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.
- Ein AWS Identity and Access Management (IAM-) Benutzer mit Administratorrechten.
- Ein Gerät, das als Greengrass-Core-Gerät eingerichtet werden kann, z. B. ein Raspberry Pi mit Raspberry Pi OS (früher Raspbian genannt) oder ein Windows 10-Gerät. Sie müssen über Administratorrechte für dieses Gerät verfügen oder Administratorrechte erwerben können, z. B. über. `sudo` Dieses Gerät muss über eine Internetverbindung verfügen.

  Sie können sich auch dafür entscheiden, ein anderes Gerät zu verwenden, das die Anforderungen für die Installation und Ausführung der AWS IoT Greengrass Core-Software erfüllt.

  Wenn Ihr Entwicklungscomputer diese Anforderungen erfüllt, können Sie ihn in diesem Tutorial als Ihr 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 außerdem der Python Launcher für Windows für alle Benutzer installiert sein.

  > **Important**
  >
  > In Windows wird Python standardmäßig nicht für alle Benutzer installiert. Wenn Sie Python installieren, müssen Sie die Installation anpassen, um sie so zu konfigurieren, dass die AWS IoT Greengrass Core-Software Python-Skripts ausführt. Wenn Sie beispielsweise den grafischen Python-Installer verwenden, gehen Sie wie folgt vor:
  >
  > 1. Wählen Sie Launcher für alle Benutzer installieren (empfohlen) aus.
  > 2. Wählen Sie aus.Customize installation.
  > 3. Wählen Sie aus.Next.
  > 4. Select Install for all users.
  > 5. Select Add Python to environment variables.

6. Wählen Sie Installieren aus.

Weitere Informationen finden Sie unter Using Python on Windows in der Python 3-Dokumentation.

- AWS Command Line Interface (AWS CLI) wurde auf Ihrem Entwicklungscomputer und auf Ihrem Gerät mit Anmeldeinformationen installiert und konfiguriert. Stellen Sie sicher, dass Sie dasselbe AWS-Region für die Konfiguration von AWS CLI auf Ihrem Entwicklungscomputer und auf Ihrem Gerät verwenden. Für die AWS CLI Verwendung AWS IoT Greengrass V2 mit benötigen Sie eine der folgenden Versionen oder höher:
  - Mindestversion AWS CLI V1: v1.18.197
  - Minimale AWS CLI V2-Version: v2.1.11

  Tip

  Sie können den folgenden Befehl ausführen, um die Version von zu überprüfen AWS CLI , die Sie haben.

  ```
  aws --version
  ```

  Weitere Informationen finden Sie unter Installation, Aktualisierung und Deinstallation von AWS CLI und Konfiguration von 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 Installation, Aktualisierung und Deinstallation von 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/die Anmeldung.
2. Folgen Sie den Online-Anweisungen.

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

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

AWS sendet Ihnen nach Abschluss des Anmeldevorgangs eine Bestätigungs-E-Mail. Sie können Ihre aktuellen Kontoaktivitäten jederzeit einsehen und Ihr Konto verwalten, indem Sie zu https://aws.amazon.com/gehen und Mein Konto auswählen.

Erstellen eines Benutzers mit Administratorzugriff

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

Sichern Sie Ihre Root-Benutzer des AWS-Kontos

1. Melden Sie sich AWS-Managementkonsoleals Kontoinhaber an, indem Sie Root-Benutzer auswählen und Ihre AWS-Konto E-Mail-Adresse eingeben. Geben Sie auf der nächsten Seite Ihr Passwort ein.

   Hilfe bei der Anmeldung mit dem Root-Benutzer finden Sie unter Anmelden als Root-Benutzer im AWS-Anmeldung Benutzerhandbuch zu.
2. Aktivieren Sie die Multi-Faktor-Authentifizierung (MFA) für den Root-Benutzer.

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

Erstellen eines Benutzers mit Administratorzugriff

1. Aktivieren Sie das IAM Identity Center.

   Anweisungen finden Sie unter Aktivieren AWS IAM Identity Center im AWS IAM Identity Center Benutzerhandbuch.

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

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

Anmelden als Administratorbenutzer

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

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

Weiteren Benutzern Zugriff zuweisen

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

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

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

   Eine genaue Anleitung finden Sie unter Gruppen hinzufügen im AWS IAM Identity Center Benutzerhandbuch.

Schritt 2: Richten Sie Ihre Umgebung ein

> **Note**
>
> Diese Schritte gelten nicht für Nucleus Lite.

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 (Raspberry Pi) ein

Bei diesen Schritten wird davon ausgegangen, dass Sie einen Raspberry Pi mit Raspberry Pi OS verwenden. Wenn Sie ein anderes Gerät oder Betriebssystem verwenden, schlagen Sie in der entsprechenden Dokumentation für Ihr Gerät nach.

Um einen Raspberry Pi einzurichten für AWS IoT Greengrass V2

1. Aktivieren Sie SSH auf Ihrem Raspberry Pi, um eine Remoteverbindung 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 Connect zu Ihrem Raspberry Pi her.

   Führen Sie auf Ihrem Entwicklungscomputer den folgenden Befehl aus. *username*Ersetzen Sie ihn durch den Namen des Benutzers, der sich anmelden soll, und *pi-ip-address* ersetzen Sie ihn 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 Sie haben möglicherweise keine Verbindung zu Ihrem Raspberry Pi hergestellt, können `ssh` aber keine Verbindung herstellen. Um eine Verbindung zu Ihrem Raspberry Pi herzustellen, können Sie PuTTY

installieren und konfigurieren, einen kostenlosen Open-Source-SSH-Client. Konsultieren Sie die PuTTY-Dokumentation, um eine Verbindung zu Ihrem Raspberry Pi herzustellen.

4. Installieren Sie die Java-Runtime, die für die Ausführung der AWS IoT Greengrass Core-Software erforderlich ist. 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 druckt die Version von Java, 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: Stellen Sie die Kernel-Parameter auf einem Raspberry Pi ein

Wenn es sich bei Ihrem Gerät um einen Raspberry Pi handelt, können Sie die folgenden Schritte ausführen, um die Linux-Kernel-Parameter anzuzeigen und zu aktualisieren:

1. Öffnen Sie die `/boot/cmdline.txt` Datei. Diese Datei spezifiziert die Linux-Kernel-Parameter, die beim Booten des Raspberry Pi angewendet werden sollen.

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

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

2. Stellen Sie sicher, dass die `/boot/cmdline.txt` Datei die folgenden Kernel-Parameter enthält. Der `systemd.unified_cgroup_hierarchy=0` Parameter gibt an, dass cgroups v1 statt 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 diese Parameter 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 zu übernehmen.

```
sudo reboot
```

Richten Sie ein Linux-Gerät ein (anderes)

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 Argumente des Installers.

```
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

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. *`password`*Durch ein sicheres Passwort ersetzen.

```
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 PsExec Programm 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. *password*Ersetzen Sie es durch das zuvor festgelegte Benutzerkennwort.

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

   Wenn das Symbol PsExec License Agreementöffnet, wählen Acceptum der Lizenz zuzustimmen und den Befehl auszuführen.

   > **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

Schritt 3: Installieren Sie die AWS IoT Greengrass Core-Software

> **Tip**
>
> Wir empfehlen Ihnen, das AWS IoT Greengrass AI Agents Context Pack auszuprobieren, um AWS IoT Greengrass schnell einzurichten und damit zu experimentieren. Das Agent Context Pack ermöglicht es KI-Agenten, Greengrass Nucleus und Nucleus Lite einzurichten, Komponenten bereitzustellen und häufig auftretende Probleme zu beheben.

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

- Installiert die Greengrass Nucleus-Komponente. Der Nucleus ist eine obligatorische Komponente und die Mindestanforderung, um die AWS IoT Greengrass Core-Software auf einem Gerät auszuführen. Weitere Informationen finden Sie unter Greengrass-Kernkomponente.

- Registriert Ihr Gerät als Objekt AWS IoT und lädt ein digitales Zertifikat herunter, mit dem Ihr Gerät eine Verbindung herstellen AWS kann. Weitere Informationen finden Sie unter Geräteauthentifizierung und Autorisierung für AWS IoT Greengrass.
- Fügt das AWS IoT Ding des Geräts einer Dinggruppe hinzu, bei der es sich um eine Gruppe oder eine Flotte von AWS 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 Sie AWS IoT im AWS IoT Core Entwicklerhandbuch unter Geräte verwalten mit.
- Erstellt die IAM-Rolle, die es Ihrem Greengrass-Core-Gerät ermöglicht, mit AWS Diensten zu interagieren. Standardmäßig ermöglicht diese Rolle Ihrem Gerät, mit Amazon Logs zu interagieren AWS IoT und Protokolle an Amazon CloudWatch Logs zu senden. Weitere Informationen finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.
- Installiert die AWS 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 Sie die 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 Kerngerät einrichten aus.
3. Geben Sie unter Schritt 1: Ein Greengrass-Core-Gerät registrieren unter Core-Gerätename den Namen des AWS IoT Geräts für Ihr Greengrass-Core-Gerät ein. Wenn das Ding nicht existiert, erstellt es das Installationsprogramm.
4. Wählen Sie unter Schritt 2: Zu einer Dinggruppe hinzufügen, um eine kontinuierliche Bereitstellung anzuwenden, für Dinggruppe die AWS IoT Dinggruppe aus, zu der Sie Ihr Kerngerät hinzufügen möchten.

   - Wenn Sie Neuen Gruppennamen eingeben auswählen, geben Sie im Feld Name der Dinggruppe den Namen der neuen Gruppe ein, die erstellt werden soll. Das Installationsprogramm erstellt die neue Gruppe für Sie.
   - Wenn Sie Vorhandene Gruppe auswählen auswählen wählen, wählen Sie im Feld Name der Dinggruppe die bestehende Gruppe aus, die Sie verwenden möchten.
   - Wenn Sie Keine Gruppe auswählen, fügt das Installationsprogramm das Kerngerät keiner Dinggruppe hinzu.

5. Führen Sie unter Schritt 3: Installation der Greengrass Core-Software die folgenden Schritte aus.

   Nucleus classic

   1. Wählen Sie Nucleus classic als Software-Runtime für Ihr Kerngerät.
   2. Wählen Sie das Betriebssystem Ihres Kerngeräts: Linux oder Windows.
   3. Geben Sie Ihre AWS Anmeldeinformationen für das Gerät ein, damit das Installationsprogramm die AWS IoT und IAM-Ressourcen für Ihr Kerngerät bereitstellen kann. Um die Sicherheit zu erhöhen, empfehlen wir Ihnen, temporäre Anmeldeinformationen für eine IAM-Rolle zu erwerben, die nur die für die Bereitstellung erforderlichen Mindestberechtigungen gewährt. 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 die Anmeldeinformationen abzurufen und sie dem AWS IoT Greengrass Core-Software-Installationsprogramm zur Verfügung zu stellen:

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

        a. Geben Sie die Zugriffsschlüssel-ID, den geheimen Zugriffsschlüssel und das Sitzungstoken aus dem IAM Identity Center ein. Weitere Informationen finden Sie unter Manuelles Aktualisieren von Anmeldeinformationen im Abschnitt Temporäre Anmeldeinformationen abrufen und aktualisieren 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 IAM-Rolle an, die Sie annehmen. Weitere Informationen zum Abrufen dieser Anmeldeinformationen finden Sie unter Temporäre Sicherheitsanmeldeinformationen anfordern 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 zur Verfügung gestellt werden soll, finden Sie unter. Minimale IAM-Richtlinie für das Installationsprogramm zur Bereitstellung von Ressourcen Weitere Informationen zum Abrufen langfristiger Anmeldeinformationen finden Sie unter Verwaltung 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 für die 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, sodass sie nicht mehr gültig sind. Weitere Informationen finden Sie im AWS Identity and Access Management Benutzerhandbuch unter Aktualisieren von Zugriffsschlüsseln.

4. Führen Sie unter Das Installationsprogramm ausführen die folgenden Schritte aus.

a. Wählen Sie unter Das Installationsprogramm herunterladen die Option Kopieren aus und führen Sie den kopierten Befehl auf Ihrem Hauptgerät aus. Mit diesem Befehl wird die neueste Version der AWS IoT Greengrass Core-Software heruntergeladen und auf Ihrem Gerät entpackt.

b. Wählen Sie unter Das Installationsprogramm ausführen die Option Kopieren aus und führen Sie den kopierten Befehl auf Ihrem Core-Gerät aus. Dieser Befehl verwendet die Namen der AWS IoT Dinge und der Dinggruppen, die Sie zuvor angegeben haben, um das AWS IoT Greengrass Core-Softwareinstallationsprogramm auszuführen und AWS Ressourcen für Ihr Kerngerät einzurichten.

Dieser Befehl führt außerdem Folgendes aus:

- 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.

- Stellen Sie die AWS IoT Greengrass CLI-Komponente bereit, ein Befehlszeilentool, mit dem Sie benutzerdefinierte Greengrass-Komponenten auf dem Kerngerät entwickeln können.
- 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.

Wenn Sie diesen Befehl ausführen, sollten Sie die folgenden Meldungen sehen, die darauf hinweisen, dass die Installation 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 ein Linux-Gerät haben und es nicht über Systemd verfügt, richtet das Installationsprogramm die Software nicht als Systemdienst ein, und die Erfolgsmeldung für die Einrichtung des Nucleus als Systemdienst wird nicht angezeigt.

Nucleus lite

1. Wählen Sie Nucleus Lite als Software-Runtime für Ihr Kerngerät.
2. Wählen Sie Ihre Geräteeinrichtungsmethode aus, um Ihr Gerät für ein Greengrass-Core-Gerät bereitzustellen.

Option 1: Richten Sie ein Gerät mit Paket-Download ein (ca. 1 MB)

1. Erstelle ein AWS IoT Ding und die Rolle für Greengrass.
2. Laden Sie die ZIP-Datei herunter, die AWS IoT Ressourcen enthält, mit denen Ihr Gerät eine Verbindung herstellen muss AWS IoT:
   - Ein Zertifikat und ein privater Schlüssel, die mithilfe AWS IoT der Zertifizierungsstelle generiert wurden.
   - Eine Schemadatei, um die Greengrass-Installation für Ihr Gerät zu initiieren.
3. Laden Sie das Paket herunter, mit dem die neueste Greengrass Nucleus Lite-Laufzeit auf Ihrem Raspberry Pi installiert wird.

4. Richten Sie Ihr Gerät als AWS IoT Greengrass Core-Gerät ein und verbinden Sie es mit: AWS IoT

    a. a. Übertragen Sie das Greengrass-Paket und das Verbindungskit mit einem USB-Stick, SCP/FTP oder SD-Karten auf Ihr Gerät.

    b. b. Entpacken Sie die Datei greengrass-package.zip im GreengrassInstaller Verzeichnis/auf dem Gerät.

    c. c. Entpacken Sie die ZIP-Datei des Connection Kits im Verzeichnis /auf dem Gerät.

    d. d. Führen Sie zur Installation den bereitgestellten Befehl auf dem Gerät aus AWS IoT Greengrass

5. Wählen Sie dann „Kerngeräte anzeigen“.

Option 2: Richten Sie ein Gerät mit einem vorkonfigurierten Beispiel-Image-Download für die gesamte Festplatte ein (ca. 100 MB)

1. Erstelle ein AWS IoT Ding und die Rolle für Greengrass.
2. Laden Sie die ZIP-Datei herunter, die AWS IoT Ressourcen enthält, mit denen Ihr Gerät eine Verbindung herstellen muss AWS IoT:

    - Ein Zertifikat und ein privater Schlüssel, die mithilfe AWS IoT der Zertifizierungsstelle generiert wurden.
    - Eine Schemadatei, um die Greengrass-Installation für Ihr Gerät zu initiieren.

3. Laden Sie das vorkonfigurierte Beispiel-Image für die gesamte Festplatte herunter, das Greengrass und das Betriebssystem enthält.

    a. Folgen Sie der Readme-Datei, die zusammen mit dem Image heruntergeladen wurde, um das Verbindungskit zu übertragen und das Image auf Ihr Gerät zu flashen.

    b. Um die Greengrass-Installation zu starten, schalten Sie das Gerät ein und starten Sie es von dem Flash-Image aus

4. Wählen Sie dann „Kerngeräte anzeigen“.

Option 3: Richten Sie ein Gerät mit Ihrem eigenen Build ein

1. Erstelle ein AWS IoT Ding und die Rolle für Greengrass.

2. Laden Sie die ZIP-Datei herunter, die AWS IoT Ressourcen enthält, mit denen Ihr Gerät eine Verbindung herstellen muss AWS IoT:

   - Ein Zertifikat und ein privater Schlüssel, die mithilfe AWS IoT der Zertifizierungsstelle generiert wurden.
   - Eine Schemadatei, um die Greengrass-Installation für Ihr Gerät zu initiieren.

3. Um Ihr eigenes Image mit Yocto aus dem Quellcode anzupassen und zu erstellen und anschließend das Verbindungskit zur Installation von Nucleus Lite zu verwenden, folgen Sie den Anweisungen unter. GitHub

   - Wählen Sie dann „Kerngeräte anzeigen“.

Installieren Sie die AWS IoT Greengrass Core-Software (CLI)

> **Note**
>
> Diese Schritte gelten nicht für Nucleus Lite.

Um die AWS IoT Greengrass Core-Software zu installieren und zu konfigurieren

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

   Linux or Unix

   ```
   cd ~
   ```

   Windows Command Prompt (CMD)

   ```
   cd %USERPROFILE%
   ```

   PowerShell

   ```
   cd ~
   ```

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

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. *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. 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 Kerngerät für den Betrieb benötigt.
- 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.

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

Ersetzen Sie die Argumentwerte in Ihrem Befehl wie folgt.

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

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

c. *region*. Das, AWS-Region in dem Ressourcen gefunden oder erstellt werden sollen.

d. *MyGreengrassCore*. Der Name der AWS IoT Sache für Ihr Greengrass-Core-Gerät. Wenn das Ding nicht existiert, erstellt es das Installationsprogramm. Das Installationsprogramm lädt die Zertifikate herunter, um sich als das AWS IoT Ding zu authentifizieren. Weitere Informationen finden Sie unter Geräteauthentifizierung und Autorisierung für AWS IoT Greengrass.

   > **Note**
   >
   > Der Name der Sache darf keine Doppelpunkte (`:`) enthalten.

e. *MyGreengrassCoreGroup*. Der Name der AWS IoT Dinggruppe für Ihr Greengrass-Core-Gerät. Wenn die Dinggruppe nicht existiert, erstellt das Installationsprogramm sie und fügt

ihr das Ding hinzu. Wenn die Dinggruppe vorhanden ist und über eine aktive Bereitstellung verfügt, lädt das Kerngerät die in der Bereitstellung angegebene Software herunter und führt sie aus.

> **Note**
>
> Der Name der Dinggruppe darf keine Doppelpunkte (`:`) enthalten.

f. *`GreengrassV2IoTThingPolicy`*. Der Name der AWS IoT Richtlinie, die es den Greengrass-Core-Geräten ermöglicht, mit AWS IoT und AWS IoT Greengrass zu kommunizieren. Wenn die AWS IoT Richtlinie nicht existiert, erstellt das Installationsprogramm eine zulässige 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 V2 Kerngeräte.

g. *`GreengrassV2TokenExchangeRole`*. Der Name der IAM-Rolle, die es dem Greengrass-Core-Gerät ermöglicht, temporäre AWS Anmeldeinformationen abzurufen. Wenn die Rolle nicht existiert, erstellt das Installationsprogramm sie und erstellt und fügt eine Richtlinie mit dem Namen hinzu. *`GreengrassV2TokenExchangeRole`*`Access` Weitere Informationen finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

h. *`GreengrassCoreTokenExchangeRoleAlias`*. Der Alias für die IAM-Rolle, der es dem Greengrass-Core-Gerät ermöglicht, später temporäre Anmeldeinformationen abzurufen. Wenn der Rollenalias nicht existiert, erstellt ihn das Installationsprogramm 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 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 diesen Befehl ausführen, sollten Sie die folgenden Meldungen sehen, die darauf hinweisen, dass die Installation 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 ein Linux-Gerät haben und es nicht über Systemd verfügt, richtet das Installationsprogramm die Software nicht als Systemdienst ein, und die Erfolgsmeldung für die Einrichtung des Nucleus als Systemdienst wird nicht angezeigt.

(Optional) Führen Sie die Greengrass-Software aus (Linux)

> **Note**
>
> Diese Schritte gelten nicht für Nucleus Lite.

Wenn Sie die Software als Systemdienst 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 Systemdienst eingerichtet hat, suchen Sie in der Ausgabe des Installationsprogramms nach der folgenden Zeile.

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

Wenn Sie diese Meldung nicht sehen, gehen Sie wie folgt vor, 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 druckt die folgende Meldung, wenn sie erfolgreich gestartet wurde.

```
Launched Nucleus successfully.
```

2. Sie müssen die aktuelle Befehlsshell geöffnet lassen, damit die AWS IoT Greengrass Core-Software weiter ausgeführt werden kann. 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. *username*Ersetzen Sie es durch den Namen des Benutzers, der sich anmelden soll, und *pi-ip-address* ersetzen Sie es durch die IP-Adresse des Geräts.

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

Weitere Informationen zur Interaktion mit dem Greengrass-Systemdienst finden Sie unterDen Greengrass Nucleus als Systemdienst konfigurieren.

Überprüfen Sie die Greengrass CLI-Installation auf dem Gerät

> **Note**
>
> Diese Schritte gelten nicht für Nucleus Lite.

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

```
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.

- *region*Ersetzen Sie es durch AWS-Region das, was Sie verwenden. Stellen Sie sicher, dass Sie dasselbe verwenden AWS-Region , das Sie zur Konfiguration AWS CLI auf Ihrem Gerät verwendet haben.
- Ersetze es *account-id* durch deine AWS-Konto ID.
- *MyGreengrassCore*Ersetzen Sie es durch den Namen Ihres Kerngeräts.

Linux, macOS, or Unix

```
aws greengrassv2 create-deployment \
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" \
  --components '{
    "aws.greengrass.Cli": {
      "componentVersion": "2.16.1"
    }
  }'
```

Windows Command Prompt (CMD)

```
aws greengrassv2 create-deployment ^
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" ^
  --components "{\"aws.greengrass.Cli\":{\"componentVersion\":\"2.16.1\"}}"
```

PowerShell

```
aws greengrassv2 create-deployment `
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" `
  --components '{\"aws.greengrass.Cli\":{\"componentVersion\":\"2.16.1\"}}'
```

> **Tip**
>
> 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.

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

Schritt 4: Entwickeln und testen Sie eine Komponente auf Ihrem Gerät

Eine Komponente ist ein Softwaremodul, das auf AWS IoT Greengrass Kerngeräten ausgeführt wird. Komponenten ermöglichen es Ihnen, komplexe Anwendungen als separate Bausteine zu erstellen und zu verwalten, die Sie von einem Greengrass-Core-Gerät auf ein anderes wiederverwenden können. Jede Komponente besteht aus einem Rezept und Artefakten.

- Rezepte

  Jede Komponente enthält eine Rezeptdatei, die ihre Metadaten definiert. Das Rezept spezifiziert auch die Konfigurationsparameter, die Komponentenabhängigkeiten, den Lebenszyklus und die Plattformkompatibilität der Komponente. Der Komponentenlebenszyklus definiert die Befehle, mit denen die Komponente installiert, ausgeführt und heruntergefahren wird. 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 umfassen, die eine Komponente verwendet. Komponenten können auch Artefakte aus Komponentenabhängigkeiten verwenden.

Mit können Sie die Greengrass-CLI verwenden AWS IoT Greengrass, um Komponenten lokal auf einem Greengrass-Core-Gerät ohne Interaktion mit der AWS Cloud zu entwickeln und zu testen. Wenn Sie Ihre lokale Komponente fertiggestellt 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 all Ihren Greengrass-Kerngeräten bereitzustellen. Weitere Informationen zu Komponenten finden Sie unterAWS IoT Greengrass Komponenten entwickeln.

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

Um eine Hello World-Komponente auf Ihrem Gerät zu entwickeln

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 zu dem 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 Plattformfähigkeit Ihrer Komponente definiert. Nehmen Sie die Komponentenversion in den Namen der Rezeptdatei auf, damit Sie erkennen können, welches Rezept welche Komponentenversion widerspiegelt. Sie können das YAML- oder das JSON-Format für Ihr Rezept wählen.

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

   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 Hauptteil. geringfügig. Patch-Nummernsystem. Version `1.0.0` steht beispielsweise für die erste Hauptversion einer Komponente. 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-Core-Gerät 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 Artefaktordnerpfad verwenden. Geben Sie den Komponentennamen und die Version an, die Sie im Rezept angeben.

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

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

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

```
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-Core-Gerät 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 Entwicklungsordner für Komponenten.

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

7. Die AWS IoT Greengrass Core-Software speichert Standarddaten 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 sehen, die dem folgenden Beispiel ähneln.

   ```
   Hello, world!
   ```

   > **Note**
   >
   > Wenn die Datei nicht existiert, 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 nicht gültig ist. Führen Sie den folgenden Befehl aus, um die AWS IoT Greengrass Kernprotokolldatei anzuzeigen. Diese Datei enthält Protokolle vom Bereitstellungsdienst 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 auf 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 die Datei `hello_world.py` in einem Texteditor und fügen Sie in Zeile 4 den folgenden Code 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 Kerngerät 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 sicherzustellen, dass die Hello World-Komponente die neue Nachricht 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 sehen, die dem folgenden Beispiel ähneln.

```
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 ein Konfigurationsupdate angeben, das definiert, wie die Konfiguration der Komponente auf dem Kerngerät geändert werden soll. Sie können angeben, welche Konfigurationswerte auf die Standardwerte zurückgesetzt werden sollen und welche neuen Konfigurationswerte auf dem Kerngerät 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 `hello-world-config-update.json` zu erstellen, die das Konfigurationsupdate enthält

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

       ```
       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 ein Konfigurationsupdate, das den Wert `friend` mit dem `Message` Parameter zusammenführt, um seinen Wert zu aktualisieren. Dieses Konfigurationsupdate spezifiziert keine Werte, die zurückgesetzt werden sollen. Sie müssen den `Message` Parameter nicht zurücksetzen, da das Merge-Update 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 sicherzustellen, dass 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 sehen, die dem folgenden Beispiel ähneln.

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

13. Wenn Sie mit dem Testen Ihrer Komponente fertig sind, entfernen Sie sie von Ihrem Kerngerät. 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 wieder auf dem Kerngerät bereitstellen können AWS IoT Greengrass. Andernfalls schlägt die Bereitstellung mit einem Versionskompatibilitätsfehler fehl, da bei der lokalen Bereitstellung eine andere Version der Komponente angegeben ist.

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 erscheint.

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 fertig und Sie können sie jetzt in den AWS IoT Greengrass Cloud-Dienst hochladen. Anschließend können Sie die Komponente auf Greengrass-Kerngeräten 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. *`123456789012`*Ersetzen Sie ihn durch Ihre AWS Konto-ID und *`region`* durch AWS-Region die, 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 Create Bucket (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 amzn-s3-demo-bucket durch den Namen des S3-Buckets.

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

```
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "Run": "py -3 -u {artifacts:path}/hello_world.py \"{configuration:/
Message}\""
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.HelloWorld/1.0.0/hello_world.py"
        }
      ]
    }
  ]
}
```

c. 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 Kerngerät 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. *123456789012*Ersetzen Sie es durch Ihre AWS-Konto ID und *region* durch AWS-Region die ID, 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 amzn-s3-demo-bucket durch den Namen des S3-Buckets.

      ```
      {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Action": [
              "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::amzn-s3-demo-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. *GreengrassV2TokenExchangeRole*Ersetzen Sie es 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 amzn-s3-demo-bucket durch den Namen des S3-Buckets.

   Linux or Unix

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

   Windows Command Prompt (CMD)

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

   PowerShell

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

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

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 amzn-s3-demo-bucket durch den Namen des S3-Buckets.

```
s3://amzn-s3-demo-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://amzn-s3-demo-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://amzn-s3-demo-bucket/artifacts/
com.example.HelloWorld/1.0.0/hello_world.py"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "Run": "py -3 -u {artifacts:path}/hello_world.py \"{configuration:/
Message}\""
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.HelloWorld/1.0.0/hello_world.py"
        }
      ]
    }
  ]
}
```

YAML

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

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://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/
hello_world.py
  - Platform:
      os: windows
    Lifecycle:
      Run: |
        py -3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/
hello_world.py
```

5. 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 lautetDEPLOYABLE.

```
{
  "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: Stellen Sie Ihre Komponente bereit

Mit AWS IoT Greengrass können Sie Komponenten für einzelne Geräte oder Gerätegruppen bereitstellen. Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass wird die Software dieser Komponente auf jedem Zielgerät installiert und ausgeführt. Sie geben an, welche Komponenten bereitgestellt werden sollen und welches Konfigurationsupdate für jede Komponente bereitgestellt werden soll. Sie können auch steuern, wie die Bereitstellung auf die Geräte verteilt wird, auf die die Bereitstellung abzielt. Weitere Informationen finden Sie unter AWS IoT Greengrass Komponenten auf Geräten bereitstellen.

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

Stellen Sie Ihre Komponente (Konsole) bereit

1. Wählen Sie im Navigationsmenü der AWS IoT Greengrass Konsole die Option Komponenten aus.
2. Wählen Sie auf der Seite Komponenten auf der Registerkarte Meine Komponenten com.example.HelloWorld.
3. Auf der com.example.HelloWorldWählen Sie auf der Seite Deploy 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 als Bereitstellungsziel die Option Kerngerät und geben Sie den Namen der AWS IoT Sache für Ihr Kerngerät an. Der Standardwert in diesem Tutorial ist*`MyGreengrassCore`*.
   c. Wählen Sie Weiter.
6. Vergewissern Sie sich auf der Seite Komponenten auswählen unter Meine Komponenten, dass com.example.HelloWorldDie Komponente ist ausgewählt, und klicken Sie auf Weiter.
7. Wählen Sie auf der Seite Komponenten konfigurieren com.example.HelloWorld, und 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"
      }
      ```

      Mit diesem Konfigurationsupdate wird der `Message` Parameter Hello World `universe` für das Gerät in dieser Bereitstellung auf festgelegt.
   c. Wählen Sie Bestätigen aus.
   d. Wählen Sie Weiter.
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. Stellen Sie sicher, dass 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 sehen, die dem folgenden Beispiel ähneln.

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

    > **Note**
    >
    > Wenn sich die Protokollmeldungen nicht ändern, 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 berechtigt ist, Artefakte aus Ihrem S3-Bucket abzurufen. 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.
    >
    > 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 auf 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.

Stellen Sie Ihre Komponente (AWS CLI) bereit

Um Ihre Hello World-Komponente bereitzustellen

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. Die `configurationUpdate` gibt an, dass die Komponentenkonfiguration in einer JSON-codierten Zeichenfolge zusammengeführt werden soll. Dieses Konfigurationsupdate legt den Hello `Message` World-Parameter `universe` für das Gerät in dieser Bereitstellung auf fest.

2. Führen Sie den folgenden Befehl aus, um die Komponente auf Ihrem Greengrass-Core-Gerät bereitzustellen. Sie können die Bereitstellung auf Geräten durchführen, bei denen es sich um einzelne Geräte handelt, oder auf Dinggruppen, bei denen es sich um Gruppen von Geräten handelt. *MyGreengrassCore*Ersetzen Sie es durch den Namen der AWS IoT Sache für Ihr Kerngerät.

   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 aus, die dem folgenden Beispiel ähnelt.

   ```
   {
     "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. Stellen Sie sicher, dass 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 sehen, die dem folgenden Beispiel ähneln.

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

> **Note**
>
> Wenn sich die Protokollmeldungen nicht ändern, 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 berechtigt ist, Artefakte aus Ihrem S3-Bucket abzurufen. 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.
>
> 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 auf 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. Die AWS IoT Greengrass Core-Software und Ihre Hello World-Komponente laufen auf Ihrem Gerät. Außerdem ist Ihre Hello World-Komponente im AWS IoT Greengrass Cloud-Dienst verfügbar, um sie auf anderen Geräten bereitzustellen. Weitere Informationen zu den Themen, die in diesem Tutorial behandelt werden, finden Sie im Folgenden:

- AWS IoT Greengrass Komponenten erstellen
- Veröffentlichen Sie Komponenten zur Bereitstellung auf Ihren Kerngeräten
- AWS IoT Greengrass Komponenten auf Geräten bereitstellen

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. Diese Schritte gelten nicht für Nucleus Lite. Wenn Sie Nucleus Lite zum ersten Mal verwenden AWS IoT Greengrass V2, empfehlen wir Ihnen, zuerst das Tutorial „Erste Schritte“ zu absolvieren, um ein Kerngerät einzurichten und die Funktionen von zu erkunden. AWS IoT Greengrass

Themen

- Unterstützte Plattformen
- Anforderungen an Speichergeräte
- Anforderungen an die Lambda-Funktion
- Richten Sie eine ein AWS-Konto
- Installieren Sie die AWS IoT Greengrass Core-Software
- Führen Sie die AWS IoT Greengrass Core-Software aus
- Führen Sie die AWS IoT Greengrass Core-Software in einem Docker-Container aus
- Konfiguration der AWS IoT Greengrass Core-Software
- Aktualisieren Sie die AWS IoT Greengrass Core-Software (OTA)
- Deinstallieren Sie die AWS IoT Greengrass Core-Software

Unterstützte Plattformen

- Von Greengrass Nucleus unterstützte Plattformen
- Von Greengrass Nucleus Lite unterstützte Plattformen

Anforderungen an Speichergeräte

- Anforderungen an das Greengrass Nucleus-Gerät
- Anforderungen an das Greengrass Nucleus Lite-Gerät

Anforderungen an die Lambda-Funktion

> **Important**
>
> Greengrass Lambda-Funktionen werden derzeit von Greengrass Nucleus Lite nicht unterstützt.

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. AWS Lambda Funktionen ausführen
- 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 der Speicherbegrenzung 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.

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/die Anmeldung.
2. Folgen Sie den Online-Anweisungen.

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

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

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 AWS CLI die Konfiguration für die Verwendung AWS IAM Identity Center im AWS Command Line Interface Benutzerhandbuch vornehmen.
In IAM (Nicht empfohlen)	Verwendung von langfristigen Anmeldeinformationen für den Zugriff auf AWS.	Folgen Sie den Anleitungen unter IAM-Benutzer für den Notfallzugriff erstellen im IAM-Benutzerhandbuch.	Sie konfigurieren den programmgesteuerten Zugriff unter Verwendung der Informationen unter Verwalten der Zugriffsschlüssel für IAM-Benutzer im IAM-Benutzerhandbuch.

Installieren Sie die AWS IoT Greengrass Core-Software

AWS IoT Greengrass erstreckt sich AWS auf Edge-Geräte, sodass diese auf die von ihnen generierten Daten reagieren können, während sie sie gleichzeitig AWS Cloud für Verwaltung, Analyse und dauerhaften Speicher verwenden. Installieren Sie die AWS IoT Greengrass Core-Software auf Edge-Geräten, um sie zu integrieren AWS IoT Greengrass und die AWS 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 Greengrass Core-Software enthält ein Installationsprogramm, das Ihr Gerät als Greengrass-Core-Gerät einrichtet. Wenn Sie das Installationsprogramm ausführen, können Sie Optionen wie den Stammordner und den AWS-Region zu verwendenden Ordner konfigurieren. Sie können wählen, ob das Installationsprogramm die erforderlichen AWS IoT und IAM-Ressourcen für Sie erstellt. Sie können sich auch dafür entscheiden, lokale Entwicklungstools bereitzustellen, um ein Gerät zu konfigurieren, das Sie für die Entwicklung benutzerdefinierter Komponenten verwenden.

Für die Verbindung AWS IoT und den Betrieb der AWS IoT Greengrass Core-Software sind die AWS Cloud folgenden IAM-Ressourcen erforderlich:

- Irgendein AWS IoT Ding. Wenn Sie ein Gerät als Objekt AWS IoT 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. Weitere Informationen finden Sie unter Geräteauthentifizierung und Autorisierung für AWS IoT Greengrass.
- (Optional) Eine AWS IoT Dinggruppe. Sie verwenden Dinggruppen, um Flotten von Greengrass-Kerngeräten zu verwalten. Wenn Sie Softwarekomponenten auf Ihren Geräten bereitstellen, können Sie wählen, ob Sie sie auf einzelnen Geräten oder auf Gerätegruppen bereitstellen möchten. Sie können einer Dinggruppe ein Gerät hinzufügen, um die Softwarekomponenten dieser Dinggruppe auf dem Gerät bereitzustellen. Weitere Informationen finden Sie unter AWS IoT Greengrass Komponenten auf Geräten bereitstellen.
- Eine IAM-Rolle. Greengrass-Core-Geräte verwenden den AWS IoT Core Credentials Provider, um Aufrufe von AWS Diensten mit einer IAM-Rolle zu autorisieren. Diese Rolle ermöglicht es Ihrem Gerät AWS IoT, mit Amazon Simple Storage Service (Amazon S3) zu interagieren,

CloudWatch Protokolle an Amazon Logs zu senden und benutzerdefinierte Komponentenartefakte herunterzuladen. Weitere Informationen finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

- Ein AWS IoT Rollenalias. Greengrass-Core-Geräte verwenden den Rollenalias, um die zu verwendende IAM-Rolle zu identifizieren. Mit dem Rollenalias können Sie die IAM-Rolle ändern, aber die Gerätekonfiguration beibehalten. Weitere Informationen finden Sie im AWS IoT Core Entwicklerhandbuch unter Autorisieren von direkten Aufrufen von AWS Diensten.

Wählen Sie eine der folgenden Optionen, um die AWS IoT Greengrass Core-Software auf Ihrem Core-Gerät zu installieren.

- Schnelle Installation

  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 AWS Anmeldeinformationen für das Installationsprogramm angeben, um Ressourcen in Ihrem AWS-Konto zu erstellen.

  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 Sie die AWS IoT Greengrass Core-Software mit automatischer Ressourcenbereitstellung.

- Manuelle Installation

  Wählen Sie diese Option, um die erforderlichen AWS Ressourcen manuell zu erstellen oder um sie hinter einer Firewall oder einem Netzwerk-Proxy zu installieren. 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 IAM-Ressourcen selbst erstellen. Sie können Ihr Gerät auch so konfigurieren, dass es eine Verbindung über Port 443 oder über einen Netzwerk-Proxy herstellt. Sie können die AWS IoT Greengrass Core-Software auch so konfigurieren, dass sie einen privaten Schlüssel und ein Zertifikat verwendet, die Sie in einem Hardware-Sicherheitsmodul (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 Flottenbereitstellung AWS IoT

  Wählen Sie diese Option, um die erforderlichen AWS Ressourcen anhand 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-Client-Zertifikats, mit dem das Gerät AWS Cloud für den normalen Betrieb eine Verbindung mit dem herstellt. Sie können die Antragszertifikate 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 sie eine Verbindung über Port 443 oder über einen Netzwerk-Proxy herstellen.

  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 wenn Sie mehr Kontrolle über den Bereitstellungsprozess wünschen. AWS IoT Greengrass bietet eine Schnittstelle, die Sie für den Informationsaustausch zwischen Ihrer benutzerdefinierten Bereitstellungsanwendung und dem AWS IoT Greengrass Core-Softwareinstaller implementieren können.

  Weitere Informationen finden Sie unter Installieren Sie die AWS IoT Greengrass Core-Software mit benutzerdefinierter Ressourcenbereitstellung.

AWS IoT Greengrass stellt auch containerisierte Umgebungen bereit, in denen die Core-Software ausgeführt wird. AWS IoT Greengrass Sie können ein Dockerfile verwenden, um es AWS IoT Greengrass in einem Docker-Container auszuführen.

Themen

- Installieren Sie die 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
- Argumente des Installers

Installieren Sie die AWS IoT Greengrass Core-Software mit automatischer Ressourcenbereitstellung

Die AWS IoT Greengrass Core-Software enthält ein Installationsprogramm, das Ihr Gerät als Greengrass-Core-Gerät einrichtet. Um ein Gerät schnell einzurichten, kann das Installationsprogramm das AWS IoT AWS IoT Ding, die Dinggruppe, die AWS IoT IAM-Rolle und den Rollenalias bereitstellen, die das Kerngerät für den Betrieb benötigt. Das Installationsprogramm kann auch die lokalen Entwicklungstools auf dem Kerngerät 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 keine AWS Anmeldeinformationen für das Gerät angeben können, können Sie die AWS Ressourcen bereitstellen, die das Kerngerät für den Betrieb benötigt. Sie können die Entwicklungstools auch auf einem Kerngerät bereitstellen, um es als Entwicklungsgerät zu verwenden. Auf diese Weise können Sie dem Gerät weniger Berechtigungen zuweisen, 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 Kerngerät die Anforderungen für die Installation und Ausführung der AWS IoT Greengrass Core-Software v2.0 erfüllt.

Themen

- Richten Sie die Geräteumgebung ein
- Geben Sie AWS Anmeldeinformationen für das Gerät ein
- Laden Sie die AWS IoT Greengrass Core-Software herunter
- Installieren Sie die Core-Software AWS IoT Greengrass

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 Argumente des Installers.

   ```
   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 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. *`password`*Durch ein sicheres Passwort ersetzen.

```
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. *`password`*Ersetzen Sie es durch das zuvor festgelegte Benutzerkennwort.

```
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

Geben Sie AWS Anmeldeinformationen für das Gerät ein

Geben Sie Ihre AWS Anmeldeinformationen für Ihr Gerät ein, 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.

Um AWS Anmeldeinformationen für das Gerät bereitzustellen

- Geben Sie Ihre AWS Anmeldeinformationen für das Gerät ein, damit das Installationsprogramm die AWS IoT und IAM-Ressourcen für Ihr Kerngerät bereitstellen kann. Um die Sicherheit zu erhöhen, empfehlen wir Ihnen, temporäre Anmeldeinformationen für eine IAM-Rolle zu erwerben, die nur die für die Bereitstellung erforderlichen Mindestberechtigungen gewährt. 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 die Anmeldeinformationen abzurufen und sie dem AWS IoT Greengrass Core-Software-Installationsprogramm zur Verfügung zu stellen:

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

  a. Geben Sie die Zugriffsschlüssel-ID, den geheimen Zugriffsschlüssel und das Sitzungstoken aus dem IAM Identity Center ein. Weitere Informationen finden Sie unter Manuelles Aktualisieren von Anmeldeinformationen im Abschnitt Temporäre Anmeldeinformationen abrufen und aktualisieren 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 Sicherheitsanmeldedaten aus einer IAM-Rolle:

  a. Geben Sie die Zugriffsschlüssel-ID, den geheimen Zugriffsschlüssel und das Sitzungstoken aus einer IAM-Rolle an, die Sie annehmen. Weitere Informationen zum Abrufen dieser Anmeldeinformationen finden Sie unter Temporäre Sicherheitsanmeldeinformationen anfordern 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 zur Verfügung gestellt werden soll, finden Sie unter. Minimale IAM-Richtlinie für das Installationsprogramm zur Bereitstellung von Ressourcen Weitere Informationen zum Abrufen langfristiger Anmeldeinformationen finden Sie unter Verwaltung 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 für die 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, sodass sie nicht mehr gültig sind. Weitere Informationen finden Sie im AWS Identity and Access Management Benutzerhandbuch unter Aktualisieren von Zugriffsschlüsseln.

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 es *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 Folgendes angeben:

- Erstellen Sie die AWS Ressourcen, die das Kerngerät für den Betrieb benötigt.
- 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.

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 Hinweise zu den Argumenten, die Sie angeben können, finden Sie unterArgumente des Installers.

> **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. Führen Sie das AWS IoT Greengrass Core-Installationsprogramm aus. Ersetzen Sie die Argumentwerte in Ihrem Befehl wie folgt.

   > **Note**
   >
   > Windows hat eine Pfadlängenbegrenzung von 260 Zeichen. Wenn Sie Windows verwenden, verwenden Sie einen Stammordner wie `C:\greengrass\v2` oder, `D:\greengrass\v2` um die Greengrass-Komponentenpfade unter der Obergrenze von 260 Zeichen zu halten.

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

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

c. *region*. Das, AWS-Region in dem Ressourcen gefunden oder erstellt werden sollen.

d. *MyGreengrassCore*. Der Name der AWS IoT Sache für Ihr Greengrass-Core-Gerät. Wenn das Ding nicht existiert, erstellt es das Installationsprogramm. Das Installationsprogramm lädt die Zertifikate herunter, um sich als das AWS IoT Ding zu authentifizieren. Weitere Informationen finden Sie unter Geräteauthentifizierung und Autorisierung für AWS IoT Greengrass.

> **Note**
>
> Der Name der Sache darf keine Doppelpunkte (`:`) enthalten.

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

> **Note**
>
> Der Name der Dinggruppe darf keine Doppelpunkte (`:`) enthalten.

f. *GreengrassV2IoTThingPolicy*. Der Name der AWS IoT Richtlinie, die es den Greengrass-Core-Geräten ermöglicht, mit AWS IoT und AWS IoT Greengrass zu kommunizieren. Wenn die AWS IoT Richtlinie nicht existiert, erstellt das Installationsprogramm eine zulässige 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 V2 Kerngeräte.

g. *GreengrassV2TokenExchangeRole*. Der Name der IAM-Rolle, die es dem Greengrass-Core-Gerät ermöglicht, temporäre AWS Anmeldeinformationen abzurufen. Wenn die Rolle nicht existiert, erstellt das Installationsprogramm sie und erstellt und fügt eine Richtlinie mit

dem Namen hinzu. *GreengrassV2TokenExchangeRole*Access Weitere Informationen finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

h. *GreengrassCoreTokenExchangeRoleAlias*. Der Alias für die IAM-Rolle, der es dem Greengrass-Core-Gerät ermöglicht, später temporäre Anmeldeinformationen abzurufen. Wenn der Rollenalias nicht existiert, erstellt ihn das Installationsprogramm 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` dass die AWS IoT Greengrass Core-Software als Systemdienst eingerichtet werden soll.

Das Installationsprogramm gibt die folgenden Meldungen aus, wenn es erfolgreich ist:

- Wenn Sie angeben`--provision`, druckt das Installationsprogramm, `Successfully configured Nucleus with provisioned resource details` ob es die Ressourcen erfolgreich konfiguriert hat.
- Wenn Sie dies angeben`--deploy-dev-tools`, druckt das Installationsprogramm, `Configured Nucleus to deploy aws.greengrass.Cli component` ob es die Einrichtung erfolgreich erstellt hat.
- 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 Dienst eingerichtet und ausgeführt hat.
- Wenn Sie nichts angeben`--setup-system-service true`, gibt das Installationsprogramm aus, `Launched Nucleus successfully` ob es erfolgreich war und die Software ausgeführt hat.

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 AWS IoT Greengrass Core-Software-Stammordner festzulegen. */greengrass/v2*Ersetzen

Sie es durch den Stammordner, den Sie in Ihrem Installationsbefehl angegeben haben, und */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 Systemdienst installiert haben, führt das Installationsprogramm die Software für Sie aus. Andernfalls müssen Sie die Software manuell ausführen. Weitere Informationen finden Sie unter Führen Sie die AWS IoT Greengrass Core-Software aus.

> **Note**
>
> Standardmäßig erlaubt die IAM-Rolle, die das Installationsprogramm erstellt, 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.

> **Note**
>
> Wenn die AWS IoT Greengrass Core-Software eine Verbindung zur Cloud herstellt, wird Ihr Gerät als Core-Gerät erkannt.

Weitere Informationen zur Konfiguration und Verwendung der Software finden Sie unter: AWS IoT Greengrass

- Konfiguration der AWS IoT Greengrass Core-Software
- AWS IoT Greengrass Komponenten entwickeln
- AWS IoT Greengrass Komponenten auf Geräten bereitstellen
- 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
- Erstelle ein Ding AWS IoT
- Erstellen Sie das Ding-Zertifikat
- Konfigurieren Sie das Ding-Zertifikat
- Erstellen Sie eine Token-Exchange-Rolle
- Laden Sie Zertifikate auf das Gerät herunter

- Richten Sie die Geräteumgebung ein
- Laden Sie die AWS IoT Greengrass Core-Software herunter
- Installieren Sie die Core-Software AWS IoT Greengrass

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"
   }
   ```

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.

   - *MyGreengrassCore*Ersetzen Sie es durch den Namen des zu verwendenden Dings. 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 ähnlich wie im folgenden Beispiel aus, 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 es *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 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-----",
  "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 Objekte unterstützt IDs, 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 bei der Initialisierung des Tokens und der Generierung des privaten Schlüssels angeben. 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. `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. *`device-ip-address`*Ersetzen Sie es durch die IP-Adresse Ihres Geräts und *`~/iotdevicekey.csr`* ersetzen Sie es 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 AWS IoT Ding herunterladen.

   ```
   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 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-----"
}
```

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 es *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 V2 Kerngeräte.

   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.

- *GreengrassV2IoTThingPolicy*Ersetzen Sie es 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 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.

- *GreengrassV2IoTThingPolicy*Ersetzen Sie es durch den Namen der 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.

   - *GreengrassV2TokenExchangeRole*Ersetzen Sie es 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.

- *GreengrassV2TokenExchangeRoleAccess*Ersetzen Sie es 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 *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 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.

   - *GreengrassCoreTokenExchangeRoleAliasPolicy*Ersetzen Sie es 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.

- *GreengrassCoreTokenExchangeRoleAliasPolicy*Ersetzen Sie es durch den Namen der AWS IoT Rollenalias-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. *`device-ip-address`*Ersetzen Sie es 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.

   > **Note**
   >
   > Windows hat eine Pfadlängenbegrenzung von 260 Zeichen. Wenn Sie Windows verwenden, verwenden Sie einen Stammordner wie `C:\greengrass\v2` oder, `D:\greengrass\v2` um die Greengrass-Komponentenpfade unter der Obergrenze von 260 Zeichen zu halten.

   Linux or Unix

   - Ersetzen Sie es */greengrass/v2* durch den zu verwendenden Ordner.

   ```
   sudo mkdir -p /greengrass/v2
   ```

   Windows Command Prompt

   - *`C:\greengrass\v2`*Ersetzen Sie durch den zu verwendenden Ordner.

   ```
   mkdir C:\greengrass\v2
   ```

PowerShell

- *C:\greengrass\v2*Ersetzen Sie durch den zu verwendenden Ordner.

```
mkdir C:\greengrass\v2
```

3. (Nur Linux) Legen Sie die Berechtigungen des übergeordneten Elements des Greengrass-Stammordners fest.

   - */greengrass*Ersetzen Sie es 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 es *C:\greengrass\v2* durch den Ordner, den Sie verwenden möchten.

   ```
   robocopy %USERPROFILE%\greengrass-v2-certs C:\greengrass\v2 /E
   ```

   PowerShell

   - *C:\greengrass\v2*Ersetzen Sie durch den zu verwendenden Ordner.

   ```
   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. *device-ip-address*Ersetzen Sie es 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.

> **Note**
>
> Windows hat eine Pfadlängenbegrenzung von 260 Zeichen. Wenn Sie Windows verwenden, verwenden Sie einen Stammordner wie `C:\greengrass\v2` oder, `D:\greengrass\v2` um die Greengrass-Komponentenpfade unter der Obergrenze von 260 Zeichen zu halten.

Linux or Unix

- Ersetzen Sie es *`/greengrass/v2`* durch den zu verwendenden Ordner.

```
sudo mkdir -p /greengrass/v2
```

Windows Command Prompt

- *`C:\greengrass\v2`*Ersetzen Sie durch den zu verwendenden Ordner.

```
mkdir C:\greengrass\v2
```

PowerShell

- *`C:\greengrass\v2`*Ersetzen Sie durch den zu verwendenden Ordner.

```
mkdir C:\greengrass\v2
```

3. (Nur Linux) Legen Sie die Berechtigungen des übergeordneten Elements des Greengrass-Stammordners fest.

   - */greengrass*Ersetzen Sie es 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 Argumente des Installers.

```
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 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. *`password`*Durch ein sicheres Passwort ersetzen.

```
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. *password*Ersetzen Sie es durch das zuvor festgelegte Benutzerkennwort.

   ```
   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 es *`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 nach der Installation der AWS IoT Greengrass Core-Software entfernen.

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 unterArgumente des Installers.

> **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: "2.16.1"
       configuration:
         awsRegion: "us-west-2"
         iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
         iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
         iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
   west-2.amazonaws.com"
   ```

   Führen Sie dann die folgenden Schritte aus:

   - Ersetzen Sie jede Instanz von */greengrass/v2* durch den Greengrass-Stammordner.
   - Ersetze es *MyGreengrassCore* durch den Namen der AWS IoT Sache.
   - *2.16.1*Ersetzen Sie durch die Version der AWS IoT Greengrass Core-Software.
   - *us-west-2*Ersetzen Sie durch den AWS-Region Ort, an dem Sie die Ressourcen erstellt haben.

- *GreengrassCoreTokenExchangeRoleAlias*Ersetzen Sie es 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: "2.16.1"
    configuration:
      awsRegion: "us-west-2"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      mqtt:
        port: 443
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "https://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
```

3. 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 -Provider-Plugin in eine Datei mit dem Namen herunter. `aws.greengrass.crypto.Pkcs11Provider.jar` *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: "2.16.1"
       configuration:
         awsRegion: "us-west-2"
         iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
         iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
         iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
   west-2.amazonaws.com"
     aws.greengrass.crypto.Pkcs11Provider:
       configuration:
         name: "softhsm_pkcs11"
         library: "/usr/local/Cellar/softhsm/2.6.1/lib/softhsm/libsofthsm2.so"
         slot: 1
         userPin: "1234"
   ```

   Führen Sie dann die folgenden Schritte aus:

   - Ersetzen Sie jede Instanz von *iotdevicekey* in 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.

- Ersetze es *MyGreengrassCore* durch den Namen der AWS IoT Sache.
- *2.16.1*Ersetzen Sie durch die Version der AWS IoT Greengrass Core-Software.
- *us-west-2*Ersetzen Sie durch den AWS-Region Ort, an dem Sie die Ressourcen erstellt haben.
- *GreengrassCoreTokenExchangeRoleAlias*Ersetzen Sie es 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: "2.16.1"
    configuration:
      awsRegion: "us-west-2"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
      mqtt:
        port: 443
```

```
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "https://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
  aws.greengrass.crypto.Pkcs11Provider:
    configuration:
      name: "softhsm_pkcs11"
      library: "/usr/local/Cellar/softhsm/2.6.1/lib/softhsm/libsofthsm2.so"
      slot: 1
      userPin: "1234"
```

4. 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 Führen Sie die AWS IoT Greengrass Core-Software aus.

> **Note**
>
> Wenn die AWS IoT Greengrass Core-Software eine Verbindung zur Cloud herstellt, wird Ihr Gerät als Core-Gerät erkannt.

Weitere Informationen zur Konfiguration und Verwendung der Software finden Sie unter: AWS IoT Greengrass

- Konfiguration der AWS IoT Greengrass Core-Software
- AWS IoT Greengrass Komponenten entwickeln
- AWS IoT Greengrass Komponenten auf Geräten bereitstellen
- 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 AWS IoT Flottenbereitstellung für Greengrass-Kerngeräte einrichten.

> **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
- Endpunkte abrufen AWS IoT
- Laden Sie Zertifikate auf das Gerät herunter
- Richten Sie die Geräteumgebung ein
- Laden Sie die AWS IoT Greengrass Core-Software herunter
- Laden Sie das AWS IoT Fleet Provisioning Plugin herunter
- Installieren Sie die AWS IoT Greengrass Core-Software
- AWS IoT Flottenbereitstellung für Greengrass-Kerngeräte einrichten
- Konfigurieren Sie das AWS IoT Flottenbereitstellungs-Plugin
- AWS IoT Changelog des Plug-ins für die Flottenbereitstellung

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. Der folgende Beispielbefehl überträgt diese Dateien in einem Ordner mit dem Namen `claim-certs` auf Ihrem Entwicklungscomputer auf das Gerät. *`device-ip-address`*Ersetzen Sie es 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.

> **Note**
>
> Windows hat eine Pfadlängenbegrenzung von 260 Zeichen. Wenn Sie Windows verwenden, verwenden Sie einen Stammordner wie `C:\greengrass\v2` oder, `D:\greengrass\v2` um die Greengrass-Komponentenpfade unter der Obergrenze von 260 Zeichen zu halten.

Linux or Unix

- Ersetzen Sie es *`/greengrass/v2`* durch den Ordner, den Sie verwenden möchten.

```
sudo mkdir -p /greengrass/v2
```

Windows Command Prompt

- *`C:\greengrass\v2`*Ersetzen Sie durch den zu verwendenden Ordner.

```
mkdir C:\greengrass\v2
```

PowerShell

- *`C:\greengrass\v2`*Ersetzen Sie durch den zu verwendenden Ordner.

```
mkdir C:\greengrass\v2
```

3. (Nur Linux) Legen Sie die Berechtigungen des übergeordneten Elements des Greengrass-Stammordners fest.

   - *`/greengrass`*Ersetzen Sie es 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 mithilfe des `--component-default-user` Installer-Arguments erstellt. Weitere Informationen finden Sie unter Argumente des Installers.

```
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.

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 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. *`password`*Durch ein sicheres Passwort ersetzen.

```
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. *password*Ersetzen Sie es durch das zuvor festgelegte Benutzerkennwort.

   ```
   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 es *`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/fleetprovisioningbyclaim-latest.jar

> **Note**
>
> Sie können eine bestimmte Version des AWS IoT Fleet Provisioning-Plug-ins von der folgenden Adresse herunterladen. Ersetzen Sie es *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 von Ihnen bereitgestellten Teilkonfigurationsdatei 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 unterArgumente des Installers.

> **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 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 Sie das AWS IoT Flottenbereitstellungs-Plugin.

   Linux or Unix

   ```
   ---
   services:
     aws.greengrass.Nucleus:
       version: "2.16.1"
     aws.greengrass.FleetProvisioningByClaim:
       configuration:
         rootPath: "/greengrass/v2"
         awsRegion: "us-west-2"
         iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
         iotCredentialEndpoint: "device-credentials-prefix.credentials.iot.us-
   west-2.amazonaws.com"
         iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
         provisioningTemplate: "GreengrassFleetProvisioningTemplate"
         claimCertificatePath: "/greengrass/v2/claim-certs/claim.pem.crt"
         claimCertificatePrivateKeyPath: "/greengrass/v2/claim-certs/
   claim.private.pem.key"
         rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
         templateParameters:
           ThingName: "MyGreengrassCore"
           ThingGroupName: "MyGreengrassCoreGroup"
   ```

Windows

```
---
services:
  aws.greengrass.Nucleus:
    version: "2.16.1"
  aws.greengrass.FleetProvisioningByClaim:
    configuration:
      rootPath: "C:\\greengrass\\v2"
      awsRegion: "us-west-2"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredentialEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "C:\\greengrass\\v2\\claim-certs\\claim.pem.crt"
      claimCertificatePrivateKeyPath: "C:\\greengrass\\v2\\claim-certs\
\claim.private.pem.key"
      rootCaPath: "C:\\greengrass\\v2\\AmazonRootCA1.pem"
      templateParameters:
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
```

Führen Sie dann die folgenden Schritte aus:

- *2.16.1*Ersetzen Sie es 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`
- *us-west-2*Ersetzen Sie durch die AWS Region, in der Sie die Provisioning-Vorlage und andere Ressourcen erstellt haben.
- Ersetzen Sie das `iotDataEndpoint` durch Ihren AWS IoT Datenendpunkt.
- Ersetzen Sie den `iotCredentialEndpoint` durch den Endpunkt Ihrer AWS IoT Anmeldeinformationen.

- *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: "2.16.1"
    configuration:
      mqtt:
        port: 443
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "http://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
  aws.greengrass.FleetProvisioningByClaim:
    configuration:
      rootPath: "/greengrass/v2"
      awsRegion: "us-west-2"
```

```
      iotDataEndpoint: "device-data-prefix-ats.iot.us-
west-2.amazonaws.com"
      iotCredentialEndpoint: "device-credentials-
prefix.credentials.iot.us-west-2.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "/greengrass/v2/claim-certs/claim.pem.crt"
      claimCertificatePrivateKeyPath: "/greengrass/v2/claim-certs/
claim.private.pem.key"
      rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
      templateParameters:
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
      mqttPort: 443
      proxyUrl: "http://my-proxy-server:1100"
      proxyUserName: "Mary_Major"
      proxyPassword: "pass@word1357"
```

Windows

```
---
services:
  aws.greengrass.Nucleus:
    version: "2.16.1"
    configuration:
      mqtt:
        port: 443
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "http://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
  aws.greengrass.FleetProvisioningByClaim:
    configuration:
      rootPath: "C:\\greengrass\\v2"
      awsRegion: "us-west-2"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-
west-2.amazonaws.com"
      iotCredentialEndpoint: "device-credentials-
prefix.credentials.iot.us-west-2.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
```

```
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "C:\\greengrass\\v2\\claim-certs\
\claim.pem.crt"
      claimCertificatePrivateKeyPath: "C:\\greengrass\\v2\\claim-certs\
\claim.private.pem.key"
      rootCaPath: "C:\\greengrass\\v2\\AmazonRootCA1.pem"
      templateParameters:
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
      mqttPort: 443
      proxyUrl: "http://my-proxy-server:1100"
      proxyUserName: "Mary_Major"
      proxyPassword: "pass@word1357"
```

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 Führen Sie die AWS IoT Greengrass Core-Software aus.

Weitere Informationen zur Konfiguration und Verwendung der Software finden Sie unter: AWS IoT Greengrass

- Konfiguration der AWS IoT Greengrass Core-Software
- AWS IoT Greengrass Komponenten entwickeln
- AWS IoT Greengrass Komponenten auf Geräten bereitstellen
- Greengrass-Befehlszeilenschnittstelle

AWS IoT Flottenbereitstellung für Greengrass-Kerngeräte einrichten

Um die AWS IoT Greengrass Core-Software mit Flottenbereitstellung zu installieren, müssen Sie zunächst die folgenden Ressourcen in Ihrem einrichten. AWS-Konto Diese Ressourcen ermöglichen es Geräten, sich bei Greengrass-Core-Geräten zu registrieren AWS IoT und als solche zu arbeiten. Folgen Sie einmal den Schritten in diesem Abschnitt, um diese Ressourcen in Ihrem AWS-Konto zu erstellen und zu konfigurieren.

- Eine IAM-Rolle für den Tokenaustausch, mit der Kerngeräte Aufrufe von Diensten autorisieren. AWS
- Ein AWS IoT Rollenalias, der auf die Token-Exchange-Rolle verweist.
- (Optional) Eine AWS IoT Richtlinie, anhand derer Kerngeräte Anrufe an die AWS IoT Greengrass Dienste AWS IoT und autorisieren. Diese AWS IoT Richtlinie muss die `iot:AssumeRoleWithCertificate` Berechtigung für den AWS IoT Rollenalias gewähren, der auf die Token-Exchange-Rolle verweist.

  Sie können eine einzige AWS IoT Richtlinie für alle Kerngeräte in Ihrer Flotte verwenden, oder Sie können Ihre Flottenbereitstellungsvorlage so konfigurieren, dass für jedes Kerngerät eine AWS IoT Richtlinie erstellt wird.
- Eine Vorlage für die AWS IoT Flottenbereitstellung. In dieser Vorlage muss Folgendes angegeben werden:
  - AWS IoT Irgendeine Ressource. Sie können eine Liste vorhandener Dinggruppen angeben, um Komponenten für jedes Gerät bereitzustellen, wenn es online geht.
  - Eine AWS IoT Richtlinienressource. Diese Ressource kann eine der folgenden Eigenschaften definieren:
    - Der Name einer vorhandenen AWS IoT Richtlinie. Wenn Sie diese Option wählen, verwenden die Kerngeräte, die Sie anhand dieser Vorlage erstellen, dieselbe AWS IoT Richtlinie, und Sie können ihre Berechtigungen als Flotte verwalten.

    - Ein AWS IoT Richtliniendokument. Wenn Sie diese Option wählen, verwendet jedes Core-Gerät, das Sie anhand dieser Vorlage erstellen, eine eigene AWS IoT Richtlinie, und Sie können die Berechtigungen für jedes einzelne Core-Gerät verwalten.
  - Eine AWS IoT Zertifikatsressource. Diese Zertifikatsressource muss den `AWS::IoT::Certificate::Id` Parameter verwenden, um das Zertifikat an das Kerngerät anzuhängen. Weitere Informationen finden Sie im AWS IoT Developer Guide unter Just-in-time Provisioning.
- Ein AWS IoT Provisioning-Anspruchszertifikat und ein privater Schlüssel für die Flottenbereitstellungsvorlage. Sie können dieses Zertifikat und den privaten Schlüssel während der Herstellung in Geräte einbetten, sodass sich die Geräte selbst registrieren und bereitstellen können, wenn sie online gehen.

  > **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
- Eine AWS IoT Richtlinie, die Sie dem Provisioning-Antragszertifikat beifügen, damit sich Geräte registrieren und die Flottenbereitstellungsvorlage verwenden können.

Themen

- Erstellen Sie eine Token-Austauschrolle
- Erstellen Sie eine AWS IoT Richtlinie

- Erstellen Sie eine Vorlage für die Flottenbereitstellung
- Erstellen Sie ein Provisioning-Anspruchszertifikat und einen privaten Schlüssel

Erstellen Sie eine Token-Austauschrolle

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.

Um eine IAM-Rolle für den Token-Austausch 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.

- *GreengrassV2TokenExchangeRole*Ersetzen Sie es 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.

- *GreengrassV2TokenExchangeRoleAccess*Ersetzen Sie es 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 *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 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

Erstellen Sie eine AWS IoT Richtlinie

Nachdem Sie ein Gerät als Objekt registriert AWS IoT haben, kann dieses Gerät ein digitales Zertifikat zur Authentifizierung verwenden. AWS Dieses Zertifikat enthält eine oder mehrere AWS IoT Richtlinien, die die Berechtigungen definieren, die ein Gerät mit dem Zertifikat verwenden kann. Diese Richtlinien ermöglichen dem Gerät die Kommunikation mit AWS IoT und AWS IoT Greengrass.

Bei der AWS IoT Flottenbereitstellung stellen Geräte eine Verbindung her, AWS IoT um ein Gerätezertifikat zu erstellen und herunterzuladen. In der Flottenbereitstellungsvorlage, die Sie im nächsten Abschnitt erstellen, können Sie angeben, ob AWS IoT den Zertifikaten aller Geräte dieselbe AWS IoT Richtlinie zugewiesen wird oder ob für jedes Gerät eine neue Richtlinie erstellt wird.

In diesem Abschnitt erstellen Sie eine AWS IoT Richtlinie, die an die Zertifikate aller Geräte AWS IoT angehängt wird. Mit diesem Ansatz können Sie die Berechtigungen für alle Geräte als Flotte

verwalten. Wenn Sie lieber für jedes Gerät eine neue AWS IoT Richtlinie erstellen möchten, können Sie diesen Abschnitt überspringen und bei der Definition Ihrer Flottenvorlage auf die darin enthaltene Richtlinie verweisen.

Um eine AWS IoT Richtlinie zu erstellen

- Erstellen Sie eine AWS IoT Richtlinie, 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 future Änderungen, die neue Greengrass-Operationen erfordern, funktioniert. Diese Richtlinie ermöglicht auch die `iot:AssumeRoleWithCertificate` Genehmigung, die es Ihren Geräten ermöglicht, die Token-Austauschrolle 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 V2 Kerngeräte.

  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.

     - Ersetzen Sie die `iot:AssumeRoleWithCertificate` Ressource durch den ARN des AWS IoT Rollenalias, den Sie im vorherigen Abschnitt erstellt haben.

     JSON

     ```
     {
       "Version":"2012-10-17",
       "Statement": [
         {
           "Effect": "Allow",
     ```

```
      "Action": [
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive",
        "iot:Connect",
        "greengrass:*"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:AssumeRoleWithCertificate",
      "Resource": "arn:aws:iot:us-east-1:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
    }
  ]
}
```

b. Erstellen Sie eine AWS IoT Richtlinie aus dem Richtliniendokument.

- *GreengrassV2IoTThingPolicy*Ersetzen Sie es durch den Namen der zu erstellenden Richtlinie.

```
aws iot create-policy --policy-name GreengrassV2IoTThingPolicy --policy-
document file://greengrass-v2-iot-policy.json
```

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

JSON

```
{
  "policyName": "GreengrassV2IoTThingPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
  "policyDocument": "{
    \"Version\": \"2012-10-17\",
    \"Statement\": [
      {
        \"Effect\": \"Allow\",
```

```
        \"Action\": [
          \"iot:Publish\",
          \"iot:Subscribe\",
          \"iot:Receive\",
          \"iot:Connect\",
          \"greengrass:*\"
        ],
        \"Resource\": [
          \"*\"
        ]
      },
      {
        \"Effect\": \"Allow\",
        \"Action\": \"iot:AssumeRoleWithCertificate\",
        \"Resource\": \"arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias\"
      }
    ]
  }",
  "policyVersionId": "1"
}
```

Erstellen Sie eine Vorlage für die Flottenbereitstellung

AWS IoT Vorlagen für die Flottenbereitstellung definieren, wie AWS IoT Dinge, Richtlinien und Zertifikate bereitgestellt werden. Um Greengrass-Kerngeräte mit dem Fleet Provisioning Plugin bereitzustellen, müssen Sie eine Vorlage erstellen, die Folgendes spezifiziert:

- Irgendeine Ressource AWS IoT . Sie können eine Liste vorhandener Dinggruppen angeben, um Komponenten für jedes Gerät bereitzustellen, wenn es online geht.
- Eine AWS IoT Richtlinienressource. Diese Ressource kann eine der folgenden Eigenschaften definieren:
  - Der Name einer vorhandenen AWS IoT Richtlinie. Wenn Sie diese Option wählen, verwenden die Kerngeräte, die Sie anhand dieser Vorlage erstellen, dieselbe AWS IoT Richtlinie, und Sie können ihre Berechtigungen als Flotte verwalten.
  - Ein AWS IoT Richtliniendokument. Wenn Sie diese Option wählen, verwendet jedes Core-Gerät, das Sie anhand dieser Vorlage erstellen, eine eigene AWS IoT Richtlinie, und Sie können die Berechtigungen für jedes einzelne Core-Gerät verwalten.

- Eine AWS IoT Zertifikatsressource. Diese Zertifikatsressource muss den `AWS::IoT::Certificate::Id` Parameter verwenden, um das Zertifikat an das Kerngerät anzuhängen. Weitere Informationen finden Sie im AWS IoT Developer Guide unter Just-in-time Provisioning.

In der Vorlage können Sie angeben, dass das AWS IoT Ding zu einer Liste vorhandener Dinggruppen hinzugefügt werden soll. Wenn das Kerngerät AWS IoT Greengrass zum ersten Mal eine Verbindung herstellt, empfängt es Greengrass-Bereitstellungen für jede Dinggruppe, der es angehört. Sie können Dinggruppen verwenden, um die neueste Software auf jedem Gerät bereitzustellen, sobald es online ist. Weitere Informationen finden Sie unter AWS IoT Greengrass Komponenten auf Geräten bereitstellen.

Für den AWS IoT Dienst sind AWS-Konto bei der Bereitstellung von Geräten Berechtigungen zum Erstellen und Aktualisieren von AWS IoT Ressourcen auf Ihrem Gerät erforderlich. Um dem AWS IoT Dienst Zugriff zu gewähren, erstellen Sie eine IAM-Rolle und geben sie bei der Erstellung der Vorlage an. AWS IoT stellt eine verwaltete Richtlinie (AWSIoTThingsRegistrierung) bereit, die den Zugriff auf alle Berechtigungen ermöglicht, die bei der Bereitstellung von Geräten verwendet werden AWS IoT könnten. 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 die Bereitstellung von Ressourcen für Geräte ermöglicht AWS IoT , und Sie erstellen eine Flottenbereitstellungsvorlage, die diese IAM-Rolle verwendet.

Um eine Vorlage für die Flottenbereitstellung zu erstellen

1. Erstellen Sie eine IAM-Rolle, die die Bereitstellung von Ressourcen in Ihrem übernehmen AWS IoT kann. AWS-Konto Gehen Sie wie folgt vor:

   a. Erstellen Sie eine Datei, die das Dokument mit der Vertrauensrichtlinie enthält, mit dem AWS IoT Sie die Rolle übernehmen können.

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

      ```
      nano aws-iot-trust-policy.json
      ```

      Kopieren Sie den folgenden JSON-Code in die Datei.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. Erstellen Sie eine IAM-Rolle mit dem Dokument zur Vertrauensrichtlinie.

- *GreengrassFleetProvisioningRole*Ersetzen Sie sie 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 Anfrage erfolgreich ist.

c. Lesen Sie sich die AWSIoTThingsRegistrierungsrichtlinie durch, die den Zugriff auf alle Berechtigungen ermöglicht, die AWS IoT möglicherweise bei der Bereitstellung von Geräten verwendet werden. Sie können diese verwaltete Richtlinie verwenden oder eine benutzerdefinierte Richtlinie erstellen, die spezifische Berechtigungen für Ihren Anwendungsfall definiert. Wenn Sie sich dafür entscheiden, eine benutzerdefinierte Richtlinie zu erstellen, tun Sie dies jetzt.

d. Fügen Sie die IAM-Richtlinie der Flottenbereitstellungsrolle hinzu.

- 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 Anfrage erfolgreich ist.

2. (Optional) Erstellen Sie einen Pre-Provisioning-Hook. Dabei handelt es sich um eine AWS Lambda Funktion, die Vorlagenparameter überprüft, die Geräte bei der Registrierung angeben. Sie können einen Pre-Provisioning-Hook verwenden, um mehr Kontrolle darüber zu erhalten, welche und wie viele Geräte in Ihrem Gerät integriert sind. AWS-KontoWeitere Informationen finden Sie unter Pre-Provisioning Hooks im Developer Guide.AWS IoT Core
3. Erstellen Sie eine Vorlage für die Flottenbereitstellung. Gehen Sie wie folgt vor:

   a. Erstellen Sie eine Datei, die das Vorlagendokument für die Bereitstellung enthält.

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

      ```
      nano greengrass-fleet-provisioning-template.json
      ```

      Schreiben Sie das Provisioning-Vorlagendokument. Sie können mit der folgenden Beispielvorlage für die Bereitstellung beginnen, in der angegeben wird, dass ein Objekt mit den folgenden AWS IoT Eigenschaften erstellt werden soll:

      - Der Name der Sache ist der Wert, den Sie im `ThingName` Vorlagenparameter angeben.
      - Das Ding ist ein Mitglied der Dinggruppe, die Sie im `ThingGroupName` Vorlagenparameter angeben. Die Dinggruppe muss in Ihrer vorhanden sein AWS-Konto.
      - Dem Zertifikat der Sache ist die angegebene AWS IoT Richtlinie `GreengrassV2IoTThingPolicy` angehängt.

      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**
>
> *MyThingMyPolicy*, und *MyCertificate* sind willkürliche Namen, die jede Ressourcenspezifikation in der Flottenbereitstellungsvorlage identifizieren. AWS IoT verwendet diese Namen nicht in den Ressourcen, die es anhand der Vorlage erstellt. Sie können diese Namen verwenden oder sie durch Werte ersetzen, die Ihnen helfen, die einzelnen Ressourcen in der Vorlage zu identifizieren.

b. Erstellen Sie die Vorlage für die Flottenbereitstellung anhand des Vorlagendokuments für die Bereitstellung von Flotten.

- Ersetzen Sie *GreengrassFleetProvisioningTemplate* durch den Namen der zu erstellenden Vorlage.
- Ersetzen Sie die Beschreibung der Vorlage 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 Argument an. `--pre-provisioning-hook`
>
> ```
> --pre-provisioning-hook targetArn=arn:aws:lambda:us-
> west-2:123456789012:function:GreengrassPreProvisioningHook
> ```

Die Antwort sieht ähnlich wie im folgenden Beispiel aus, wenn die Anfrage erfolgreich ist.

```
{
    "templateArn": "arn:aws:iot:us-west-2:123456789012:provisioningtemplate/
GreengrassFleetProvisioningTemplate",
    "templateName": "GreengrassFleetProvisioningTemplate",
    "defaultVersionId": 1
}
```

Erstellen Sie ein Provisioning-Anspruchszertifikat und einen privaten Schlüssel

Anspruchszertifikate sind X.509-Zertifikate, die es Geräten ermöglichen, sich als AWS IoT Dinge zu registrieren und ein eindeutiges X.509-Gerätezertifikat abzurufen, das für den regulären Betrieb verwendet werden kann. Nachdem Sie ein Anspruchszertifikat erstellt haben, fügen Sie eine AWS IoT Richtlinie hinzu, die es Geräten ermöglicht, damit eindeutige Gerätezertifikate zu erstellen und diese mit einer Flottenbereitstellungsvorlage bereitzustellen. Geräte mit dem Antragszertifikat können nur mithilfe der Bereitstellungsvorlage bereitgestellt werden, 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**
>
> 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

Um ein Provisioning zu erstellen, beanspruchen Sie ein Zertifikat und einen privaten Schlüssel

1. Erstellen Sie einen Ordner, in den Sie das Anspruchszertifikat und den privaten Schlüssel herunterladen.

   ```
   mkdir claim-certs
   ```

2. Erstellen und speichern Sie ein Zertifikat und einen privaten Schlüssel, um sie für die Bereitstellung zu verwenden. AWS IoT stellt Client-Zertifikate 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 Anfrage erfolgreich ist. Speichern Sie den ARN des Zertifikats, um ihn später zu verwenden.

3. Erstellen und fügen Sie eine AWS IoT Richtlinie hinzu, die es Geräten ermöglicht, das Zertifikat zu verwenden, um eindeutige Gerätezertifikate zu erstellen und diese mit der Flottenbereitstellungsvorlage bereitzustellen. Die folgende Richtlinie ermöglicht den Zugriff auf die MQTT-API zur Gerätebereitstellung. Weitere Informationen finden Sie unter MQTT-API zur Gerätebereitstellung im Entwicklerhandbuch.AWS IoT Core

   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-provisioning-claim-iot-policy.json
      ```

      Kopieren Sie den folgenden JSON-Code in die Datei.

- Ersetzen Sie jede Instanz von *region* durch die Instanz, für AWS-Region die Sie die Flottenbereitstellung eingerichtet haben.
- Ersetzen Sie jede Instanz von *account-id* durch Ihre AWS-Konto ID.
- Ersetzen Sie jede Instanz von *GreengrassFleetProvisioningTemplate* durch den Namen der Flottenbereitstellungsvorlage, die Sie im vorherigen Abschnitt erstellt haben.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:Connect",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Receive"
      ],
      "Resource": [
      "arn:aws:iot:us-east-1:123456789012:topic/$aws/certificates/create/
*",
  "arn:aws:iot:us-east-1:123456789012:topic/$aws/provisioning-
templates/GreengrassFleetProvisioningTemplate/provision/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
      "Resource": [
      "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/certificates/
create/*",
  "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/provisioning-
templates/GreengrassFleetProvisioningTemplate/provision/*"
      ]
    }
  ]
```

```
}
```

b. Erstellen Sie eine AWS IoT Richtlinie anhand des Richtliniendokuments.

- *GreengrassProvisioningClaimPolicy*Ersetzen Sie es durch den Namen der zu erstellenden Richtlinie.

```
aws iot create-policy --policy-name GreengrassProvisioningClaimPolicy --policy-
document file://greengrass-provisioning-claim-iot-policy.json
```

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

JSON

```
{
  "policyName": "GreengrassProvisioningClaimPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassProvisioningClaimPolicy",
  "policyDocument": "{
    \"Version\": \"2012-10-17\",
    \"Statement\": [
      {
        \"Effect\": \"Allow\",
        \"Action\": \"iot:Connect\",
        \"Resource\": \"*\"
      },
      {
        \"Effect\": \"Allow\",
        \"Action\": [
          \"iot:Publish\",
          \"iot:Receive\"
        ],
        \"Resource\": [
        \"arn:aws:iot:us-east-1:123456789012:topic/$aws/certificates/
create/*\",
  \"arn:aws:iot:us-east-1:123456789012:topic/$aws/provisioning-
templates/GreengrassFleetProvisioningTemplate/provision/*\"
        ]
      },
      {
        \"Effect\": \"Allow\",
```

```
        \"Action\": \"iot:Subscribe\",
        \"Resource\": [
        \"arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
certificates/create/*\",
  \"arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/provisioning-
templates/GreengrassFleetProvisioningTemplate/provision/*\"
        ]
      }
    ]
  }",
  "policyVersionId": "1"
}
```

4. Fügen Sie die AWS IoT Richtlinie dem Provisioning Claim Certificate bei.

   - *GreengrassProvisioningClaimPolicy*Ersetzen Sie es durch den Namen der Richtlinie, die angehängt werden soll.
   - Ersetzen Sie den Ziel-ARN durch den ARN des Provisioning-Anspruchszertifikats.

   ```
   aws iot attach-policy --policy-name GreengrassProvisioningClaimPolicy --
   target arn:aws:iot:us-west-2:123456789012:cert/
   aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
   ```

   Der Befehl hat keine Ausgabe, wenn die Anfrage erfolgreich ist.

Sie verfügen jetzt über ein Provisioning-Anspruchszertifikat und einen privaten Schlüssel, mit denen sich Geräte registrieren AWS IoT und sich selbst als Greengrass-Core-Geräte bereitstellen können. Sie können das Antragszertifikat und den privaten Schlüssel während der Herstellung in Geräte einbetten oder das Zertifikat und den Schlüssel auf die Geräte kopieren, bevor Sie die AWS IoT Greengrass Core-Software installieren. Weitere Informationen finden Sie unter Installieren Sie die AWS IoT Greengrass Core-Software mit AWS IoT Flottenbereitstellung.

Konfigurieren Sie das AWS IoT Flottenbereitstellungs-Plugin

Das AWS IoT Fleet Provisioning Plugin bietet die folgenden Konfigurationsparameter, die Sie anpassen können, wenn Sie die AWS IoT Greengrass Core-Software mit Fleet Provisioning installieren.

`rootPath`

Der Pfad zu dem Ordner, der als Stammverzeichnis für die AWS IoT Greengrass Core-Software verwendet werden soll.

`awsRegion`

Der AWS-Region , den das Fleet Provisioning-Plugin zur Bereitstellung von AWS Ressourcen verwendet.

`iotDataEndpoint`

Der AWS IoT Datenendpunkt für Ihren AWS-Konto.

`iotCredentialEndpoint`

Der Endpunkt der AWS IoT Anmeldeinformationen für Ihren AWS-Konto.

`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.

`provisioningTemplate`

Die Vorlage für die AWS IoT Flottenbereitstellung, die für die Bereitstellung AWS von Ressourcen verwendet werden soll. In dieser Vorlage muss Folgendes angegeben werden:

- AWS IoT Irgendeine Ressource. Sie können eine Liste vorhandener Dinggruppen angeben, um Komponenten für jedes Gerät bereitzustellen, wenn es online geht.
- Eine AWS IoT Richtlinienressource. Diese Ressource kann eine der folgenden Eigenschaften definieren:
  - Der Name einer vorhandenen AWS IoT Richtlinie. Wenn Sie diese Option wählen, verwenden die Kerngeräte, die Sie anhand dieser Vorlage erstellen, dieselbe AWS IoT Richtlinie, und Sie können ihre Berechtigungen als Flotte verwalten.
  - Ein AWS IoT Richtliniendokument. Wenn Sie diese Option wählen, verwendet jedes Core-Gerät, das Sie anhand dieser Vorlage erstellen, eine eigene AWS IoT Richtlinie, und Sie können die Berechtigungen für jedes einzelne Core-Gerät verwalten.
- Eine AWS IoT Zertifikatsressource. Diese Zertifikatsressource muss den `AWS::IoT::Certificate::Id` Parameter verwenden, um das Zertifikat an das Kerngerät

anzuhängen. Weitere Informationen finden Sie im AWS IoT Developer Guide unter Just-in-time Provisioning.

Weitere Informationen finden Sie im AWS IoT Core Developer Guide unter Provisioning Templates.

`claimCertificatePath`

Der Pfad zum Provisioning-Anspruchszertifikat für die Bereitstellungsvorlage, das Sie unter angeben. `provisioningTemplate` Weitere Informationen finden Sie unter CreateProvisioningClaim in der AWS IoT Core -API-Referenz.

`claimCertificatePrivateKeyPath`

Der Pfad zum privaten Schlüssel des Provisioning-Anspruchszertifikats für die Bereitstellungsvorlage, die Sie in angeben. `provisioningTemplate` Weitere Informationen finden Sie unter CreateProvisioningClaim in der AWS IoT Core -API-Referenz.

> **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

`rootCaPath`

Der Pfad zum Zertifikat der Amazon Root Certificate Authority (CA).

`templateParameters`

(Optional) Die Zuordnung der Parameter, die der Flottenbereitstellungsvorlage zur Verfügung gestellt werden sollen. Weitere Informationen finden Sie im Abschnitt mit den Parametern für Bereitstellungsvorlagen im AWS IoT Core Entwicklerhandbuch.

`deviceId`

(Optional) Die Geräte-ID, die als Client-ID verwendet werden soll, wenn das Fleet Provisioning-Plugin eine MQTT-Verbindung herstellt. AWS IoT

Standard: Eine zufällige UUID.

`mqttPort`

(Optional) Der Port, der für MQTT-Verbindungen verwendet werden soll.

Standard: `8883`

`proxyUrl`

(Optional) Die URL des Proxyservers im Format`scheme://userinfo@host:port`. Um einen HTTPS-Proxy zu verwenden, müssen Sie Version 1.1.0 oder höher des Fleet Provisioning-Plug-ins verwenden.

- `scheme`— Das Schema, das oder sein muss. `http` `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

`proxyUserName`

(Optional) Der Benutzername, der den Proxyserver authentifiziert.

`proxyPassword`

(Optional) Der Benutzername, der den Proxyserver authentifiziert.

CSRPath

(Optional) Der Pfad zur CSR-Datei (Certificate Signing Request), die verwendet werden soll, um das Gerätezertifikat aus einer CSR zu erstellen. Weitere Informationen finden Sie im Entwicklerhandbuch unter Provisioning by Claim AWS IoT Core .

csrPrivateKeyPfad

(Optional, erforderlich, falls `csrPath` deklariert) Der Pfad zu dem privaten Schlüssel, der zur Generierung der CSR verwendet wurde. Der private Schlüssel muss zur Generierung der CSR verwendet worden sein. Weitere Informationen finden Sie im AWS IoT Core Entwicklerhandbuch unter Provisioning by Claim.

Pfad des Zertifikats

(Optional) Der Pfad, der zum Speichern des heruntergeladenen Gerätezertifikats verwendet werden soll.

privateKeyPath

(Optional) Der Pfad, der zum Speichern des heruntergeladenen privaten Geräteschlüssels verwendet werden soll.

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
1.2.2	Fehlerkorrekturen und Verbesserungen • Fügt Unterstützung für benutzerdefinierte Pfade für Zertifikate (`certificatePath`) und private Schlüssel (`privateKeyPath`) für Kerngeräte hinzu.
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 per Zertifikatsignieranforderung mit konfigurierbarem privaten Schlüsselpfad. • 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 erhalten 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
- Richten Sie die Geräteumgebung ein
- Laden Sie die AWS IoT Greengrass Core-Software herunter
- Installieren Sie die Core-Software AWS IoT Greengrass
- Entwickeln Sie benutzerdefinierte Provisioning-Plugins

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 Argumente des Installers.

   ```
   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 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. *`password`*Durch ein sicheres Passwort ersetzen.

```
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. *`password`*Ersetzen Sie es durch das zuvor festgelegte Benutzerkennwort.

```
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 es *`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 Teilkonfigurationsdatei, die Sie bereitstellen, 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 unterArgumente des Installers.

> **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.

So installieren Sie die AWS IoT Greengrass Core-Software (Linux)

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.

```
---
system:
  rootpath: "/greengrass/v2"
  # The following values are optional. Return them from the provisioning plugin or
 set them here.
  # certificateFilePath: ""
  # privateKeyPath: ""
  # rootCaPath: ""
  # thingName: ""
services:
  aws.greengrass.Nucleus:
    version: "2.16.1"
    configuration:
      # The following values are optional. Return them from the provisioning plugin
 or set them here.
      # awsRegion: ""
      # iotRoleAlias: ""
      # iotDataEndpoint: ""
      # iotCredEndpoint: ""
  com.example.CustomProvisioning:
    configuration:
      # You can specify configuration parameters to provide to your plugin.
      # pluginParameter: ""
```

Führen Sie dann die folgenden Schritte aus:

- *2.16.1*Ersetzen Sie es 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: "2.16.1"
    configuration:
      mqtt:
        port: 443
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "http://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
      # The following values are optional. Return them from the provisioning
 plugin or set them here.
      # awsRegion: ""
      # iotRoleAlias: ""
      # iotDataEndpoint: ""
      # iotCredEndpoint: ""
  com.example.CustomProvisioning:
    configuration:
      # You can specify configuration parameters to provide to your plugin.
      # pluginParameter: ""
```

3. 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.

   > **Note**
   >
   > Windows hat eine Pfadlängenbegrenzung von 260 Zeichen. Wenn Sie Windows verwenden, verwenden Sie einen Stammordner wie `C:\greengrass\v2` oder, `D:\greengrass\v2` um die Greengrass-Komponentenpfade unter der Obergrenze von 260 Zeichen zu halten.

   - 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 Führen Sie die AWS IoT Greengrass Core-Software aus.

Weitere Informationen zur Konfiguration und Verwendung der Software finden Sie unter: AWS IoT Greengrass

- Konfiguration der AWS IoT Greengrass Core-Software
- AWS IoT Greengrass Komponenten entwickeln
- AWS IoT Greengrass Komponenten auf Geräten bereitstellen
- Greengrass-Befehlszeilenschnittstelle

Entwickeln Sie benutzerdefinierte Provisioning-Plugins

Um ein benutzerdefiniertes Provisioning-Plugin zu entwickeln, erstellen Sie eine Java-Klasse, die die Schnittstelle implementiert. `com.aws.greengrass.provisioning.DeviceIdentityInterface` 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 die Greengrass Nucleus-Komponente. Das AWS IoT Greengrass Core-Softwareinstallationsprogramm verwendet diese Bereitstellungskonfiguration, um die AWS IoT Greengrass Core-Software auf einem Gerät zu konfigurieren.

Nachdem Sie ein benutzerdefiniertes Provisioning-Plugin entwickelt haben, erstellen Sie es als JAR-Datei, die Sie dem AWS IoT Greengrass Core-Softwareinstaller zur Verfügung stellen können, um Ihr Plugin während der Installation auszuführen. Das Installationsprogramm führt Ihr benutzerdefiniertes

Provisioning-Plugin in derselben JVM aus, die das Installationsprogramm verwendet, sodass Sie eine JAR erstellen können, die nur Ihren Plugin-Code enthält.

> **Note**
>
> Das AWS IoT Fleet-Provisioning-Plugin implementiert die Option, Fleet Provisioning während der Installation `DeviceIdentityInterface` zu verwenden. Das Fleet Provisioning-Plugin ist Open Source, sodass Sie den Quellcode untersuchen können, um ein Beispiel für die Verwendung der Provisioning-Plug-in-Schnittstelle zu sehen. Weitere Informationen finden Sie im AWS IoT Fleet Provisioning Plugin unter. GitHub

Themen

- Voraussetzungen
- Implementieren Sie die DeviceIdentityInterface Schnittstelle

Voraussetzungen

Um ein benutzerdefiniertes Provisioning-Plugin zu entwickeln, müssen Sie eine Java-Klasse erstellen, die die folgenden Anforderungen erfüllt:

- Verwendet das `com.aws.greengrass` Paket oder ein Paket innerhalb des `com.aws.greengrass` Pakets.
- Hat einen Konstruktor ohne Argumente.
- Implementiert die `DeviceIdentityInterface` Schnittstelle. Weitere Informationen finden Sie unter Implementieren Sie die DeviceIdentityInterface Schnittstelle.

Implementieren Sie die DeviceIdentityInterface Schnittstelle

Um die `com.aws.greengrass.provisioning.DeviceIdentityInterface` Schnittstelle in Ihrem benutzerdefinierten Plugin zu verwenden, fügen Sie den Greengrass-Kern als Abhängigkeit zu Ihrem Projekt hinzu.

Um das DeviceIdentityInterface in einem benutzerdefinierten Provisioning-Plugin-Projekt zu verwenden

- Sie können die Greengrass Nucleus JAR-Datei als Bibliothek oder den Greengrass Nucleus als Maven-Abhängigkeit hinzufügen. Führen Sie eine der folgenden Aktionen aus:
  - Um die Greengrass Nucleus JAR-Datei als Bibliothek hinzuzufügen, laden Sie die AWS IoT Greengrass Core-Software herunter, die die Greengrass Nucleus-JAR enthält. 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

    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 vom `nucleus` Artefakt in der Gruppe hinzu. `com.aws.greengrass` Sie müssen auch das `greengrass-common` Repository hinzufügen, da der Greengrass-Kern nicht im Maven Central Repository 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 Schnittstelle DeviceIdentityInterface

Die `com.aws.greengrass.provisioning.DeviceIdentityInterface` Schnittstelle hat die folgende Form.

> **Note**
>
> Sie können diese Klassen auch im com.aws.greengrass.provisioning Paket des Greengrass Nucleus-Quellcodes unter erkunden. 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 in `SystemConfiguration` und `NucleusConfiguration` ist für die Installation der Core-Software erforderlich, Sie können jedoch zurückkehren. AWS IoT Greengrass `null` Wenn Ihr benutzerdefiniertes Provisioning-Plug-in `null` für einen beliebigen Konfigurationswert zurückgegeben wird, müssen Sie diesen Wert in der System- oder Nucleus-Konfiguration angeben, wenn Sie die `config.yaml` Datei für das AWS IoT Greengrass Core-Software-Installationsprogramm erstellen. Wenn Ihr benutzerdefiniertes Provisioning-Plugin für eine Option, in der Sie auch definiert haben, einen Wert ungleich Null zurückgibt`config.yaml`, ersetzt das Installationsprogramm den Wert in `config.yaml` durch den vom Plugin zurückgegebenen Wert.

Argumente des Installers

Die AWS IoT Greengrass Core-Software umfasst ein Installationsprogramm, das die Software einrichtet und die erforderlichen AWS Ressourcen bereitstellt, damit das Greengrass-Core-Gerät ausgeführt werden kann. Das Installationsprogramm enthält die folgenden Argumente, die Sie zur Konfiguration der Installation angeben können:

`-h`, `--help`

(Optional) Zeigt die Hilfeinformationen des Installationsprogramms an.

`--version`

(Optional) Zeigt die Version der AWS IoT Greengrass Core-Software an.

`-Droot`

(Optional) Der Pfad zu dem Ordner, der als Stammverzeichnis für die AWS IoT Greengrass Core-Software verwendet werden soll.

> **Note**
>
> Dieses Argument legt eine JVM-Eigenschaft fest, sodass Sie sie zuvor angeben müssen, `-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 AWS IoT Greengrass Core-Software verwendet, um die erforderlichen AWS Ressourcen abzurufen oder zu erstellen.

`-p`, `--provision`

(Optional) Sie können dieses Gerät als Objekt registrieren AWS IoT und die AWS Ressourcen bereitstellen, die das Kerngerät benötigt. Wenn Sie angeben`true`, stellt die AWS IoT Greengrass Core-Software ein AWS IoT Ding, (optional) eine AWS IoT Dinggruppe, eine IAM-Rolle und einen AWS IoT Rollenalias bereit.

Standard: `false`

`-tn`, `--thing-name`

(Optional) Der Name der AWS IoT Sache, die Sie als dieses Kerngerät registrieren. Wenn das Ding mit dem Namen in Ihrem nicht existiert AWS-Konto, wird es von der AWS IoT Greengrass Core-Software erstellt.

> **Note**
>
> Der Dingname darf keine Doppelpunkte (`:`) enthalten.

Sie müssen angeben`--provision true`, ob dieses Argument angewendet werden soll.

Standard: `GreengrassV2IotThing_` plus eine zufällige UUID.

`-tgn`, `--thing-group-name`

(Optional) Der Name der Dinggruppe AWS IoT , der Sie das Ding dieses Kerngeräts hinzufügen. AWS IoT Wenn eine Bereitstellung auf diese Dinggruppe abzielt, empfängt dieses Kerngerät diese Bereitstellung, wenn es eine Verbindung herstellt AWS IoT Greengrass. Wenn die Dinggruppe mit diesem Namen in Ihrer nicht existiert AWS-Konto, wird sie von der AWS IoT Greengrass Core-Software erstellt.

> **Note**
>
> Der Name der Dinggruppe darf keine Doppelpunkte (`:`) enthalten.

Sie müssen angeben`--provision true`, ob dieses Argument angewendet werden soll.

`-tpn`, `--thing-policy-name`

Diese Funktion ist für Version 2.4.0 und höher der Greengrass Nucleus-Komponente verfügbar.

(Optional) Der Name der AWS IoT Richtlinie, die an das Ding-Zertifikat dieses Kerngeräts angehängt werden soll. AWS IoT Wenn die AWS IoT Richtlinie mit diesem Namen in Ihrem nicht vorhanden ist AWS-Konto, wird sie von der AWS IoT Greengrass Core-Software erstellt.

Die AWS IoT Greengrass Core-Software erstellt standardmäßig eine permissive AWS IoT Richtlinie. Sie können den Geltungsbereich dieser Richtlinie einschränken oder eine benutzerdefinierte Richtlinie erstellen, mit der Sie die Berechtigungen für Ihren Anwendungsfall einschränken. Weitere Informationen finden Sie unter Minimale AWS IoT Richtlinie für AWS IoT Greengrass V2 Kerngeräte.

Sie müssen angeben`--provision true`, ob dieses Argument angewendet werden soll.

Standard: `GreengrassV2IoTThingPolicy`

`-trn`, `--tes-role-name`

(Optional) Der Name der IAM-Rolle, die zum Abrufen von AWS Anmeldeinformationen verwendet werden soll, mit denen das Kerngerät mit AWS Diensten interagieren kann. Wenn die Rolle mit diesem Namen in Ihrer nicht vorhanden ist AWS-Konto, erstellt die AWS IoT Greengrass Core-Software sie anhand 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, ob dieses Argument angewendet `--provision true` werden soll.

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 Kerngerät bereitstellt. Wenn der Rollenalias mit diesem Namen in Ihrem nicht vorhanden ist AWS-Konto, erstellt ihn die AWS IoT Greengrass Core-Software und verweist ihn auf die von Ihnen angegebene IAM-Rolle.

Sie müssen angeben`--provision true`, ob dieses Argument angewendet werden soll.

Standard: `GreengrassV2TokenExchangeRoleAlias`

`-ss`, `--setup-system-service`

(Optional) Sie können die AWS IoT Greengrass Core-Software als Systemdienst einrichten, der ausgeführt wird, wenn das Gerät gestartet wird. Der Name des Systemdienstes lautet`greengrass`. Weitere Informationen finden Sie unter Den Greengrass Nucleus als Systemdienst konfigurieren.

Auf Linux-Betriebssystemen setzt dieses Argument voraus, dass das Systemd-Init-System auf dem Gerät verfügbar ist.

> **Important**
>
> Auf Windows Core-Geräten müssen Sie die AWS IoT Greengrass Core-Software als Systemdienst 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 auf Windows-Betriebssystemen ausführen.

Auf Linux-Betriebssystemen können Sie die Gruppe auch optional angeben. Geben Sie den Benutzer und die Gruppe durch einen Doppelpunkt getrennt an. Beispiel, **`ggc_user:ggc_group`**.

Die folgenden zusätzlichen Überlegungen gelten für Linux-Betriebssysteme:

- Wenn Sie als Root ausführen, ist der Standardbenutzer der Komponente der Benutzer, der in der Konfigurationsdatei definiert ist. Wenn die Konfigurationsdatei keinen Benutzer definiert, ist dies standardmäßig. `ggc_user:ggc_group` Wenn sie existieren `ggc_user` oder `ggc_group` nicht, erstellt die Software sie.
- Wenn Sie die Software 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 Kerngerät 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 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.

Sie müssen angeben`--provision true`, ob dieses Argument angewendet werden soll.

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 Nucleus-Konfiguration 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 Kerngerät. Wir empfehlen, dass in der Konfigurationsdatei nur die Konfigurationen aufgeführt sind, 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 JAR-Dateien für das Provisioning-Plugin bereitzustellen, z. B.

für die Installation mit Flottenbereitstellung oder benutzerdefinierter Bereitstellung oder für die Installation mit dem privaten Schlüssel und dem Zertifikat in einem Hardwaresicherheitsmodul.

`-s`, `--start`

(Optional) Sie können die AWS IoT Greengrass Core-Software nach der Installation starten und optional Ressourcen bereitstellen.

Standard: `true`

Führen Sie die AWS IoT Greengrass Core-Software aus

Nachdem Sie die AWS IoT Greengrass Core-Software installiert haben, führen Sie sie aus, um Ihr Gerät damit zu verbinden AWS IoT Greengrass.

Wenn Sie die AWS IoT Greengrass Core-Software installieren, können Sie angeben, ob sie als Systemdienst 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 Booten Ihres Geräts ausgeführt wird.

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

Themen

- Überprüfen Sie, ob die AWS IoT Greengrass Core-Software als Systemdienst ausgeführt wird
- Führen Sie die AWS IoT Greengrass Core-Software als Systemdienst aus
- Führen Sie die AWS IoT Greengrass Core-Software ohne Systemdienst aus

Überprüfen Sie, ob die AWS IoT Greengrass Core-Software als Systemdienst 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 Systemdienst zu installieren. Linux-Geräte benötigen das Systemd-Init-System, um die AWS IoT Greengrass Core-Software als Systemdienst einzurichten. Wenn Sie diese Option verwenden, führt das

Installationsprogramm die Software für Sie aus und konfiguriert sie so, dass sie beim Booten Ihres Geräts ausgeführt wird. Das Installationsprogramm gibt die folgende Meldung aus, wenn es die AWS IoT Greengrass Core-Software erfolgreich als Systemdienst installiert hat.

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

Wenn Sie die AWS IoT Greengrass Core-Software bereits installiert haben und die Installationsausgabe nicht verfügbar ist, können Sie überprüfen, ob die Software als Systemdienst installiert wurde.

Um zu überprüfen, ob die AWS IoT Greengrass Core-Software als Systemdienst installiert ist

- Führen Sie den folgenden Befehl aus, um den Status des Greengrass-Systemdienstes 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 Systemdienst 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 `greengrass.service` nicht gefunden wird, ist die AWS IoT Greengrass Core-Software nicht als Systemdienst installiert. Informationen zum Ausführen der Software finden Sie unterFühren Sie die AWS IoT Greengrass Core-Software ohne Systemdienst aus.

  Windows Command Prompt (CMD)

  ```
  sc query greengrass
  ```

Die Antwort ähnelt dem folgenden Beispiel, wenn die AWS IoT Greengrass Core-Software als Windows-Dienst 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-Dienst installiert und aktiv ist.

```
Status   Name               DisplayName
------   ----               -----------
Running  greengrass         greengrass
```

Führen Sie die AWS IoT Greengrass Core-Software als Systemdienst aus

Wenn die AWS IoT Greengrass Core-Software als Systemdienst installiert ist, können Sie den Systemdienstmanager verwenden, um die Software zu starten, zu beenden und zu verwalten. Weitere Informationen finden Sie unter Den Greengrass Nucleus als Systemdienst konfigurieren.

Um die AWS IoT Greengrass Core-Software auszuführen

- 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
```

Führen Sie die AWS IoT Greengrass Core-Software ohne Systemdienst aus

Wenn die Core-Software auf AWS IoT Greengrass Linux-Core-Geräten nicht als Systemdienst installiert ist, können Sie das Loader-Skript der Software ausführen, um die Software auszuführen.

Um die AWS IoT Greengrass Core-Software ohne Systemdienst auszuführen

- 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 weiterläuft.
  - Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Greengrass-Stammordner, den Sie verwenden.

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

Die Software druckt die folgende Meldung, wenn sie erfolgreich gestartet wurde.

```
Launched Nucleus successfully.
```

Führen Sie die AWS IoT Greengrass Core-Software in einem Docker-Container aus

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 zur Verfügung stellt, 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 angeben und dem AWS IoT Greengrass Core-Softwareinstaller erlauben möchten, automatisch die AWS Ressourcen bereitzustellen, die ein Greengrass-Core-Gerät für den Betrieb benötigt. Wenn Sie keine AWS Anmeldeinformationen angeben möchten, können Sie AWS Ressourcen manuell bereitstellen und die AWS IoT Greengrass Core-Software im Docker-Container ausführen.

Themen

- Unterstützte Plattformen und Anforderungen
- AWS IoT Greengrass Docker-Software wird heruntergeladen
- Wählen Sie aus, wie Ressourcen bereitgestellt werden AWS
- Erstellen Sie das AWS IoT Greengrass Container-Image aus einem Dockerfile
- AWS IoT Greengrass In einem Docker-Container mit automatischer Ressourcenbereitstellung ausführen
- AWS IoT Greengrass In einem Docker-Container mit manueller Ressourcenbereitstellung ausführen
- Fehlerbehebung AWS IoT Greengrass in einem Docker-Container

Unterstützte Plattformen und Anforderungen

Hostcomputer 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.

Komponenten im Prozessmodus ausführen

AWS IoT Greengrass unterstützt nicht die Ausführung von Lambda-Funktionen oder AWS-bereitgestellten Komponenten in einer isolierten Laufzeitumgebung innerhalb des AWS IoT

Greengrass Docker-Containers. Sie müssen diese Komponenten im Prozessmodus ohne jegliche Isolierung ausführen.

Wenn Sie eine Lambda-Funktionskomponente konfigurieren, setzen Sie den Isolationsmodus auf Kein Container. Weitere Informationen finden Sie unter AWS Lambda Funktionen ausführen.

Wenn Sie eine der folgenden Komponenten AWS bereitstellen, aktualisieren Sie die Konfiguration für jede Komponente, auf die der `containerMode` Parameter gesetzt werden soll. `NoContainer` Weitere Informationen zu Konfigurationsupdates finden Sie unterKomponentenkonfigurationen aktualisieren.

- CloudWatch Metriken
- Device Defender
- Firehose
- Modbus-RTU-Protokolladapter
- Amazon SNS

AWS IoT Greengrass Docker-Software wird heruntergeladen

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 auf einer anderen Plattformarchitektur ausgeführt wird. AWS IoT Greengrass

Laden Sie das Dockerfile-Paket von herunter. GitHub

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 dem Dockerfile finden Sie unter. Erstellen Sie das AWS IoT Greengrass Container-Image aus einem Dockerfile

Wählen Sie aus, wie Ressourcen bereitgestellt werden AWS

Wenn Sie die AWS IoT Greengrass Core-Software in einem Docker-Container installieren, können Sie wählen, ob Sie die AWS Ressourcen, die ein Greengrass-Core-Gerät für den Betrieb benötigt, automatisch bereitstellen oder Ressourcen verwenden möchten, die Sie manuell bereitstellen.

- Automatische Ressourcenbereitstellung — Das Installationsprogramm stellt das AWS IoT AWS IoT Ding, die Dinggruppe, die AWS IoT IAM-Rolle und den Rollenalias bereit, wenn Sie das

AWS IoT Greengrass Container-Image zum ersten Mal ausführen. Das Installationsprogramm kann auch die lokalen Entwicklungstools auf dem Kerngerät bereitstellen, sodass Sie das Gerät verwenden können, um benutzerdefinierte Softwarekomponenten zu entwickeln und zu testen. Um diese Ressourcen automatisch bereitzustellen, müssen Sie AWS Anmeldeinformationen als Umgebungsvariablen für das Docker-Image angeben.

Um die automatische Bereitstellung zu verwenden, müssen Sie die Docker-Umgebungsvariable festlegen `PROVISION=true` und eine Anmeldeinformationsdatei bereitstellen, um Ihre AWS Anmeldeinformationen für den Container bereitzustellen.

- Manuelle Ressourcenbereitstellung — Wenn Sie keine AWS Anmeldeinformationen für den Container bereitstellen möchten, können Sie die AWS Ressourcen manuell bereitstellen, bevor Sie das Container-Image ausführen. AWS IoT Greengrass 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` Manuelles Provisioning ist die Standardoption.

Weitere Informationen finden Sie unter Erstellen Sie das AWS IoT Greengrass Container-Image aus einem Dockerfile.

Erstellen Sie das AWS IoT Greengrass Container-Image aus einem Dockerfile

AWS stellt ein Dockerfile bereit, das Sie herunterladen und verwenden können, um die AWS IoT Greengrass Core-Software in einem Docker-Container auszuführen. Dockerfiles enthalten Quellcode zum Erstellen von Container-Images. AWS IoT Greengrass

Bevor Sie ein AWS IoT Greengrass Container-Image erstellen, müssen Sie Ihr Dockerfile so konfigurieren, dass es die Version der AWS IoT Greengrass Core-Software auswählt, die Sie installieren möchten. Sie können auch Umgebungsvariablen konfigurieren, um festzulegen, wie Ressourcen während der Installation bereitgestellt werden, und andere Installationsoptionen anpassen. In diesem Abschnitt wird beschrieben, wie Sie ein AWS IoT Greengrass Docker-Image aus einer Docker-Datei konfigurieren und erstellen.

Laden Sie das Dockerfile-Paket herunter

Sie können das AWS IoT Greengrass Dockerfile-Paket herunterladen von: 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.

Geben Sie die AWS IoT Greengrass Core-Softwareversion an

Verwenden Sie das folgende Build-Argument in der Docker-Datei, 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 Core-Software. AWS IoT Greengrass

`GREENGRASS_RELEASE_VERSION`

Die Version der AWS IoT Greengrass Core-Software. Standardmäßig lädt das Dockerfile die neueste verfügbare Version des Greengrass-Nukleus herunter. Stellen Sie den Wert auf die Version des Nucleus ein, den Sie herunterladen möchten.

Festlegen von Umgebungsvariablen

Mit Umgebungsvariablen können Sie anpassen, wie die 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 Ihren Container `docker run` zum Starten verwenden, übergeben Sie Umgebungsvariablen als Argumente im Befehl oder legen Sie Umgebungsvariablen in einer Umgebungsvariablendatei fest und übergeben Sie dann die Datei als Argument. Weitere Informationen zum Setzen von Umgebungsvariablen in Docker finden Sie in den Umgebungsvariablen in der Docker-Dokumentation.
- Wenn Sie Ihren Container `docker-compose up` zum Starten verwenden, legen Sie Umgebungsvariablen in einer Umgebungsvariablendatei fest und übergeben Sie die Datei dann als Argument. Weitere Informationen zum Setzen 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` PIDs an alle Mitglieder der PID-Gruppe, sodass die AWS IoT Greengrass Core-Software korrekt heruntergefahren werden kann, wenn der Docker-Container gestoppt wird.

`GGC_ROOT_PATH`

(Optional) Der Pfad zu dem Ordner innerhalb des Containers, der als Stammverzeichnis für die AWS IoT Greengrass Core-Software verwendet werden soll.

Standard: `/greengrass/v2`

`PROVISION`

(Optional) Legt fest, ob der AWS IoT Greengrass Core AWS Ressourcen bereitstellt.

- Wenn Sie dies angeben`true`, registriert die AWS IoT Greengrass Core-Software das Container-Image als eine AWS IoT Sache und stellt die AWS Ressourcen bereit, die das Greengrass-Core-Gerät benötigt. Die AWS IoT Greengrass Core-Software stellt ein AWS IoT Ding, (optional) eine AWS IoT Dinggruppe, eine IAM-Rolle und einen AWS IoT Rollenalias bereit. Weitere Informationen finden Sie unter AWS IoT Greengrass In einem Docker-Container mit automatischer Ressourcenbereitstellung ausführen.
- Wenn Sie dies angeben`false`, müssen Sie eine Konfigurationsdatei erstellen, die Sie dem AWS IoT Greengrass Core-Installationsprogramm zur Verfügung stellen, in der angegeben ist, dass die AWS Ressourcen und Zertifikate verwendet werden sollen, die Sie manuell erstellt haben. Weitere Informationen finden Sie unter AWS IoT Greengrass In einem Docker-Container mit manueller Ressourcenbereitstellung ausführen.

Standard: `false`

`AWS_REGION`

(Optional) Die AWS-Region , die die AWS IoT Greengrass Core-Software verwendet, um die erforderlichen AWS Ressourcen abzurufen oder zu erstellen.

Standard: `us-east-1`.

`THING_NAME`

(Optional) Der Name des Geräts AWS IoT , das Sie als dieses Core-Gerät registrieren. Wenn das Ding mit diesem Namen in Ihrem nicht existiert AWS-Konto, wird es von der AWS IoT Greengrass Core-Software erstellt.

Sie müssen angeben`PROVISION=true`, ob dieses Argument angewendet werden soll.

Standard: `GreengrassV2IotThing_` plus eine zufällige UUID.

`THING_GROUP_NAME`

(Optional) Der Name der Dinggruppe AWS IoT , der Sie dieses Kerngerät hinzufügen. AWS IoT Wenn eine Bereitstellung auf diese Dinggruppe abzielt, erhalten dieses und andere Kerngeräte in dieser Gruppe diese Bereitstellung, wenn eine Verbindung hergestellt wird. AWS IoT Greengrass Wenn die Dinggruppe mit diesem Namen in Ihrer nicht existiert AWS-Konto, wird sie von der AWS IoT Greengrass Core-Software erstellt.

Sie müssen angeben`PROVISION=true`, ob dieses Argument angewendet werden soll.

`TES_ROLE_NAME`

(Optional) Der Name der IAM-Rolle, die zum Abrufen von AWS Anmeldeinformationen verwendet werden soll, die es dem Greengrass-Core-Gerät ermöglichen, mit AWS Diensten zu interagieren. Wenn die Rolle mit diesem Namen in Ihrer nicht vorhanden ist AWS-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-Core-Gerät bereitstellt. Wenn der Rollenalias mit diesem Namen in Ihrem nicht vorhanden ist AWS-Konto, erstellt ihn die AWS IoT Greengrass Core-Software 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 Gruppe, die die AWS IoT Greengrass Core-Software zum Ausführen von Komponenten verwendet. Geben Sie den Benutzer und die Gruppe an, getrennt durch einen Doppelpunkt. Die Gruppe ist optional. Sie können beispielsweise **`ggc_user:ggc_group`** oder **`ggc_user`** angeben.

- Wenn Sie als Root ausführen, werden 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 sie existieren `ggc_user` oder `ggc_group` nicht, erstellt die Software sie.
- Wenn Sie die Software 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 zu verwenden, nicht in Produktionsumgebungen. Diese Komponente bietet Zugriff auf Informationen und Vorgänge, 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.

Standard: `false`

`INIT_CONFIG`

(Optional) Der Pfad zur Konfigurationsdatei, die für die Installation der AWS IoT Greengrass Core-Software verwendet werden soll. Sie können diese Option verwenden, um beispielsweise

neue Greengrass Core-Geräte mit einer bestimmten Nucleus-Konfiguration einzurichten oder um manuell bereitgestellte Ressourcen anzugeben. Sie müssen Ihre Konfigurationsdatei in den Pfad einhängen, den Sie in diesem Argument angeben.

`TRUSTED_PLUGIN`

Diese Funktion ist für Version 2.4.0 und höher der Greengrass Nucleus-Komponente verfügbar.

(Optional) Der Pfad zu einer JAR-Datei, die als vertrauenswürdiges Plugin geladen werden soll. Verwenden Sie diese Option, um JAR-Dateien für das Provisioning-Plugin bereitzustellen, z. B. um sie zusammen mit der Flottenbereitstellung oder der benutzerdefinierten Bereitstellung zu installieren.

`THING_POLICY_NAME`

Diese Funktion ist für Version 2.4.0 und höher der Greengrass Nucleus-Komponente verfügbar.

(Optional) Der Name der AWS IoT Richtlinie, die an das Ding-Zertifikat dieses Kerngeräts angehängt werden soll. AWS IoT Wenn die AWS IoT Richtlinie mit diesem Namen in Ihrem System nicht vorhanden ist, wird sie von AWS-Konto der AWS IoT Greengrass Core-Software erstellt.

Sie müssen angeben`PROVISION=true`, ob dieses Argument angewendet werden soll.

> **Note**
>
> Die AWS IoT Greengrass Core-Software erstellt standardmäßig eine permissive AWS IoT Richtlinie. Sie können den Geltungsbereich dieser Richtlinie einschränken oder eine benutzerdefinierte Richtlinie erstellen, mit der Sie die Berechtigungen für Ihren Anwendungsfall einschränken. Weitere Informationen finden Sie unter Minimale AWS IoT Richtlinie für AWS IoT Greengrass V2 Kerngeräte.

Geben Sie die zu installierenden Abhängigkeiten an

Die RUN-Anweisung im AWS IoT Greengrass Dockerfile bereitet die Containerumgebung für die Ausführung des AWS IoT Greengrass Core-Softwareinstallers vor. Sie können die Abhängigkeiten anpassen, die installiert werden, bevor das AWS IoT Greengrass Core-Softwareinstallationsprogramm im Docker-Container ausgeführt wird.

Erstellen Sie das Image AWS IoT Greengrass

Verwenden Sie das AWS IoT Greengrass Dockerfile, um ein AWS IoT Greengrass Container-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 das Image auch mit der Docker-CLI 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 das 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 dem 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 Greengrass Core-Software installiert. Sie können die AWS IoT Greengrass Core-Software jetzt in einem Docker-Container ausführen.

AWS IoT Greengrass In einem Docker-Container mit automatischer Ressourcenbereitstellung ausführen

Dieses Tutorial zeigt Ihnen, wie Sie die 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 keine AWS Anmeldeinformationen für den Container angeben können, können Sie die AWS Ressourcen bereitstellen, die das Kerngerät für den Betrieb benötigt. Sie können die Entwicklungstools auch auf einem Kerngerät bereitstellen, um es als Entwicklungsgerät zu verwenden. Auf diese Weise können Sie dem Gerät weniger Berechtigungen gewähren, wenn Sie den Container ausführen. Weitere Informationen finden Sie unter AWS IoT Greengrass In einem Docker-Container mit manueller Ressourcenbereitstellung ausführen.

Voraussetzungen

Um dieses Tutorial abzuschließen, benötigen Sie Folgendes.

- Ein AWS-Konto. Falls Sie noch keines haben, beachten Sie die Informationen unter Richten Sie eine ein AWS-Konto.
- Ein AWS IAM-Benutzer mit Berechtigungen zur Bereitstellung der AWS IoT und der IAM-Ressourcen für ein Greengrass-Core-Gerät. Das AWS IoT Greengrass Core-Softwareinstallationsprogramm verwendet Ihre AWS Anmeldeinformationen, um diese Ressourcen automatisch bereitzustellen. Informationen zur IAM-Mindestrichtlinie für die automatische Bereitstellung von Ressourcen finden Sie unterMinimale IAM-Richtlinie für das Installationsprogramm zur Bereitstellung von Ressourcen.
- Ein AWS IoT Greengrass Docker-Image. Sie können ein Image aus dem 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 Anmeldeinformationsdatei auf dem Host-Computer, die Ihre Sicherheitsanmeldeinformationen enthält. AWS Wenn Sie das AWS IoT Greengrass Docker-Image ausführen, müssen Sie den Ordner, der diese Anmeldeinformationsdatei enthält, `/root/.aws/` im Docker-Container bereitstellen. Das AWS IoT Greengrass Installationsprogramm verwendet diese Anmeldeinformationen, um Ressourcen in Ihrem bereitzustellen. AWS-Konto Informationen zu den minimalen IAM-Richtlinien, die das Installationsprogramm für die automatische Bereitstellung von Ressourcen benötigt, finden Sie unterMinimale 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 Verwaltung 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 Sicherheitsanmeldedaten mit dem AWS CLI 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` des `./greengrass-v2-credentials` Ordners 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 einbeziehen.

> **Important**
>
> Entfernen Sie die Anmeldeinformationsdatei vom Host-Computer, nachdem Sie den AWS IoT Greengrass Container gestartet haben. Wenn Sie die Anmeldeinformationsdatei nicht entfernen, bleiben Ihre AWS Anmeldeinformationen im Container gespeichert. Weitere Informationen finden Sie unter Führen Sie die AWS IoT Greengrass Core-Software in einem Container aus.

Erstellen Sie eine Umgebungsdatei

In diesem Tutorial werden mithilfe einer Umgebungsdatei die Umgebungsvariablen festgelegt, die an das AWS IoT Greengrass Core-Software-Installationsprogramm im Docker-Container übergeben werden. Sie können auch das `--env` Argument `-e` oder in Ihrem `docker run` Befehl verwenden, um Umgebungsvariablen im Docker-Container festzulegen, oder Sie können die Variablen in einem `environment` Block in der `docker-compose.yml` Datei festlegen.

1. Verwenden Sie einen Texteditor, um eine Umgebungsdatei mit dem Namen `.env` zu erstellen.

   Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um mithilfe von GNU Nano die Datei `.env` im aktuellen Verzeichnis zu erstellen.

   ```
   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*. Der AWS-Region Ort, an dem Sie die Ressourcen erstellt haben.
- *MyGreengrassCore*. Der Name der AWS IoT Sache. Wenn das Ding nicht existiert, erstellt es das Installationsprogramm. Das Installationsprogramm lädt die Zertifikate herunter, um sich als das AWS IoT Ding zu authentifizieren.
- *MyGreengrassCoreGroup*. Der Name der AWS IoT Dinggruppe. Wenn die Dinggruppe nicht existiert, erstellt das Installationsprogramm sie und fügt ihr das Ding hinzu. Wenn die Dinggruppe vorhanden ist und über eine aktive Bereitstellung verfügt, lädt das Kerngerät die in 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 abzurufen. Wenn die Rolle nicht existiert, erstellt das Installationsprogramm sie und erstellt eine Richtlinie mit dem Namen Access und fügt sie an. *GreengrassV2TokenExchangeRole* Weitere Informationen finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.
- *GreengrassCoreTokenExchangeRoleAlias*. Der Alias der Token-Exchange-Rolle. Wenn der Rollenalias nicht existiert, erstellt ihn das Installationsprogramm und verweist ihn auf die von Ihnen angegebene IAM-Token-Exchange-Rolle. 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 zu verwenden, nicht in Produktionsumgebungen. Diese Komponente bietet Zugriff auf Informationen und Vorgänge, 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 die AWS IoT Greengrass Core-Software in einem Container aus

Dieses Tutorial zeigt Ihnen, wie Sie das Docker-Image starten, das Sie in einem 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

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 docker run:

   - `--rm`. Reinigt den Behälter, wenn er austritt.
   - `--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 beenden.

   - `-it`. (Optional) Führt den Docker-Container im Vordergrund als interaktiven Prozess aus. Sie können dies durch das `-d` Argument ersetzen, den Docker-Container stattdessen im getrennten Modus auszuführen. Weitere Informationen finden Sie unter Separated vs Foreground in der Docker-Dokumentation.
   - `--name`. Führt einen Container mit dem Namen aus `aws-iot-greengrass`
   - `-v`. Hängt ein Volume in den Docker-Container ein, um die Konfigurationsdatei und die Zertifikatsdateien für die AWS IoT Greengrass Ausführung innerhalb des Containers 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 Setzen von Umgebungsvariablen erstellt haben. Wenn Sie

keine Umgebungsdatei erstellt haben, können Sie `--env` Argumente verwenden, um Umgebungsvariablen direkt in Ihrem Docker-Run-Befehl 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-Verkehr AWS IoT Greengrass verwendet wird. Verwenden Sie zusätzliche Argumente, um andere Ports zu öffnen. `-p`

> **Note**
>
> Um Ihren Docker-Container mit erhöhter Sicherheit auszuführen, können Sie die `--cap-add` Argumente `--cap-drop` und verwenden, um Linux-Funktionen für Ihren Container selektiv zu aktivieren. Weitere Informationen finden Sie in der Docker-Dokumentation unter Runtime-Rechte und Linux-Funktionen.

2. Entfernen Sie die Anmeldeinformationen `./greengrass-v2-credentials` vom Host-Gerät.

```
rm -rf ./greengrass-v2-credentials
```

> **Important**
>
> Sie entfernen diese Anmeldeinformationen, da sie umfassende Berechtigungen bieten, die das Kerngerät 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 angeben müssen, verwenden Sie den Token-Austauschdienst. Weitere Informationen finden Sie unter Interagieren Sie mit AWS Diensten.

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 mithilfe von GNU Nano die Datei `docker-compose.yml` im aktuellen Verzeichnis zu erstellen.

```
nano docker-compose.yml
```

> **Note**
>
> Sie können auch die neueste Version der AWS bereitgestellten Compose-Datei von herunterladen und verwenden. GitHub

2. Fügen Sie der Compose-Datei den folgenden Inhalt hinzu. Die Datei sollte ähnlich wie im folgenden Beispiel aussehen. *`docker-image`*Ersetzen Sie es 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 dieser Compose-Beispieldatei 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-Verkehr AWS IoT Greengrass verwendet wird.
- `env_file`— Gibt die Umgebungsdatei an, um die Umgebungsvariablen festzulegen, die an das AWS IoT Greengrass Core-Software-Installationsprogramm im Docker-Container übergeben werden. Dieser Parameter ist nur erforderlich, wenn Sie eine Umgebungsdatei zum Setzen 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 selektiv Linux-Funktionen für Ihren Container zu aktivieren. Weitere Informationen finden Sie unter Runtime-Rechte 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 `./greengrass-v2-credentials` vom Host-Gerät.

```
rm -rf ./greengrass-v2-credentials
```

> **Important**
>
> Sie entfernen diese Anmeldeinformationen, da sie umfassende Berechtigungen bieten, die das Kerngerät 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 angeben müssen, verwenden Sie den Token-Austauschdienst. Weitere Informationen finden Sie unter Interagieren Sie mit AWS Diensten.

Nächste Schritte

AWS IoT Greengrass Die Kernsoftware läuft jetzt in einem Docker-Container. Führen Sie den folgenden Befehl aus, um die Container-ID für den aktuell laufenden Container abzurufen.

```
docker ps
```

Anschließend können Sie den folgenden Befehl ausführen, um auf den Container zuzugreifen und die im Container ausgeführte AWS IoT Greengrass Core-Software zu erkunden.

```
docker exec -it container-id /bin/bash
```

Informationen zum Erstellen einer einfachen Komponente finden Sie Schritt 4: Entwickeln und testen Sie eine Komponente auf Ihrem Gerät unter Tutorial: Erste Schritte mit AWS IoT Greengrass V2

> **Note**
>
> Wenn Sie `docker exec` Befehle innerhalb des Docker-Containers ausführen, werden diese Befehle nicht in den Docker-Protokollen protokolliert. Um Ihre Befehle in den Docker-Protokollen zu protokollieren, fügen Sie dem Docker-Container eine interaktive Shell hinzu. Weitere Informationen finden Sie unter Hängen Sie eine interaktive Shell an den Docker-Container an.

Die AWS IoT Greengrass Core-Protokolldatei wird aufgerufen `greengrass.log` und befindet sich in. */greengrass/v2*`/logs` Die Protokolldateien der Komponenten befinden sich ebenfalls im selben Verzeichnis. Um Greengrass-Logs 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, dass Sie nur das Verzeichnis an das temporäre */greengrass/v2*`/logs` Protokollverzeichnis auf dem Host binden, anstatt das gesamte Greengrass-Verzeichnis zu mounten. Weitere Informationen finden Sie unter Greengrass-Logs außerhalb des Docker-Containers beibehalten.

Um einen laufenden AWS IoT Greengrass Docker-Container zu beenden, führen Sie oder aus. `docker stop` `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 übrig gebliebenen Zombie-Prozesse hilft. Weitere Informationen finden Sie unter Spezifizieren Sie einen Init-Prozess in der Docker-Dokumentation.

Informationen zur Behebung von Problemen bei der Ausführung AWS IoT Greengrass in einem Docker-Container finden Sie unter. Fehlerbehebung AWS IoT Greengrass in einem Docker-Container

AWS IoT Greengrass In einem Docker-Container mit manueller Ressourcenbereitstellung ausführen

Dieses Tutorial zeigt Ihnen, wie Sie die AWS IoT Greengrass Core-Software im Docker-Container mit manuell bereitgestellten Ressourcen installieren und ausführen. AWS

Themen

- Voraussetzungen
- Endpunkte abrufen AWS IoT
- Erstelle ein Ding AWS IoT
- Erstellen Sie das Ding-Zertifikat
- Erstellen Sie eine Token-Austausch-Rolle
- Laden Sie Zertifikate auf das Gerät herunter
- Erstellen Sie eine Konfigurationsdatei
- Erstellen Sie eine Umgebungsdatei
- Führen Sie die AWS IoT Greengrass Core-Software in einem Container aus
- Nächste Schritte

Voraussetzungen

Zum Durcharbeiten dieses Tutorials ist Folgendes erforderlich:

- Ein. 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 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.

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"
   }
   ```

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 AWS IoT etwas zu erschaffen

1. Erstelle AWS IoT etwas für dein Gerät. Führen Sie auf Ihrem Entwicklungscomputer den folgenden Befehl aus.

   - *MyGreengrassCore*Ersetzen Sie es durch den Namen des zu verwendenden Dings. 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 ähnlich wie im folgenden Beispiel aus, 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 es *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.

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 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-----",
  "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.

Als Nächstes konfigurieren Sie das Ding-Zertifikat. Weitere Informationen finden Sie unter Konfigurieren Sie das Ding-Zertifikat.

Erstellen Sie eine Token-Austausch-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.

- *GreengrassV2TokenExchangeRole*Ersetzen Sie es 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.

- *GreengrassV2TokenExchangeRoleAccess*Ersetzen Sie es 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 *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 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.

   - *GreengrassCoreTokenExchangeRoleAliasPolicy*Ersetzen Sie es 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.

- *GreengrassCoreTokenExchangeRoleAliasPolicy*Ersetzen Sie es durch den Namen der AWS IoT Rollenalias-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 laden Sie das Zertifikat der Amazon Root Certificate Authority (CA) herunter. Wenn Sie dann planen, die AWS IoT Greengrass Core-Software in Docker auf einem anderen Computer als Ihrem Entwicklungscomputer auszuführen, kopieren Sie die Zertifikate auf diesen Host-Computer. Die AWS IoT Greengrass Core-Software verwendet diese Zertifikate, um eine Verbindung zum AWS IoT Cloud-Dienst herzustellen.

Um Zertifikate auf das Gerät herunterzuladen

1. Laden Sie das Zertifikat der Amazon Root Certificate Authority (CA) auf Ihren Entwicklungscomputer herunter. AWS IoT Zertifikate sind standardmäßig dem Root-CA-Zertifikat von Amazon 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 Host-Computer. 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. *`device-ip-address`*Ersetzen Sie es durch die IP-Adresse Ihres Host-Computers.

   ```
   scp -r greengrass-v2-certs/ device-ip-address:~
   ```

Erstellen Sie eine Konfigurationsdatei

1. Erstellen Sie auf dem Host-Computer einen Ordner, in dem Sie Ihre Konfigurationsdatei ablegen.

   ```
   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 `config.yaml` zu verwenden.

```
nano ./greengrass-v2-config/config.yaml
```

3. Kopieren Sie den folgenden YAML-Inhalt in die Datei. Diese Teilkonfigurationsdatei spezifiziert Systemparameter und Greengrass-Nukleus-Parameter.

```
---
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 der AWS IoT Sache.
- *nucleus-version*. Die Version der zu AWS IoT Greengrass installierenden 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 es, um die Image-Version **`docker inspect image-id`** zu sehen.

- *region*. Der AWS-Region Ort, an dem Sie Ihre AWS IoT Ressourcen erstellt haben. Sie müssen auch denselben Wert für die `AWS_REGION` Umgebungsvariable in Ihrer Umgebungsdatei angeben.
- *GreengrassCoreTokenExchangeRoleAlias*. Der Alias der Token-Exchange-Rolle.
- *device-data-prefix*. Das Präfix für Ihren AWS IoT Datenendpunkt.
- *device-credentials-prefix*. Das Präfix für den Endpunkt Ihrer AWS IoT Anmeldeinformationen.

Erstellen Sie eine Umgebungsdatei

In diesem Tutorial werden mithilfe einer Umgebungsdatei die Umgebungsvariablen festgelegt, die an das AWS IoT Greengrass Core-Softwareinstallationsprogramm im Docker-Container übergeben werden. Sie können auch das `--env` Argument `-e` oder in Ihrem `docker run` Befehl verwenden, um Umgebungsvariablen im Docker-Container festzulegen, oder Sie können die Variablen in einem `environment` Block in der `docker-compose.yml` Datei festlegen.

1. Verwenden Sie einen Texteditor, um eine Umgebungsdatei mit dem Namen `.env` zu erstellen.

   Auf einem Linux-basierten System können Sie beispielsweise den folgenden Befehl ausführen, um mithilfe von GNU Nano die Datei `.env` im aktuellen Verzeichnis zu erstellen.

   ```
   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 für die Installation der AWS IoT Greengrass Core-Software verwendet werden soll.
   - *region*. Der AWS-Region Ort, an dem Sie Ihre AWS IoT Ressourcen erstellt haben. Sie müssen denselben Wert für den `awsRegion` Konfigurationsparameter in Ihrer Konfigurationsdatei angeben.

- */tmp/config/*. Der Ordner, in den 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 zu verwenden, nicht in Produktionsumgebungen. 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 die AWS IoT Greengrass Core-Software in einem Container aus

Dieses Tutorial zeigt Ihnen, wie Sie das Docker-Image starten, das Sie in einem 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

- Dieses Tutorial zeigt Ihnen, wie Sie das Docker-Image starten, das Sie in einem Docker-Container erstellt 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 docker run:

  - `--rm`. Reinigt den Behälter, wenn er austritt.
  - `--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 beenden.

- `-it`. (Optional) Führt den Docker-Container im Vordergrund als interaktiven Prozess aus. Sie können dies durch das `-d` Argument ersetzen, den Docker-Container stattdessen im getrennten Modus auszuführen. Weitere Informationen finden Sie unter Separated vs Foreground in der Docker-Dokumentation.
- `--name`. Führt einen Container mit dem Namen aus `aws-iot-greengrass`
- `-v`. Hängt ein Volume in den Docker-Container ein, um die Konfigurationsdatei und die Zertifikatsdateien für die AWS IoT Greengrass Ausführung innerhalb des Containers 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 Setzen von Umgebungsvariablen erstellt haben. Wenn Sie keine Umgebungsdatei erstellt haben, können Sie `--env` Argumente verwenden, um Umgebungsvariablen direkt in Ihrem Docker-Run-Befehl 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-Verkehr AWS IoT Greengrass verwendet wird. Verwenden Sie zusätzliche Argumente, um andere Ports zu öffnen. `-p`

> **Note**
>
> Um Ihren Docker-Container mit erhöhter Sicherheit auszuführen, können Sie die `--cap-add` Argumente `--cap-drop` und verwenden, um selektiv Linux-Funktionen für Ihren Container zu aktivieren. Weitere Informationen finden Sie in der Docker-Dokumentation unter Runtime-Rechte und Linux-Funktionen.

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 mithilfe von GNU Nano die Datei `docker-compose.yml` im aktuellen Verzeichnis zu erstellen.

   ```
   nano docker-compose.yml
   ```

   > **Note**
   >
   > Sie können auch die neueste Version der AWS bereitgestellten Compose-Datei von herunterladen und verwenden. GitHub

2. Fügen Sie der Compose-Datei den folgenden Inhalt hinzu. Die Datei sollte ähnlich wie im folgenden Beispiel aussehen. *`your-container-name:version`*Ersetzen Sie es 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 dieser Compose-Beispieldatei 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-Verkehr AWS IoT Greengrass verwendet wird.

- `env_file`— Gibt die Umgebungsdatei an, um die Umgebungsvariablen festzulegen, die an das AWS IoT Greengrass Core-Software-Installationsprogramm im Docker-Container übergeben werden. Dieser Parameter ist nur erforderlich, wenn Sie eine Umgebungsdatei zum Setzen 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 selektiv Linux-Funktionen für Ihren Container zu aktivieren. Weitere Informationen finden Sie unter Runtime-Rechte 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 Die Kernsoftware läuft jetzt in einem Docker-Container. Führen Sie den folgenden Befehl aus, um die Container-ID für den aktuell laufenden Container abzurufen.

```
docker ps
```

Anschließend können Sie den folgenden Befehl ausführen, um auf den Container zuzugreifen und die im Container ausgeführte AWS IoT Greengrass Core-Software zu erkunden.

```
docker exec -it container-id /bin/bash
```

Informationen zum Erstellen einer einfachen Komponente finden Sie Schritt 4: Entwickeln und testen Sie eine Komponente auf Ihrem Gerät unter Tutorial: Erste Schritte mit AWS IoT Greengrass V2

> **Note**
>
> Wenn Sie `docker exec` Befehle innerhalb des Docker-Containers ausführen, werden diese Befehle nicht in den Docker-Protokollen protokolliert. Um Ihre Befehle in den Docker-Protokollen zu protokollieren, fügen Sie dem Docker-Container eine interaktive Shell hinzu. Weitere Informationen finden Sie unter Hängen Sie eine interaktive Shell an den Docker-Container an.

Die AWS IoT Greengrass Core-Protokolldatei wird aufgerufen `greengrass.log` und befindet sich in. */greengrass/v2*`/logs` Die Protokolldateien der Komponenten befinden sich ebenfalls im selben Verzeichnis. Um Greengrass-Logs 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, dass Sie nur das Verzeichnis an das temporäre */greengrass/v2*`/logs` Protokollverzeichnis auf dem Host binden, anstatt das gesamte Greengrass-Verzeichnis zu mounten. Weitere Informationen finden Sie unter Greengrass-Logs außerhalb des Docker-Containers beibehalten.

Um einen laufenden AWS IoT Greengrass Docker-Container zu beenden, führen Sie oder aus. `docker stop` `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 übrig gebliebenen Zombie-Prozesse hilft. Weitere Informationen finden Sie unter Spezifizieren Sie einen Init-Prozess in der Docker-Dokumentation.

Informationen zur Behebung von Problemen bei der Ausführung AWS IoT Greengrass in einem Docker-Container finden Sie unter. Fehlerbehebung AWS IoT Greengrass in einem Docker-Container

Fehlerbehebung AWS IoT Greengrass in einem Docker-Container

Verwenden Sie die folgenden Informationen, um Probleme bei der Ausführung AWS IoT Greengrass in einem Docker-Container zu beheben und Probleme AWS IoT Greengrass im Docker-Container zu debuggen.

Themen

- Behebung von Problemen beim Ausführen des Docker-Containers
- Debuggen AWS IoT Greengrass in einem Docker-Container

Behebung von Problemen beim Ausführen des Docker-Containers

Verwenden Sie die folgenden Informationen, um Probleme bei der Ausführung AWS IoT Greengrass in einem Docker-Container zu beheben.

Themen

- Fehler: Interaktive Anmeldung von einem Nicht-TTY-Gerät aus nicht möglich
- Fehler: Unbekannte Optionen: - no-include-email
- Fehler: Eine Firewall blockiert die Freigabe von Dateien zwischen Fenstern und den Containern.
- Fehler: Beim Aufrufen der GetAuthorizationToken Operation ist ein Fehler aufgetreten (AccessDeniedException): User: arn:aws:iam: :user/ account-id ist <user-name>nicht berechtigt,: ecr: on resource: * auszuführen GetAuthorizationToken
- Fehler: Sie haben Ihr Pull-Rate-Limit erreicht

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 Sie die neueste AWS CLI Version installiert haben (z. B. Run:`pip install awscli --upgrade --user`). Weitere Informationen finden Sie im AWS Command Line Interface Benutzerhandbuch AWS Command Line Interface unter Installation von unter Microsoft Windows.

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 aufgetreten (AccessDeniedException): User: arn:aws:iam: :user/ ***account-id*** ist <user-name>nicht berechtigt,: ecr: on resource: * auszuführen GetAuthorizationToken

Möglicherweise erhalten Sie diesen Fehler, wenn Sie den `aws ecr get-login-password` Befehl ausführen, wenn Sie nicht über ausreichende Berechtigungen für den Zugriff auf ein Amazon ECR-Repository verfügen. Weitere Informationen finden Sie unter Beispiele für Amazon ECR Repository-Richtlinien und Zugriff auf ein Amazon ECR-Repository im Amazon ECR-Benutzerhandbuch.

Fehler: Sie haben Ihr Pull-Rate-Limit erreicht

Docker Hub begrenzt die Anzahl der Pull-Anfragen, die anonyme und kostenlose Docker Hub-Benutzer stellen können. Wenn du die Ratenlimits für anonyme oder kostenlose Benutzer-Pull-Requests überschreitest, erhältst du einen der folgenden Fehler:

```
ERROR: toomanyrequests: Too Many Requests.
```

```
You have reached your pull rate limit.
```

Um diese Fehler zu beheben, kannst du einige Stunden warten, bevor du einen weiteren Pull-Request versuchst. Wenn Sie planen, regelmäßig eine große Anzahl von Pull-Requests einzureichen, finden Sie auf der Docker Hub-Website Informationen zu Ratenbegrenzungen und Optionen für die Authentifizierung und Aktualisierung Ihres Docker-Kontos.

Debuggen AWS IoT Greengrass in einem Docker-Container

Zum Debuggen von Problemen mit einem Docker-Container können Sie die Greengrass-Laufzeitprotokolle erhalten oder eine interaktive Shell an den Docker-Container anfügen.

Greengrass-Logs 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-Mounten des Verzeichnisses ausführen. */greengrass/v2*/logs

Um das */greengrass/v2*/logs Verzeichnis per Bind-Mount zu mounten, führen Sie einen der folgenden Schritte aus, wenn Sie einen neuen Docker-Container ausführen. AWS IoT Greengrass

- In Ihren Befehl aufnehmen-v */tmp/logs*:*/greengrass/v2*/logs:ro. docker run

  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
```

Sie können dann Ihre Logs */tmp/logs* auf Ihrem Host überprüfen, um Greengrass-Logs zu sehen, während AWS IoT Greengrass es im Docker-Container läuft.

Hinweise zum Ausführen von Greengrass-Docker-Containern finden Sie unter und AWS IoT Greengrass In Docker mit manueller Bereitstellung ausführen AWS IoT Greengrass In Docker mit automatischer Bereitstellung ausführen

Hängen Sie eine interaktive Shell an den Docker-Container an

Wenn Sie docker exec Befehle innerhalb des Docker-Containers ausführen, werden diese Befehle nicht in den Docker-Protokollen erfasst. Wenn Sie Ihre Befehle in den Docker-Protokollen protokollieren, können Sie den Status des Greengrass Docker-Containers untersuchen. Führen Sie eine der folgenden Aktionen aus:

- Führen Sie den folgenden Befehl in einem separaten Terminal aus, um die Standardeingabe, -ausgabe und -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 angehängt ist.

```
docker exec -it container-id sh -c "command > /proc/1/fd/1"
```

Allgemeine Informationen AWS IoT Greengrass zur Problembehandlung finden Sie unterFehlerbehebung.

Konfiguration der 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
- Greengrass Nucleus im Netzwerk IPv6 konfigurieren

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 erhalten Sie unter Grüngraskern und Aktualisieren Sie die 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 Sie die 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": "-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"
}
```

> **Important**
>
> Die Ausführung der AWS IoT Greengrass Core-Software mit minimaler Speicherzuweisung kann sich auf Systemen mit niedrigen Spezifikationen erheblich auf die Leistung auswirken, da die JVM bei geringerem Speicherverbrauch mehr Rechenleistung erbringt. Wir empfehlen, die Optionen so zu optimieren, dass sie Ihren Speicher- und Leistungsanforderungen entsprechen.

Diese Beispielupdates zur Zusammenführung von Konfigurationen verwenden die folgenden JVM-Optionen:

`-XX:+UseSerialGC`

Gibt an, dass die serielle Garbage-Collection für den JVM-Heap-Speicherplatz verwendet werden soll. Der serielle Garbage-Collector ist langsamer, verbraucht 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 beansprucht, verbraucht die mehrfache Verwendung des JIT-Compilers mehr Speicher als eine einzelne Kompilierung.

`-Xmx`*NN*`m`

Legt die maximale JVM-Heap-Größe fest.

> **Important**
>
> Wenn die maximale Heap-Größe zu niedrig eingestellt wird, kann dies zu einer langsameren Leistung oder zu Fehlern führen. out-of-memory Wir empfehlen, Ihre aktuelle Heap-Nutzung zu messen, bevor Sie mit der `-Xmx`*NN*`m` Option eine maximale Größe festlegen. Konfigurieren Sie Ihre JVM mit der `-XX:NativeMemoryTracking=detail` JVM-Option. Messen Sie dann Ihre aktuelle Heap-Nutzung, indem Sie die `VM.native_memory` Befehlsanfrage im jcmd Utility verwenden.

Wenn die Messung des Heaps keine Option ist, verwenden Sie ihn `-Xmx64m` als Startwert, um die Heap-Größe auf 64 MB zu begrenzen. Von dort aus können Sie dann die maximale Heap-Größe schrittweise verringern. 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. Die Menge der benötigten JVM-Heap-Größe kann auch im Laufe der Zeit variieren, je nachdem, welche Plugin-Komponenten auf dem Kerngerät bereitgestellt werden. Wenn die maximale Heap-Größe für Ihre Umgebung zu niedrig ist, kann die AWS IoT Greengrass Core-Software aufgrund unzureichenden Speichers auf unerwartete Fehler stoßen. Wenn Sie eine langsamere Leistung oder Fehler aufgrund unzureichenden Speichers feststellen, kehren Sie zu einer zweifelsfrei funktionierenden Einstellung zurück. Wenn Ihre normale Größe des festgeschriebenen Heaps

beispielsweise ist`41428KB`, verwenden Sie diese Option, `-Xmx40m` um die Heap-Nutzung leicht einzuschränken.

`-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 (möglicherweise 20-mal langsamer für Bereitstellungen auf Low-End-Systemen) als das Ausführen von JIT-kompiliertem Code. Der kompilierte Code belegt jedoch keinen Speicherplatz.

Informationen 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

Komponententyp	Konfigurieren Sie den Komponentenbenutzer
Generisch	Ja
Lambda (nicht containerisiert)	Ja
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
- Konfigurieren Sie den Standardkomponentenbenutzer

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": "2.16.1",
      "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

Komponententyp	Konfigurieren Sie Systemressourcenlimits
Generisch	Ja
Lambda (nicht containerisiert)	Ja
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": "2.16.1",
      "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 Geräteverkehr über einen Proxy oder eine Firewall zulassen.

AWS IoT Greengrass Stellt die folgenden Konfigurationsoptionen zur Verfügung, 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": "2.16.1",
      "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": "2.16.1",
      "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. Geräteverkehr über einen Proxy oder eine Firewall zulassen

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
- Konfigurieren Sie während der Installation einen Netzwerk-Proxy
- Ermöglichen Sie dem Kerngerät, einem HTTPS-Proxy zu vertrauen

- Das NetworkProxy-Objekt

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": "2.16.1",
      "configurationUpdate": {
        "merge": "{\"networkProxy\":{\"noProxyAddresses\":
\"http://192.168.0.1,www.example.com\",\"proxy\":{\"url\":\"https://my-proxy-
server:1100\",\"username\":\"Mary_Major\",\"password\":\"pass@word1357\"}}}"
      }
    }
  }
}
```

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 Kerngerät.

   - 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 Kontingente für den AWS IoT Core Message Broker-Service in der Allgemeine AWS-Referenz.

Greengrass Nucleus im Netzwerk IPv6 konfigurieren

Greengrass Nucleus spricht AWS IoT Core über APIsGreengrass mit. APIs Greengrass-Unterstützung in einer IPv6 Dual-Stack-Umgebung.

Um Dual-Stack-Endpunkte zu aktivieren für: IPv6

- Fügen Sie Systemeigenschaften `aws.useDualstackEndpoint=true` hinzu und zu `java.net.preferIPv6Addresses=true` `jvmOptions`
- Setzen Sie `s3EndpointType` auf `DUALSTACK`

Legen Sie diese Option während der Bereitstellung fest, oder stellen Sie sie manuell mit dem `--init-config` Installer-Argument bereit. Weitere Informationen finden Sie unter Verwenden von Amazon S3 S3-Dual-Stack-Endpunkten.

Example Code für die Bereitstellung:

```
{
    "jvmOptions": "-Daws.useDualstackEndpoint=true",
    "s3EndpointType":"DUALSTACK"
}
```

Example **`config.yaml`**durch manuelle Bereitstellung:

```
---
system:
  ...
services:
  aws.greengrass.Nucleus:
```

```
    ...
    configuration:
      ...
      jvmOptions: "-Daws.useDualstackEndpoint=true -Djava.net.preferIPv6Addresses=true"
      s3EndpointType: "DUALSTACK"
```

Aktualisieren Sie die AWS IoT Greengrass Core-Software (OTA)

Die AWS IoT Greengrass Core-Software besteht aus der Greengrass Nucleus-Komponente und anderen optionalen Komponenten, die Sie auf Ihren Geräten bereitstellen können, um over-the-air (OTA) -Updates der Software durchzuführen. Diese Funktion ist in die AWS IoT Greengrass Core-Software integriert.

OTA-Updates machen Folgendes effizienter:

- Schließen von Sicherheitslücken.
- Behebung von Software-Stabilitätsproblemen.
- Bereitstellung neuer oder verbesserter Funktionen.

Themen

- Voraussetzungen
- Überlegungen zu Kerngeräten
- Aktualisierungsverhalten von Greengrass Nucleus
- Führen Sie ein OTA-Update durch

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 zum verfügen, AWS 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 ausgestattet sein. AWS IoT Greengrass
- Die AWS IoT Greengrass Core-Software muss als Systemdienst eingerichtet und ausgeführt werden. OTA-Updates funktionieren nicht, wenn Sie den Nucleus über die JAR-Datei

ausführen,`Greengrass.jar`. Weitere Informationen finden Sie unter Den Greengrass Nucleus als Systemdienst konfigurieren.

Überlegungen zu Kerngeräten

Bevor Sie ein OTA-Update durchführen, sollten Sie sich der Auswirkungen auf die Kerngeräte, die Sie aktualisieren, und deren verbundene Client-Geräte bewusst sein:

- Der Greengrass-Kern wird abgeschaltet.
- Alle Komponenten, die auf dem Kerngerät laufen, werden ebenfalls heruntergefahren. Wenn diese Komponenten auf lokale Ressourcen schreiben, können sie diese Ressourcen in einem falschen Zustand belassen, sofern sie nicht ordnungsgemäß heruntergefahren werden. Komponenten können die Interprozesskommunikation verwenden, um der Nucleus-Komponente mitzuteilen, dass sie die Aktualisierung zurückstellen soll, bis sie die von ihnen verwendeten Ressourcen bereinigt haben.
- Während die Nucleus-Komponente heruntergefahren wird, verliert das Core-Gerät seine Verbindungen zu den AWS Cloud und den lokalen Geräten. Das Kerngerät leitet während des Herunterfahrens keine Nachrichten von Client-Geräten weiter.
- Langlebige Lambda-Funktionen, die als Komponenten ausgeführt werden, verlieren ihre dynamischen Statusinformationen und verwerfen alle ausstehenden Aufgaben.

Aktualisierungsverhalten von Greengrass Nucleus

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.

Wenn sich die Version der Greengrass Nucleus-Komponente ändert, wird die AWS IoT Greengrass Core-Software — die den Nucleus und alle anderen Komponenten auf Ihrem Gerät umfasst — neu gestartet, um die Änderungen zu übernehmen. Aufgrund der Auswirkungen auf die Kerngeräte, wenn die Nucleus-Komponente aktualisiert wird, möchten Sie möglicherweise kontrollieren, wann eine neue Nucleus-Patch-Version auf Ihren Geräten bereitgestellt wird. Dazu müssen Sie die Greengrass Nucleus-Komponente direkt in Ihre Bereitstellung aufnehmen. Das direkte

Einbeziehen einer Komponente bedeutet, dass Sie eine bestimmte Version dieser Komponente in Ihre Bereitstellungskonfiguration aufnehmen und sich nicht auf Komponentenabhängigkeiten verlassen, um diese Komponente auf Ihren Geräten bereitzustellen. Weitere Informationen zur Definition von Abhängigkeiten in Ihren Komponentenrezepten finden Sie unterRezeptformat.

Sehen Sie sich die folgende Tabelle an, um das Aktualisierungsverhalten für die Greengrass Nucleus-Komponente auf der Grundlage Ihrer Aktionen und Bereitstellungskonfigurationen zu verstehen.

Aktion	Bereitstellungskonfiguration	Verhalten beim Update von Nucleus
Fügen Sie neue Geräte zu einer Dinggruppe hinzu, auf die eine bestehende Bereitstellung abzielt, ohne die Bereitstellung zu überarbeiten.	Die Bereitstellung umfasst nicht direkt den Greengrass-Nukleus. Die Bereitstellung umfasst direkt mindestens eine von AWS-bereitgestellt e Komponente oder eine benutzerdefinierte Komponent e, die von einer von AWS-bereitgestellten Komponent e oder vom Greengrass-Kern abhängt.	Installiert auf neuen Geräten die neueste Patch-Version von Nucleus, die alle Anforderu ngen an die Abhängigkeit von Komponenten erfüllt. Auf vorhandenen Geräten wird die installierte Version von Nucleus nicht aktualisiert.
Fügen Sie neue Geräte zu einer Dinggruppe hinzu, auf die eine bestehende Bereitstellung abzielt, ohne die Bereitstellung zu überarbeiten.	Die Bereitstellung umfasst direkt eine bestimmte Version des Greengrass-Kerns.	Installiert auf neuen Geräten die angegebene Nucleus-V ersion. Auf vorhandenen Geräten wird die installierte Version von Nucleus nicht aktualisiert.
Erstellen Sie ein neues Deployment oder überarbei	Die Bereitstellung umfasst nicht direkt den Greengrass-Nukleus.	Installiert auf allen Zielgeräten die neueste Patch-Version des Nucleus, die alle Anforderu

Aktion	Bereitstellungskonfiguration	Verhalten beim Update von Nucleus
ten Sie ein bestehendes Deployment.	Die Bereitstellung umfasst direkt mindestens eine von AWS-bereitgestellt e Komponente oder eine benutzerdefinierte Komponent e, die von einer von AWS-bereitgestellten Komponent e oder vom Greengrass-Kern abhängt.	ngen an die Abhängigkeit von Komponenten erfüllt, auch auf allen neuen Geräten, die Sie der Zielgruppe hinzufügen.
Erstellen Sie ein neues Deployment oder überarbei ten Sie ein bestehendes Deployment.	Die Bereitstellung umfasst direkt eine bestimmte Version des Greengrass-Kerns.	Installiert auf allen Zielgerät en die angegebene Nucleus-Version, einschließlich aller neuen Geräte, die Sie der Ziel-Dinggruppe hinzufügen.

Führen Sie ein OTA-Update durch

Um ein OTA-Update durchzuführen, erstellen Sie eine Bereitstellung, die die Nucleus-Komponente und die zu installierende Version enthält.

Deinstallieren Sie die 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.

Um die AWS IoT Greengrass Core-Software zu deinstallieren

1. Wenn Sie die Software als Systemdienst ausführen, müssen Sie den Dienst beenden, 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 Dienst.

```
sudo systemctl disable greengrass.service
```

3. Entfernen Sie den Dienst.

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

4. Stellen Sie sicher, dass der Dienst 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 Dienst.

```
sc config "greengrass" start=disabled
```

3. Entfernen Sie den Dienst.

```
sc delete "greengrass"
```

4. Starten Sie das Gerät neu.

Windows (PowerShell)

> **Note**
>
> Sie müssen PowerShell als Administrator ausführen, um diese Befehle ausführen zu können.

1. Stoppen Sie den Service .

   ```
   Stop-Service -Name "greengrass"
   ```

2. Deaktivieren Sie den Dienst.

   ```
   Set-Service -Name "greengrass" -Status stopped -StartupType disabled
   ```

3. Entfernen Sie den Dienst.

   - 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 Kerngerät aus dem AWS IoT Greengrass Dienst. Dieser Schritt entfernt die Statusinformationen des Kerngeräts aus dem AWS Cloud. Schließen Sie diesen Schritt unbedingt ab, wenn Sie die AWS IoT Greengrass Core-Software auf einem Kerngerät mit demselben Namen erneut installieren möchten.
   - Gehen Sie wie folgt vor, um ein Core-Gerät von der AWS IoT Greengrass Konsole zu löschen:
     a. Navigieren Sie zur AWS IoT Greengrass -Konsole.
     b. Wählen Sie Core-Geräte.
     c. Wählen Sie das zu löschende Kerngerät aus.
     d. Wählen Sie Löschen.
     e. Wählen Sie im Bestätigungsmodus die Option Löschen aus.
   - Verwenden Sie den DeleteCoreDeviceVorgang AWS Command Line Interface, um ein Kerngerät mit dem zu löschen. Führen Sie den folgenden Befehl aus und *MyGreengrassCore* ersetzen Sie ihn durch den Namen des Kerngeräts.

     ```
     aws greengrassv2 delete-core-device --core-device-thing-name MyGreengrassCore
     ```

AWS IoT Greengrass V2 Tutorials

AWS IoT Greengrass ist ein Dienst, mit dem Sie AWS Lambda Funktionen, Modelle für maschinelles Lernen und anderen Code auf Edge-Geräten ausführen können. Auf diese Weise können Sie Daten lokal verarbeiten und so die Latenz- und Bandbreitenkosten reduzieren und gleichzeitig die sichere Kommunikation mit der Cloud aufrechterhalten.

Sie können die folgenden Tutorials absolvieren, um mehr über AWS IoT Greengrass V2 und seine Funktionen zu erfahren.

Themen

- Video-Tutorials
- Tutorial: Entwickeln Sie eine Greengrass-Komponente, die Komponenten-Updates verzögert
- Tutorial: Interagieren Sie mit lokalen IoT-Geräten über MQTT
- Tutorial: Schützen Sie Greengrass Nucleus mit Trusted Platform Module (TPM)
- Tutorial: Schützen Sie AWS IoT Greengrass Nucleus Lite mit Trusted Platform Module (TPM)
- Tutorial: Erste Schritte mit SageMaker AI Edge Manager
- Tutorial: Führen Sie mit Lite eine Inferenz zur Klassifizierung von Beispielbildern durch TensorFlow
- Tutorial: Führen Sie mit Lite eine Beispielinferenz zur Bildklassifizierung von Bildern aus einer Kamera durch TensorFlow

Video-Tutorials

Videos

Die folgende Liste enthält Video-Tutorials für AWS IoT Greengrass V2.

- AWS IoT Greengrass Nucleus Lite-Tutorial: Erste Schritte

  10. April 2025 — In diesem Tutorial erfahren Sie mehr über Greengrass Nucleus Lite und wann Sie es für Ihre AWS IoT Geräte auswählen sollten. Sie erfahren, wie Sie Nucleus Lite installieren und Ihre Komponenten darauf bereitstellen.
- AWS IoT Greengrass V2 Tutorial: Erste Schritte

  16. Dezember 2021 — Fangen Sie an mit AWS IoT Greengrass V2. In diesem Tutorial erfahren Sie, wie Sie Greengrass installieren und Ihre erste Bereitstellung abschließen.

- AWS IoT Greengrass V2 Komponentenentwicklung mit lokaler Debug-Konsole

  21. August 2023 — Erfahren Sie, wie Sie die lokale Debug-Konsole von Greengrass V2 verwenden, um Ihre Komponenten schnell zu entwickeln und zu debuggen.

- MQTT-Client-Geräte Connect mit AWS IoT Greengrass V2

  2. August 2023 — In diesem Tutorial erfahren Sie, wie Sie ein Kerngerät so konfigurieren, dass es mit lokalen AWS IoT Geräten interagiert, die über MQTT eine Verbindung zum Kerngerät herstellen.

- AWS IoT Greengrass V2 Tutorial: Verwendung von MQTT-Anwendungen

  16. Dezember 2021 — Erstellen, testen und implementieren Sie eine AWS IoT Greengrass V2 native Komponente, über die Sie MQTT veröffentlichen und abonnieren können. AWS IoT Core

Tutorial: Entwickeln Sie eine Greengrass-Komponente, die Komponenten-Updates verzögert

Sie können dieses Tutorial abschließen, um eine Komponente zu entwickeln, die over-the-air Bereitstellungsupdates verzögert. Wenn Sie Updates auf Ihren Geräten bereitstellen, sollten Sie Updates möglicherweise aufgrund von Bedingungen wie den folgenden verzögern:

- Das Gerät hat einen niedrigen Akkuladestand.
- Auf dem Gerät wird ein Prozess oder Auftrag ausgeführt, der nicht unterbrochen werden kann.
- Das Gerät hat eine eingeschränkte oder teure Internetverbindung.

> **Note**
>
> Eine Komponente ist ein Softwaremodul, das auf AWS IoT Greengrass Kerngeräten ausgeführt wird. Komponenten ermöglichen es Ihnen, komplexe Anwendungen als separate Bausteine zu erstellen und zu verwalten, die Sie von einem Greengrass-Core-Gerät auf ein anderes wiederverwenden können.

In diesem Tutorial führen Sie folgende Aufgaben aus:

1. Installieren Sie das 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 Komponenten-Updates verzögert, wenn der Akkuladestand des Kerngeräts unter einem Schwellenwert liegt. Diese Komponente abonniert Aktualisierungsbenachrichtigungen mithilfe der SubscribeToComponentUpdatesIPC-Operation. Wenn sie die Benachrichtigung erhält, prüft sie, ob der Akkuladestand unter einem einstellbaren Schwellenwert liegt. Wenn der Akkuladestand unter dem Schwellenwert liegt, wird die Aktualisierung mithilfe des DeferComponentUpdateIPC-Vorgangs für 30 Sekunden verschoben. Sie entwickeln diese Komponente auf Ihrem Entwicklungscomputer mit der GDK-CLI.

   > **Note**
   >
   > Diese Komponente liest den Akkustand aus einer Datei, die Sie auf dem Kerngerät erstellen, um eine echte Batterie nachzuahmen, sodass Sie dieses Tutorial auf einem Kerngerät ohne Batterie durchführen können.

3. Veröffentlichen Sie diese Komponente im AWS IoT Greengrass Service.
4. Stellen Sie diese Komponente AWS Cloud auf einem Greengrass-Core-Gerät bereit, um sie zu testen. Anschließend ändern Sie den virtuellen Akkuladestand auf dem Kerngerät und erstellen zusätzliche Bereitstellungen, um zu sehen, wie das Kerngerät Updates verzögert, wenn der Akkuladestand niedrig ist.

Sie können damit rechnen, 20—30 Minuten für dieses Tutorial aufzuwenden.

Voraussetzungen

Zum Durcharbeiten dieses Tutorials ist Folgendes erforderlich:

- Ein 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 Administratorrechten.
- Ein Greengrass-Core-Gerät mit einer Internetverbindung. Weitere Informationen zur Einrichtung eines Core-Geräts finden Sie unterEinrichtung 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 außerdem der Python Launcher für Windows für alle Benutzer installiert sein.

> **Important**
>
> In Windows wird Python standardmäßig nicht für alle Benutzer installiert. Wenn Sie Python installieren, müssen Sie die Installation anpassen, um sie so zu konfigurieren, dass die AWS IoT Greengrass Core-Software Python-Skripts ausführt. Wenn Sie beispielsweise den grafischen Python-Installer verwenden, gehen Sie wie folgt vor:
>
> 1. Wählen Sie Launcher für alle Benutzer installieren (empfohlen) aus.
> 2. Wählen Sie aus.Customize installation.
> 3. Wählen Sie aus.Next.
> 4. Select Install for all users.
> 5. Select Add Python to environment variables.
> 6. Wählen Sie Installieren aus.
>
> Weitere Informationen finden Sie unter Using Python on 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) wurde mit Anmeldeinformationen auf Ihrem Entwicklungscomputer installiert und konfiguriert. Weitere Informationen finden Sie unter Installation, Aktualisierung und Deinstallation von AWS CLI und Konfiguration von 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 Installation, Aktualisierung und Deinstallation von AWS CLI Version 1.

Schritt 1: Installieren Sie das Greengrass Development Kit CLI

Das 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.

Um die neueste Version der GDK CLI zu installieren

1. Führen Sie auf Ihrem Entwicklungscomputer den folgenden Befehl aus, um die neueste Version der GDK-CLI aus dem GitHubRepository 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.

   - Auf Linux-Geräten fügen Sie `/home/MyUser/.local/bin` es zu PATH hinzu und *`MyUser`* ersetzen Sie es durch den Namen Ihres Benutzers.
   - Fügen Sie auf Windows-Geräten PATH *`PythonPath`*`\\Scripts` hinzu und *`PythonPath`* ersetzen Sie es durch den Pfad zum Python-Ordner auf Ihrem Gerät.

Schritt 2: Entwickeln Sie eine Komponente, die Updates verzögert

In diesem Abschnitt entwickeln Sie eine Hello World-Komponente in Python, die Komponenten-Updates verzögert, wenn der Akkuladestand des Kerngeräts unter einem Schwellenwert liegt, den Sie bei der Bereitstellung der Komponente konfigurieren. In dieser Komponente verwenden Sie die IPC-Schnittstelle (Interprocess Communication) in der AWS IoT Device SDK Version 2 für Python. Sie verwenden den SubscribeToComponentUpdatesIPC-Vorgang, um Benachrichtigungen zu erhalten, wenn das Kerngerät eine Bereitstellung empfängt. Anschließend verwenden Sie

den DeferComponentUpdateIPC-Vorgang, um das Update je nach Akkuladestand des Geräts zu verzögern oder zu bestätigen.

Um eine Hello World-Komponente zu entwickeln, die Updates verzögert

1. Erstellen Sie auf Ihrem Entwicklungscomputer einen Ordner für den Quellcode der Komponente.

   ```
   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 benannten GDK-CLI-Konfigurationsdatei, um `gdk-config.json` Komponenten zu erstellen und zu veröffentlichen. Diese Konfigurationsdatei befindet sich im Stammverzeichnis des Komponentenordners.

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

   ```
   nano gdk-config.json
   ```

   Kopieren Sie den folgenden JSON-Code in die Datei.

   - Ersetze es *`Amazon`* durch deinen Namen.
   - *`us-west-2`*Ersetzen Sie durch den AWS-Region Ort, an dem Ihr Kerngerät betrieben wird. Die GDK-CLI veröffentlicht die Komponente darin AWS-Region.
   - Ersetzen Sie es *`greengrass-component-artifacts`* durch 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 hoch, dessen Name aus diesem Wert AWS-Region, dem und Ihrer AWS-Konto ID im folgenden Format gebildet wird:. *`bucketPrefix`*`-`*`region`*`-`*`accountId`*

     Wenn Sie beispielsweise **`greengrass-component-artifacts`** und angeben und Ihre AWS-Konto ID lautet **`us-west-2123456789012`**, verwendet die GDK-CLI den S3-Bucket mit dem Name`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 spezifiziert Folgendes:

- Die Version, die verwendet werden soll, wenn die GDK-CLI die Greengrass-Komponente im AWS IoT Greengrass Cloud-Dienst veröffentlicht. `NEXT_PATCH`gibt an, dass die nächste Patch-Version nach der neuesten im AWS IoT Greengrass Cloud-Dienst verfügbaren Version ausgewählt werden soll. Wenn die Komponente noch keine Version im AWS IoT Greengrass Cloud-Dienst hat, verwendet `1.0.0` die GDK-CLI.
- 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.
- Der AWS-Region Ort, an dem 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 Quellcode der Komponente in einer Datei mit dem Namen zu erstellen. `main.py`

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

   ```
   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 macht Folgendes:

- Liest den Akkuladestand des Kerngeräts aus einer virtuellen Batteriestatusdatei, die Sie später auf dem Kerngerät erstellen werden. Diese virtuelle Batteriestanddatei ahmt einen echten Akku nach, sodass Sie dieses Tutorial auf Kerngeräten ohne Batterie durchführen können.
- Liest Befehlszeilenargumente für den Akkugrenzwert und den Pfad zur virtuellen Batteriestanddatei. Das Komponentenrezept legt diese Befehlszeilenargumente auf der Grundlage von Konfigurationsparametern fest, sodass Sie diese Werte bei der Bereitstellung der Komponente anpassen können.
- Verwendet den IPC-Client V2 in der Version AWS IoT Device SDK 2 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 Aktualisierungsbenachrichtigungen mithilfe der IPC-Operation. SubscribeToComponentUpdates Die AWS IoT Greengrass Core-Software sendet vor und nach jeder Bereitstellung Benachrichtigungen. Die Komponente ruft bei jedem Empfang einer Benachrichtigung die folgende Funktion auf. Wenn sich die Benachrichtigung auf eine bevorstehende Bereitstellung bezieht, prüft die Komponente, ob der Akkuladestand unter einem Schwellenwert liegt. Wenn der Akkuladestand unter dem Schwellenwert liegt, verschiebt die Komponente das Update mithilfe des DeferComponentUpdateIPC-Vorgangs für 30 Sekunden. Andernfalls bestätigt die Komponente das Update, wenn der Akkuladestand 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 Greengrass Core-Software sendet keine Aktualisierungsbenachrichtigungen für lokale Bereitstellungen. Sie stellen diese Komponente daher mithilfe des AWS IoT Greengrass Cloud-Dienstes bereit, um sie zu testen.

4. Verwenden Sie einen Texteditor, um das Komponentenrezept in einer Datei mit dem Namen `recipe.json` oder `recipe.yaml` zu erstellen. 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 die Datei mit GNU Nano zu erstellen.

```
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 die Datei mit GNU Nano zu erstellen.

```
nano recipe.yaml
```

Kopieren Sie das 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 spezifiziert Folgendes:

- Standardkonfigurationsparameter für den Akkugrenzwert, den virtuellen Batteriedateipfad auf Linux-Kerngeräten und den virtuellen Batteriedateipfad auf Windows-Kerngeräten.
- Ein `install` Lebenszyklus, der die neueste Version von AWS IoT Device SDK v2 für Python installiert.
- Ein `run` Lebenszyklus, in dem die Python-Anwendung ausgeführt wird`main.py`.
- Platzhalter wie `COMPONENT_NAME` und`COMPONENT_VERSION`, bei denen 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 Sie die Komponente im AWS IoT Greengrass Service

In diesem Abschnitt veröffentlichen Sie die Hello World-Komponente im AWS IoT Greengrass Cloud-Dienst. Sobald eine Komponente im AWS IoT Greengrass Cloud-Dienst verfügbar ist, können Sie sie auf Kerngeräten bereitstellen. Sie verwenden die GDK-CLI, um die Komponente von Ihrem Entwicklungscomputer im AWS IoT Greengrass Cloud-Dienst zu veröffentlichen. Die GDK-CLI lädt das Rezept und die Artefakte der Komponente für Sie hoch.

Um die Hello World-Komponente im Service zu veröffentlichen AWS IoT Greengrass

1. Führen Sie den folgenden Befehl aus, um die Komponente mit der GDK-CLI zu erstellen. Der Befehl component build erstellt ein Rezept und Artefakte auf der Grundlage 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 sehen, die dem folgenden Beispiel ähneln.

   ```
   [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-Dienst zu veröffentlichen. Mit dem Befehl zum Veröffentlichen der Komponente wird das ZIP-Dateiartefakt der Komponente in einen S3-Bucket hochgeladen. Anschließend aktualisiert er den S3-URI der ZIP-Datei im Komponentenrezept und lädt das Rezept in den Service hoch.

AWS IoT Greengrass In diesem Prozess prüft die GDK-CLI, welche Version der Hello World-Komponente bereits im AWS IoT Greengrass Cloud-Dienst verfügbar ist, sodass sie die nächste Patch-Version nach dieser Version auswählen kann. Wenn die Komponente noch nicht existiert, verwendet die GDK-CLI die Version`1.0.0`.

```
gdk component publish
```

Sie sollten Meldungen sehen, die dem folgenden Beispiel ähneln. Die Ausgabe gibt Ihnen die Version der Komponente an, 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 S3-Bucket-Namen aus der Ausgabe. Sie verwenden den Bucket-Namen später, damit das Kerngerät Komponentenartefakte aus diesem Bucket herunterladen kann.
4. (Optional) Sehen Sie sich 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 Greengrass Konsole die Option Komponenten aus.

b. Wählen Sie auf der Seite „Komponenten“ die Registerkarte Meine Komponenten und dann aus com.example.BatteryAwareHelloWorld.

   Auf dieser Seite finden Sie die Rezeptur der Komponente und weitere Informationen zur Komponente.

5. 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. *greengrass-component-artifacts-us-west-2-123456789012*Ersetzen Sie 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`

      JSON

```
{
  "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 als Namen den Wert 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.

Schritt 4: Bereitstellen und Testen der Komponente auf einem Kerngerät

In diesem Abschnitt stellen Sie die Komponente auf dem Kerngerät bereit, um deren Funktionalität zu testen. Auf dem Kerngerät erstellen Sie die virtuelle Batteriestanddatei, um eine echte Batterie nachzuahmen. Anschließend erstellen Sie weitere Bereitstellungen und beobachten die Komponentenprotokolldateien auf dem Kerngerät, um zu sehen, ob die Komponenten Aktualisierungen zurückstellen und bestätigen.

Um die Hello World-Komponente bereitzustellen und zu testen, die Updates verzögert

1. Verwenden Sie einen Texteditor, um eine virtuelle Batteriestanddatei zu erstellen. Diese Datei ahmt eine echte Batterie nach.

- 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 die Datei mit GNU Nano zu erstellen.

```
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 Kerngerät bereit. Gehen Sie wie folgt vor:

   a. Wählen Sie im Navigationsmenü der AWS IoT Greengrass Konsole die Option Komponenten aus.

   b. Wählen Sie auf der Seite Komponenten die Registerkarte Meine Komponenten und wählen Sie dann com.example.BatteryAwareHelloWorld.

   c. Auf der com.example.BatteryAwareHelloWorldWählen Sie auf der Seite Deploy aus.

   d. Wählen Sie unter Zur Bereitstellung hinzufügen eine vorhandene Bereitstellung aus, die Sie überarbeiten möchten, oder erstellen Sie eine neue Bereitstellung und klicken Sie dann auf Weiter.

   e. Wenn Sie eine neue Bereitstellung erstellen möchten, wählen Sie das Ziel-Core-Gerät oder die Dinggruppe für die Bereitstellung aus. Wählen Sie auf der Seite „Ziel angeben“ unter Bereitstellungsziel ein Kerngerät oder eine Dinggruppe aus, und klicken Sie dann auf Weiter.

   f. Vergewissern Sie sich auf der Seite „Komponenten auswählen“, dass com.example.BatteryAwareHelloWorldDie Komponente ist ausgewählt, wählen Sie Weiter.

   g. Wählen Sie auf der Seite Komponenten konfigurieren com.example.BatteryAwareHelloWorld, und gehen Sie dann wie folgt vor:

      i. Wählen Sie Komponente konfigurieren aus.

ii. In der Konfiguration com.example.BatteryAwareHelloWorldModal, unter Konfigurationsupdate, unter Konfiguration zum Zusammenführen, geben Sie das folgende Konfigurationsupdate ein.

```
{
  "BatteryThreshold": 70
}
```

iii. Wählen Sie Bestätigen, um das Modal zu schließen, und wählen Sie dann Weiter.

h. Vergewissern Sie sich, dass auf der Seite Erweiterte Einstellungen bestätigen im Abschnitt Bereitstellungsrichtlinien unter Richtlinie für Komponenten-Updates die Option Komponenten benachrichtigen ausgewählt ist. Die Option Komponenten benachrichtigen ist standardmäßig ausgewählt, wenn Sie eine neue Einrichtung erstellen.

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

Es kann bis zu einer Minute dauern, bis die Bereitstellung abgeschlossen ist.

3. Die AWS IoT Greengrass Core-Software speichert Standarddaten von Komponentenprozessen in Protokolldateien im `logs` Ordner. Führen Sie den folgenden Befehl aus, um zu überprüfen, ob die Hello World-Komponente ausgeführt wird und Statusmeldungen ausgibt.

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 sehen, die dem folgenden Beispiel ähneln.

```
Hello, World! Battery level (50) is below threshold (70), so the component will
 defer updates.
```

> **Note**
>
> Wenn die Datei nicht existiert, 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 Kerngerät nicht berechtigt ist, die Artefakte der Komponente aus dem S3-Bucket herunterzuladen. Führen Sie den folgenden Befehl aus, um die Protokolldatei der AWS IoT Greengrass Core-Software anzuzeigen. Diese Datei enthält Protokolle vom Bereitstellungsdienst 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 auf 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 Kerngerät, um sicherzustellen, dass die Komponente das Update verzögert. Gehen Sie wie folgt vor:

   a. Wählen Sie im Navigationsmenü der AWS IoT Greengrass Konsole die Option Deployments aus.

   b. Wählen Sie die Bereitstellung aus, die Sie zuvor erstellt oder überarbeitet haben.

   c. Wählen Sie auf der Bereitstellungsseite die Option Überarbeiten aus.

   d. Wählen Sie im Modus Bereitstellung überarbeiten die Option Bereitstellung überarbeiten aus.

   e. Wählen Sie bei jedem Schritt Weiter und anschließend Bereitstellen aus.

5. Führen Sie den folgenden Befehl aus, um die Protokolle der Komponente erneut anzuzeigen und sicherzustellen, dass das Update 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 sehen, die dem folgenden Beispiel ähneln. Die Komponente verzögert das Update für 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 Akkuladestand zu bearbeiten und den Akkustand 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 die Datei mit GNU Nano zu erstellen.

   ```
   sudo nano /home/ggc_user/virtual_battery.json
   ```

   Ändern Sie den Akkuladestand auf. `80`

   ```
   {
   ```

```
  "battery_level": 80
}
```

7. Führen Sie den folgenden Befehl aus, um die Protokolle der Komponente erneut einzusehen und zu überprüfen, ob das Update 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 sehen, die den folgenden Beispielen ähneln.

   ```
   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 verzögert oder bestätigt Updates je nach Akkuladestand des Kerngeräts. Weitere Informationen zu den Themen, die in diesem Tutorial behandelt werden, finden Sie im Folgenden:

- AWS IoT Greengrass Komponenten entwickeln
- AWS IoT Greengrass Komponenten auf Geräten bereitstellen
- 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 für das Entwicklungskit

Tutorial: Interagieren Sie mit lokalen IoT-Geräten über MQTT

Sie können dieses Tutorial abschließen, um ein Kerngerät für die Interaktion mit lokalen IoT-Geräten, sogenannten Client-Geräten, zu konfigurieren, die über MQTT eine Verbindung zum Kerngerät herstellen. In diesem Tutorial konfigurieren Sie AWS IoT Dinge, die Cloud Discovery verwenden, um eine Verbindung zum Kerngerät als Client-Geräte herzustellen. Wenn Sie Cloud Discovery konfigurieren, kann ein Client-Gerät eine Anfrage an den AWS IoT Greengrass Cloud-Dienst senden, um Kerngeräte zu erkennen. Die Antwort von AWS IoT Greengrass enthält Verbindungsinformationen und Zertifikate für die Kerngeräte, 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, wo es über MQTT kommunizieren kann.

In diesem Tutorial führen Sie folgende Aufgaben aus:

1. Überprüfen und aktualisieren Sie bei Bedarf die Berechtigungen des Kerngeräts.
2. Ordnen Sie Client-Geräte dem Core-Gerät zu, sodass sie das Core-Gerät mithilfe der Cloud-Erkennung erkennen können.
3. Stellen Sie Greengrass-Komponenten auf dem Kerngerät bereit, um die Unterstützung von Client-Geräten zu aktivieren.
4. Connect die Client-Geräte mit dem Kerngerät und testen Sie die Kommunikation mit dem AWS IoT Core Cloud-Dienst.
5. Entwickeln Sie eine benutzerdefinierte Greengrass-Komponente, die mit den Client-Geräten kommuniziert.
6. Entwickeln Sie eine benutzerdefinierte Komponente, die mit den Geräteschatten der Client-Geräte AWS IoT interagiert.

In diesem Tutorial werden ein Single-Core-Gerät und ein einzelnes Client-Gerät verwendet. Sie können dem Tutorial auch folgen, um mehrere Client-Geräte zu verbinden und zu testen.

Sie können davon ausgehen, dass Sie für dieses Tutorial 30—60 Minuten aufwenden werden.

Voraussetzungen

Zum Durcharbeiten dieses Tutorials ist Folgendes erforderlich:

- Ein. 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 Administratorrechten.
- Ein Greengrass-Core-Gerät. Weitere Informationen zur Einrichtung eines Core-Geräts finden Sie unterEinrichtung von AWS IoT Greengrass Kerngeräten.
  - Auf dem Kerngerät muss Greengrass Nucleus v2.6.0 oder höher ausgeführt werden. Diese Version beinhaltet Unterstützung für Platzhalter in der lokalen publish/subscribe Kommunikation und Unterstützung für Client-Geräteschatten.

    > **Note**
    >
    > Für die Unterstützung von Client-Geräten ist Greengrass Nucleus v2.2.0 oder höher erforderlich. In diesem Tutorial werden jedoch neuere Funktionen untersucht, wie z. B. die Unterstützung für lokale MQTT-Platzhalter publish/subscribe und die Unterstützung von Client-Geräteschatten. Für diese Funktionen ist Greengrass Nucleus v2.6.0 oder höher erforderlich.

  - Das Core-Gerät muss sich im selben Netzwerk wie die Client-Geräte befinden, um eine Verbindung herstellen zu können.
  - (Optional) Um die Module zu vervollständigen, in denen Sie benutzerdefinierte Greengrass-Komponenten entwickeln, muss auf dem Kerngerät die Greengrass-CLI ausgeführt werden. Weitere Informationen finden Sie unter Installieren Sie die Greengrass-CLI.
- Eine AWS IoT Sache, die Sie in diesem Tutorial als Client-Gerät verbinden können. Weitere Informationen finden Sie im AWS IoT Core Entwicklerhandbuch unter AWS IoT Ressourcen erstellen.
  - Die AWS IoT Richtlinie des Client-Geräts muss die `greengrass:Discover` Genehmigung 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.
  - Auf dem Client-Gerät muss Python 3 ausgeführt werden.
  - Auf dem Client-Gerät muss Git ausgeführt werden.

Schritt 1: Überprüfen und aktualisieren Sie die AWS IoT Core-Geräterichtlinie

Um Client-Geräte zu unterstützen, muss die AWS IoT Richtlinie eines Kerngeräts 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 Netzwerkverbindungsinformationen des Kerngeräts an den AWS IoT Greengrass Cloud-Dienst meldet.

Weitere Informationen zu diesen Berechtigungen und AWS IoT Richtlinien für Kerngeräte finden Sie unter AWS IoT Richtlinien für den Betrieb auf Datenebene undMinimale AWS IoT Richtlinie zur Unterstützung von Client-Geräten.

In diesem Abschnitt überprüfen Sie die AWS IoT Richtlinien für Ihr Hauptgerät und fügen alle erforderlichen Berechtigungen hinzu, die noch fehlen. Wenn Sie das AWS IoT Greengrass Core-Softwareinstallationsprogramm zur Bereitstellung von Ressourcen verwendet haben, verfügt Ihr Kerngerät über eine AWS IoT Richtlinie, die den Zugriff auf alle AWS IoT Greengrass Aktionen ermöglicht (`greengrass:*`). In diesem Fall müssen Sie die AWS IoT Richtlinie nur aktualisieren, wenn Sie planen, die Shadow Manager-Komponente so zu konfigurieren, dass Geräteschatten mit synchronisiert AWS IoT Core werden. Andernfalls können Sie diesen Abschnitt überspringen.

Um die AWS IoT Richtlinien eines Kerngeräts zu überprüfen und zu aktualisieren

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 Seite mit den Details zum Kerngerät den Link zum Ding des Kerngeräts aus. Dieser Link öffnet die Seite mit den Ding-Details in der AWS IoT Konsole.
4. Wählen Sie auf der Seite mit den Ding-Details die Option Zertifikate aus.
5. Wählen Sie auf der Registerkarte Zertifikate das aktive Zertifikat der Sache aus.
6. Wählen Sie auf der Seite mit den Zertifikatsdetails die Option Richtlinien aus.
7. Wählen Sie auf der Registerkarte Richtlinien die AWS IoT Richtlinie aus, die Sie überprüfen und aktualisieren möchten. Sie können jeder Richtlinie, die mit dem aktiven Zertifikat des Kerngeräts verknüpft ist, die erforderlichen Berechtigungen hinzufügen.

> **Note**
>
> Wenn Sie das AWS IoT Greengrass Core-Softwareinstallationsprogramm zur Bereitstellung von Ressourcen verwendet haben, gibt es zwei AWS IoT Richtlinien. Wir empfehlen, dass Sie die genannte Richtlinie wählen GreengrassV2IoTThingPolicy, falls sie existiert. Kerngeräte, die Sie mit dem Schnellinstallationsprogramm erstellen, verwenden standardmäßig diesen Richtliniennamen. Wenn Sie dieser Richtlinie Berechtigungen hinzufügen, gewähren Sie diese Berechtigungen auch anderen Kerngeräten, 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 fehlenden erforderlichen Berechtigungen hinzu.
10. Um eine neue Richtlinienversion als aktive Version festzulegen, wählen Sie unter Status der Richtlinienversion die Option Die bearbeitete Version als aktive Version für diese Richtlinie festlegen aus.
11. Wählen Sie Als neue Version speichern aus.

Schritt 2: Aktivieren Sie die Unterstützung für Client-Geräte

Damit ein Client-Gerät über Cloud Discovery eine Verbindung zu einem Core-Gerät herstellen kann, müssen Sie die Geräte verknüpfen. Wenn Sie ein Client-Gerät einem Core-Gerät zuordnen, ermöglichen Sie diesem Client-Gerät, die IP-Adressen und Zertifikate des Core-Geräts abzurufen, um die Verbindung herzustellen.

Damit Client-Geräte eine sichere Verbindung zu einem Kerngerät herstellen und mit den Greengrass-Komponenten kommunizieren können AWS IoT Core, stellen Sie die folgenden Greengrass-Komponenten auf dem Kerngerät bereit:

- Authentifizierung auf Client-Geräten (`aws.greengrass.clientdevices.Auth`)

  Stellen Sie die Authentifizierungskomponente für Client-Geräte bereit, um Client-Geräte zu authentifizieren und Aktionen auf Client-Geräten zu autorisieren. Mit dieser Komponente können Ihre Geräte eine AWS IoT Verbindung zu einem Kerngerät herstellen.

  Für diese Komponente ist eine gewisse Konfiguration erforderlich, um sie verwenden zu können. Sie müssen Gruppen von Client-Geräten und die Operationen angeben, zu deren Ausführung jede

Gruppe berechtigt ist, z. B. Verbindungen herzustellen und über MQTT zu kommunizieren. Weitere Informationen finden Sie unter Konfiguration der Authentifizierungskomponente für Client-Geräte.

- MQTT 3.1.1-Broker (Moquette) (`aws.greengrass.clientdevices.mqtt.Moquette`)

  Stellen Sie die Moquette MQTT-Broker-Komponente bereit, um einen einfachen MQTT-Broker auszuführen. Der Moquette MQTT-Broker ist mit MQTT 3.1.1 kompatibel und bietet lokale Unterstützung für QoS 0, QoS 1, QoS 2, gespeicherte Nachrichten, Lastwill-Nachrichten und persistente Abonnements.

  Sie müssen diese Komponente nicht konfigurieren, um sie verwenden zu können. Sie können jedoch den Port konfigurieren, an dem diese Komponente den MQTT-Broker betreibt. Standardmäßig verwendet es Port 8883.

- MQTT-Brücke (`aws.greengrass.clientdevices.mqtt.Bridge`)

  (Optional) Stellen Sie die MQTT-Bridge-Komponente bereit, um Nachrichten zwischen Client-Geräten (lokales MQTT), lokales Publish/Subscribe und MQTT weiterzuleiten. AWS IoT Core Konfigurieren Sie diese Komponente, um Client-Geräte mit Client-Geräten von Greengrass-Komponenten zu synchronisieren AWS IoT Core und mit diesen zu interagieren.

  Für die Verwendung dieser 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 Kerngeräts automatisch an den AWS IoT Greengrass Cloud-Dienst zu melden. Sie können diese Komponente nicht verwenden, wenn Sie eine komplexe Netzwerkkonfiguration haben, z. B. eine, bei der ein Router den MQTT-Broker-Port an das Kerngerät weiterleitet.

  Sie müssen diese Komponente nicht konfigurieren, um sie verwenden zu können.

In diesem Abschnitt verwenden Sie die AWS IoT Greengrass Konsole, um Client-Geräte zuzuordnen und Client-Gerätekomponenten auf einem Kerngerät bereitzustellen.

Um die Unterstützung von Client-Geräten zu aktivieren

1. Navigieren Sie zur AWS IoT Greengrass -Konsole.
2. Wählen Sie im linken Navigationsmenü die Option Core-Geräte aus.

3. Wählen Sie auf der Seite Core-Geräte das Core-Gerät aus, für das Sie die Unterstützung von Client-Geräten aktivieren möchten.
4. Wählen Sie auf der Seite mit den Details zu den Kerngeräten die Registerkarte Client-Geräte aus.
5. Wählen Sie auf der Registerkarte Client-Geräte die Option Cloud-Discovery konfigurieren aus.

   Die Seite „Erkennung von Kerngeräten konfigurieren“ wird geöffnet. Auf dieser Seite können Sie Client-Geräte einem Core-Gerät zuordnen und Client-Gerätekomponenten bereitstellen. Auf dieser Seite wird das Kerngerät für Sie in Schritt 1: Ziel-Core-Geräte auswählen ausgewählt.

   > **Note**
   >
   > Sie können diese Seite auch verwenden, um die Kerngeräteerkennung für eine Dinggruppe zu konfigurieren. Wenn Sie diese Option wählen, können Sie Client-Gerätekomponenten auf allen Kerngeräten in einer Dinggruppe bereitstellen. Wenn Sie diese Option wählen, müssen Sie die Client-Geräte jedoch später, nachdem Sie die Bereitstellung erstellt haben, manuell jedem Kerngerät zuordnen. In diesem Tutorial konfigurieren Sie ein Single-Core-Gerät.

6. Ordnen Sie in Schritt 2: Client-Geräte zuordnen das AWS IoT Ding des Client-Geräts dem Core-Gerät zu. Auf diese Weise kann das Client-Gerät die Cloud-Erkennung verwenden, um die Konnektivitätsinformationen und Zertifikate des Kerngeräts abzurufen. Gehen Sie wie folgt vor:

   a. Wählen Sie Client-Geräte zuordnen aus.
   b. Geben Sie im Modal Client-Geräte mit Kerngerät verknüpfen den Namen des Objekts ein, AWS IoT das Sie verknüpfen möchten.
   c. Wählen Sie Hinzufügen aus.
   d. Wählen Sie Associate aus.
7. Stellen Sie in Schritt 3: Greengrass-Komponenten konfigurieren und bereitstellen Komponenten bereit, um die Unterstützung von Client-Geräten zu aktivieren. Wenn das Ziel-Core-Gerät schon einmal bereitgestellt wurde, wird diese Bereitstellung auf dieser Seite überarbeitet. Andernfalls erstellt diese Seite eine neue Bereitstellung für das Kerngerät. Gehen Sie wie folgt vor, um die Komponenten des Client-Geräts zu konfigurieren und bereitzustellen:

   a. Auf dem Kerngerät muss Greengrass Nucleus v2.6.0 oder höher ausgeführt werden, um dieses Tutorial abschließen zu können. Wenn auf dem Core-Gerät eine frühere Version ausgeführt wird, gehen Sie wie folgt vor:

i. Wählen Sie das Feld aus, um die aws.greengrass.NucleusKomponente bereitzustellen.

ii. Wählen Sie für die aws.greengrass.NucleusKomponente Konfiguration bearbeiten aus.

iii. Wählen Sie als Komponentenversion Version 2.6.0 oder höher aus.

iv. Wählen Sie Bestätigen aus.

> **Note**
>
> Wenn Sie den Greengrass Nucleus von einer früheren Nebenversion aktualisieren und auf dem Core-Gerät vom Kern AWS bereitgestellte Komponenten ausgeführt werden, die vom Nucleus abhängen, müssen Sie auch die von AWS-bereitgestellten Komponenten auf neuere Versionen aktualisieren. Sie können die Version dieser Komponenten konfigurieren, wenn Sie sich die Bereitstellung später in diesem Tutorial ansehen. Weitere Informationen finden Sie unter Aktualisieren Sie die AWS IoT Greengrass Core-Software (OTA).

b. Wählen Sie für die aws.greengrass.clientdevices.AuthKomponente Konfiguration bearbeiten aus.

c. Konfigurieren Sie im Modal „Konfiguration bearbeiten“ für die Komponente „Client-Geräteauthentifizierung“ eine Autorisierungsrichtlinie, die es Client-Geräten ermöglicht, den MQTT-Broker auf dem Kerngerät 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 Autorisierungsrichtlinie für Client-Geräte enthält. Jede Autorisierungsrichtlinie für Gerätegruppen spezifiziert eine Reihe von Aktionen und die Ressourcen, auf denen ein Client-Gerät diese Aktionen ausführen kann.

- Diese Richtlinie ermöglicht es Client-Geräten, deren Namen mit beginnen`MyClientDevice`, sich zu allen MQTT-Themen zu verbinden und zu kommunizieren. *`MyClientDevice*`*Ersetzen Sie es durch den Namen des Dings AWS IoT , das als Client-Gerät verbunden werden soll. Sie können auch einen Namen mit dem * Platzhalter angeben, der dem Namen des Client-Geräts entspricht. Der * Platzhalter muss am Ende des Namens stehen.

  Wenn Sie ein zweites Client-Gerät verbinden möchten, *`MyOtherClientDevice*`* ersetzen Sie es durch den Namen dieses Client-Geräts oder durch ein

Platzhaltermuster, das dem Namen dieses Client-Geräts entspricht. Andernfalls können Sie diesen Abschnitt der Auswahlregel entfernen oder beibehalten, der es Client-Geräten mit übereinstimmenden Namen ermöglicht, eine Verbindung herzustellen und `MyOtherClientDevice*` zu kommunizieren.

- Diese Richtlinie verwendet einen `OR` Operator, um auch Client-Geräten, deren Namen mit beginnen, die Möglichkeit zu geben`MyOtherClientDevice`, sich zu allen MQTT-Themen zu verbinden und zu kommunizieren. Sie können diese Klausel in der Auswahlregel entfernen oder sie so ändern, dass sie den Client-Geräten entspricht, für die eine Verbindung hergestellt werden soll.
- Diese Richtlinie ermöglicht es den Client-Geräten, alle MQTT-Themen zu veröffentlichen und zu abonnieren. Um bewährte Sicherheitsverfahren einzuhalten, beschränken Sie die `mqtt:subscribe` Operationen `mqtt:publish` und auf die minimale Anzahl 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 Komponente für die Client-Geräteauthentifizierung](#).

ii. Wählen Sie Bestätigen aus.

d. Wählen Sie für die aws.greengrass.clientdevices.mqtt.BridgeKomponente Konfiguration bearbeiten aus.

e. Konfigurieren Sie im Modal „Konfiguration bearbeiten“ für die MQTT-Bridge-Komponente eine Themenzuordnung, an die MQTT-Nachrichten von Client-Geräten weitergeleitet werden. AWS 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 legt fest, dass MQTT-Nachrichten zum `clients/+/hello/world` Themenfilter von Client-Geräten an den AWS IoT Core Cloud-Dienst weitergeleitet werden. 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 in dieser Region noch nicht eingerichtet haben, öffnet die Konsole ein Modal, um die Servicerolle für Sie einzurichten. Die Authentifizierungskomponente für Clientgeräte verwendet diese Servicerolle, um die Identität von Client-Geräten zu überprüfen, und die IP-Detector-Komponente verwendet diese Servicerolle, um die Verbindungsinformationen der Kerngeräte zu verwalten. Klicken Sie aufGewähren von Berechtigungenaus.
10. Wählen Sie auf der Seite Überprüfen die Option Bereitstellen aus, um die Bereitstellung auf dem Kerngerät zu starten.
11. Um zu überprüfen, ob die Bereitstellung erfolgreich ist, überprüfen Sie den Status der Bereitstellung und überprüfen Sie die Protokolle auf dem Kerngerät. Um den Status der Bereitstellung auf dem Kerngerät zu überprüfen, können Sie in der Bereitstellungsübersicht die Option Target auswählen. Weitere Informationen finden Sie hier:

    - Überprüfen Sie den Bereitstellungsstatus
    - AWS IoT Greengrass Protokolle überwachen

Schritt 3: Client-Geräte Connect

Client-Geräte können das verwenden, AWS IoT Device SDK um ein Kerngerät zu erkennen, eine Verbindung herzustellen und mit ihm zu kommunizieren. Das Client-Gerät muss ein AWS IoT Ding sein. Weitere Informationen finden Sie im AWS IoT Core Entwicklerhandbuch unter Ein Ding-Objekt erstellen.

In diesem Abschnitt installieren Sie AWS IoT Device SDK Version 2 für Python und führen die Greengrass Discovery-Beispielanwendung aus dem AWS IoT Device SDK aus.

> **Note**
>
> Das AWS IoT Device SDK ist auch in anderen Programmiersprachen verfügbar. In diesem Tutorial wird AWS IoT Device SDK Version 2 für Python verwendet, aber Sie können die andere Version SDKs für Ihren Anwendungsfall untersuchen. Weitere Informationen finden Sie unter AWS IoT Gerät SDKs im AWS IoT Core Entwicklerhandbuch.

So verbinden Sie ein Client-Gerät mit einem Core-Gerät

1. Laden Sie die AWS IoT Device SDK Version v2 für Python herunter und installieren Sie sie auf dem AWS IoT Ding, 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 for Python, um es herunterzuladen.

   ```
   git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
   ```

   b. Installieren Sie die AWS IoT Device SDK Version v2 für Python.

   ```
   python3 -m pip install --user ./aws-iot-device-sdk-python-v2
   ```

2. Wechseln Sie in den Samples-Ordner in der AWS IoT Device SDK Version v2 für Python.

   ```
   cd aws-iot-device-sdk-python-v2/samples/greengrass
   ```

3. Führen Sie die Greengrass-Discovery-Beispielanwendung aus. Diese Anwendung erwartet Argumente, die den Namen des Dings auf dem Client-Gerät, das zu verwendende MQTT-Thema und die zu verwendende Nachricht sowie die Zertifikate angeben, die die Verbindung authentifizieren und sichern. Das folgende Beispiel sendet eine Hello World-Nachricht an das `clients/MyClientDevice1/hello/world` Thema.

   > **Note**
   >
   > Dieses Thema entspricht dem Thema, in dem Sie AWS IoT Core zuvor die MQTT-Bridge für die Weiterleitung von Nachrichten konfiguriert haben.

- *MyClientDevice1*Ersetzen Sie es durch den Ding-Namen des Client-Geräts.
- *~/certs/AmazonRootCA1.pem*Ersetzen Sie durch den Pfad zum Amazon-Root-CA-Zertifikat auf dem Client-Gerät.
- *~/certs/device.pem.crt*Ersetzen Sie durch den Pfad zum Gerätezertifikat auf dem Client-Gerät.
- *~/certs/private.pem.key*Ersetzen Sie durch den Pfad zur privaten Schlüsseldatei auf dem Client-Gerät.
- *us-east-1*Ersetzen Sie 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 zehnmal und trennt die Verbindung. Außerdem abonniert sie dasselbe Thema, in dem sie Nachrichten veröffentlicht. Wenn die Ausgabe anzeigt, dass die Anwendung MQTT-Nachrichten zu diesem Thema empfangen 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 Behebung von Problemen mit der Greengrass-Erkennung.

Sie können auch die Greengrass-Protokolle auf dem Kerngerät einsehen, um zu überprüfen, ob das Client-Gerät erfolgreich eine Verbindung herstellt und Nachrichten sendet. Weitere Informationen finden Sie unter AWS IoT Greengrass Protokolle überwachen.

4. Stellen Sie sicher, dass die MQTT-Bridge die Nachrichten vom Client-Gerät an weiterleitet. AWS 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 Nachrichten auf dem Client-Gerät vom Kerngerät zu abonnieren.

   d. Wählen Sie Subscribe (Abonnieren) aus.

   e. Führen Sie die publish/subscribe Anwendung erneut auf dem Client-Gerät aus.

Der MQTT-Testclient zeigt die Nachrichten an, die Sie vom Client-Gerät zu Themen senden, die diesem Themenfilter entsprechen.

Schritt 4: Entwickeln Sie eine Komponente, die mit Client-Geräten kommuniziert

Sie können Greengrass-Komponenten entwickeln, die mit Client-Geräten kommunizieren. Komponenten verwenden Interprozesskommunikation (IPC) und die lokale publish/subscribe Schnittstelle, um auf einem Kerngerät zu kommunizieren. Um mit Client-Geräten zu interagieren, konfigurieren Sie die MQTT-Bridge-Komponente so, dass sie Nachrichten zwischen Client-Geräten und der lokalen Schnittstelle weiterleitet. publish/subscribe

In diesem Abschnitt aktualisieren Sie die MQTT-Bridge-Komponente, um Nachrichten von Client-Geräten an die lokale publish/subscribe Schnittstelle weiterzuleiten. Anschließend entwickeln Sie eine Komponente, die diese Nachrichten abonniert und die Nachrichten ausdruckt, wenn sie sie empfängt.

Um eine Komponente zu entwickeln, die mit Client-Geräten kommuniziert

1. Überarbeiten Sie die Bereitstellung auf dem Kerngerät und konfigurieren Sie die MQTT-Bridge-Komponente so, dass sie Nachrichten von Client-Geräten an Local Publish/Subscribe weiterleitet. Gehen Sie wie folgt vor:

   a. Navigieren Sie zur AWS IoT Greengrass -Konsole.

   b. Wählen Sie im linken Navigationsmenü die Option 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 Seite mit den Details zum Kerngerät die Registerkarte Client-Geräte aus.

   e. Wählen Sie auf der Registerkarte Client-Geräte die Option Cloud-Discovery konfigurieren aus.

      Die Seite „Erkennung von Kerngeräten konfigurieren“ wird geöffnet. Auf dieser Seite können Sie ändern oder konfigurieren, welche Komponenten des Client-Geräts auf dem Kerngerät bereitgestellt werden.

   f. Wählen Sie in Schritt 3 für die aws.greengrass.clientdevices.mqtt.BridgeKomponente Konfiguration bearbeiten aus.

g. Konfigurieren Sie im Modal „Konfiguration bearbeiten“ für die MQTT-Bridge-Komponente ein Themen-Mapping, das MQTT-Nachrichten von Client-Geräten an die lokale Schnittstelle weiterleitet. publish/subscribe Gehen Sie wie folgt vor:

   i. Geben Sie unter Konfiguration im Block Konfiguration zum Zusammenführen von Code die folgende Konfiguration ein. Diese Konfiguration legt fest, dass MQTT-Nachrichten zu Themen, die dem `clients/+/hello/world` Themenfilter entsprechen, von Client-Geräten an den AWS IoT Core Cloud-Dienst und den lokalen publish/subscribe Greengrass-Broker weitergeleitet werden.

```
{
  "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 Kerngerät zu starten.

j. Um zu überprüfen, ob die Bereitstellung erfolgreich ist, überprüfen Sie den Status der Bereitstellung und überprüfen Sie die Protokolle auf dem Kerngerät. Um den Status der Bereitstellung auf dem Kerngerät zu überprüfen, können Sie in der Bereitstellungsübersicht die Option Target auswählen. Weitere Informationen finden Sie hier:

   - Überprüfen Sie den Bereitstellungsstatus
   - AWS IoT Greengrass Protokolle überwachen

2. Entwickeln und implementieren Sie eine Greengrass-Komponente, die Hello World-Nachrichten von Client-Geräten abonniert. Gehen Sie wie folgt vor:

   a. Erstellen Sie Ordner für Rezepte und Artefakte auf dem Kerngerät.

      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 Artefaktordnerpfad verwenden. Geben Sie den Komponentennamen und die Version an, 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 Version 2 für Python installiert und ein Skript ausgeführt werden soll, das das Thema abonniert und Nachrichten ausgibt.

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

      ```
      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-Skript-Artefakt `hello_world_subscriber.py` mit dem folgenden Inhalt zu erstellen. Diese Anwendung verwendet den publish/subscribe IPC-Dienst, um das `clients/+/hello/world` Thema zu abonnieren und die empfangenen Nachrichten zu drucken.

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

```
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 Version 2 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 ob sie 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 Kerngerät Nachrichten empfängt.

4. Führen Sie auf dem Client-Gerät erneut die Greengrass-Discovery-Beispielanwendung aus, um Nachrichten an das Kerngerät 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. Sehen Sie sich die Komponentenprotokolle erneut an, um sicherzustellen, dass 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 Sie eine Komponente, die mit Schatten auf Client-Geräten interagiert

Sie können Greengrass-Komponenten entwickeln, die mit den Geräteschatten des AWS IoT Client-Geräts interagieren. Ein Shadow ist ein JSON-Dokument, das die aktuellen oder gewünschten Statusinformationen für ein Objekt AWS IoT speichert, z. B. für ein Client-Gerät. Benutzerdefinierte Komponenten können auf die Schatten von Client-Geräten zugreifen, um deren Status zu verwalten, auch wenn das Client-Gerät nicht mit dem Client-Gerät verbunden AWS IoT ist. 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 Kerngerät 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 Shadows der Client-Geräte aktualisiert, und Sie führen auf den Client-Geräten eine Beispielanwendung aus, die auf Shadow-Updates von der Komponente reagiert. Diese Komponente stellt eine intelligente Lichtmanagement-Anwendung dar, bei der das Kerngerät den Farbstatus der intelligenten Lampen verwaltet, die als Client-Geräte mit dem Gerät verbunden sind.

Um eine Komponente zu entwickeln, die mit Schatten von Client-Geräten interagiert

1. Überarbeiten Sie die Bereitstellung auf dem Kerngerät, um die Shadow-Manager-Komponente bereitzustellen, und konfigurieren Sie die MQTT-Bridge-Komponente so, dass sie Shadow-Nachrichten zwischen Client-Geräten und lokalem Publish/Subscribe weiterleitet, wo der Shadow Manager kommuniziert. Gehen Sie wie folgt vor:

a. Navigieren Sie zur AWS IoT Greengrass -Konsole.

b. Wählen Sie im linken Navigationsmenü die Option Kerngeräte aus.

c. Wählen Sie auf der Seite Core-Geräte das Core-Gerät aus, das Sie für dieses Tutorial verwenden.

d. Wählen Sie auf der Seite mit den Details zum Kerngerät die Registerkarte Client-Geräte aus.

e. Wählen Sie auf der Registerkarte Client-Geräte die Option Cloud-Discovery konfigurieren aus.

   Die Seite „Erkennung von Kerngeräten konfigurieren“ wird geöffnet. Auf dieser Seite können Sie ändern oder konfigurieren, welche Komponenten des Client-Geräts auf dem Kerngerät bereitgestellt werden.

f. Wählen Sie in Schritt 3 für die aws.greengrass.clientdevices.mqtt.BridgeKomponente Konfiguration bearbeiten aus.

g. Konfigurieren Sie im Modal „Konfiguration bearbeiten“ für die MQTT-Bridge-Komponente eine Themenzuordnung, die MQTT-Nachrichten zu Device-Shadow-Themen zwischen Client-Geräten und der lokalen Publish/Subscribe-Schnittstelle weiterleitet. Sie bestätigen auch, dass die Bereitstellung eine kompatible MQTT-Bridge-Version spezifiziert. Die Unterstützung von Client-Geräteshadow erfordert MQTT Bridge v2.2.0 oder höher. Gehen Sie wie folgt vor:

   i. Wählen Sie für die Komponentenversion Version 2.2.0 oder höher.

   ii. Geben Sie unter Konfiguration im Block Konfiguration zum Zusammenführen von Code die folgende Konfiguration ein. Diese Konfiguration legt fest, dass MQTT-Nachrichten zu Shadow-Themen 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.ShadowManagerKomponente 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 Kerngerät zu starten.

k. Um zu überprüfen, ob die Bereitstellung erfolgreich ist, überprüfen Sie den Status der Bereitstellung und überprüfen Sie die Protokolle auf dem Kerngerät. Um den Status der Bereitstellung auf dem Kerngerät zu überprüfen, können Sie in der Bereitstellungsübersicht die Option Target auswählen. Weitere Informationen finden Sie hier:

- Überprüfen Sie den Bereitstellungsstatus
- AWS IoT Greengrass Protokolle überwachen

2. Entwickeln und implementieren Sie eine Greengrass-Komponente, die Smart-Light-Client-Geräte verwaltet. Gehen Sie wie folgt vor:

a. Erstellen Sie einen Ordner mit den Artefakten der Komponente auf dem Kerngerät.

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 Artefaktordnerpfad verwenden. Geben Sie den Komponentennamen und die Version an, 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 Version 2 für Python installiert und ein Skript ausgeführt werden soll, das mit den Schatten von Smart-Light-Client-Geräten interagiert, um deren Farben zu verwalten.

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

   ```
   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-Skript-Artefakt `smart_light_manager.py` mit dem folgenden Inhalt zu erstellen. Diese Anwendung verwendet den Shadow-IPC-Dienst, um Shadows auf Client-Geräten abzurufen und zu aktualisieren, und den lokalen publish/subscribe IPC-Dienst, um gemeldete Shadow-Updates zu empfangen.

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

```
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 macht Folgendes:

- Liest die Konfiguration der Komponente, um die Liste der zu verwaltenden Smart-Light-Client-Geräte abzurufen.
- Abonniert Benachrichtigungen über Konfigurationsupdates mithilfe der SubscribeToConfigurationUpdate IPC-Operation. Die AWS IoT Greengrass Core-Software sendet jedes Mal Benachrichtigungen, wenn sich die Konfiguration der Komponente ändert. Wenn die Komponente eine Benachrichtigung über ein Konfigurationsupdate erhält, aktualisiert sie die Liste der von ihr verwalteten Smart-Light-Client-Geräte.
- Ruft den Schatten jedes Smart-Light-Client-Geräts auf seinen ursprünglichen Farbstatus ab.
- Stellt die Farbe jedes Smart-Light-Client-Geräts alle 15 Sekunden auf eine zufällige Farbe ein. Die Komponente aktualisiert den Dingschatten des Client-Geräts, um seine Farbe

zu ändern. Dieser Vorgang sendet ein Shadow-Delta-Ereignis über MQTT an das Client-Gerät.

- Abonniert akzeptierte Shadow-Update-Nachrichten auf der lokalen Publish/Subscribe-Schnittstelle mithilfe der IPC-Operation. SubscribeToTopic Diese Komponente empfängt diese Nachrichten, um die Farbe der einzelnen Smart-Light-Client-Geräte zu verfolgen. Wenn ein Smart-Light-Client-Gerät ein Shadow-Update erhält, sendet es eine MQTT-Nachricht, um zu bestätigen, dass es das Update erhalten hat. Die MQTT-Bridge leitet diese Nachricht an die lokale Schnittstelle weiter. publish/subscribe

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. *`MyClientDevice1`*Ersetzen Sie es durch den Ding-Namen des Client-Geräts.

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. Sehen Sie sich die Komponentenprotokolle an, um zu überprüfen, ob die Komponente erfolgreich installiert und ausgeführt wurde.

   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 zur Änderung der Farbe des Smart Light-Client-Geräts. Der Shadow-Manager empfängt die Anfrage und legt den `desired` Status des Shadows fest. Das Smart-Light-Client-Gerät läuft jedoch noch nicht, sodass sich der `reported` Status des Shadows 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 Meldungen druckt.

4. Laden Sie eine Beispielanwendung herunter und führen Sie sie aus, die Greengrass Discovery verwendet und Device Shadow-Updates abonniert. Gehen Sie auf dem Client-Gerät wie folgt vor:

   a. Verwenden Sie einen Texteditor, um ein Python-Skript `basic_discovery_shadow.py` mit dem folgenden Inhalt zu erstellen. Diese Anwendung verwendet Greengrass Discovery und Shadows, 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 die Datei mit GNU Nano zu erstellen.

      ```
      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

      # ------------------------------- ARGUMENT PARSING
       ----------------------------------------
      import argparse
      ```

```
def parse_sample_input():
    parser = argparse.ArgumentParser(
        description="AWS IoT Greengrass Discovery Shadow",
        formatter_class=argparse.ArgumentDefaultsHelpFormatter,
    )

    # Connection / TLS
    parser.add_argument("--cert", required=True, dest="input_cert",
                        help="Path to the certificate file to use during mTLS
 connection establishment")
    parser.add_argument("--key", required=True, dest="input_key",
                        help="Path to the private key file to use during mTLS
 connection establishment")
    parser.add_argument("--ca_file", dest="input_ca", help="Path to optional CA
 bundle (PEM)")

    # Shadow
    parser.add_argument("--thing_name", required=True, dest="input_thing_name",
 help="The name assigned to your IoT Thing.")
    parser.add_argument("--shadow_property", dest="input_shadow_property",
 default="color",
                        help="The name of the shadow property you want to
 change (optional, default='color'")
    parser.add_argument("--region", dest="input_region", help="The region to
 connect through.", required=True)
    parser.add_argument("--print_discover_resp_only",
 dest="input_print_discover_resp_only", default=False)

    return parser.parse_args()

args = parse_sample_input()

# ------------------------------- ARGUMENT PARSING END
 ---------------------------------------


# Using globals to simplify sample code
is_sample_done = threading.Event()
mqtt_connection = None
shadow_thing_name = args.input_thing_name
shadow_property = args.input_shadow_property

SHADOW_VALUE_DEFAULT = "off"
```

```
class LockedData:
    def __init__(self):
        self.lock = threading.Lock()
        self.shadow_value = None
        self.disconnect_called = False
        self.request_tokens = set()

locked_data = LockedData()


def on_connection_interupted(connection, error, **kwargs):
    print('connection interrupted with error {}'.format(error))


def on_connection_resumed(connection, return_code, session_present, **kwargs):
    print('connection resumed with return code {}, session present
 {}'.format(return_code, session_present))

# Try IoT endpoints until we find one that works
def try_iot_endpoints():
    for gg_group in discover_response.gg_groups:
        for gg_core in gg_group.cores:
            for connectivity_info in gg_core.connectivity:
                try:
                    print('Trying core {} at host {} port
 {}'.format(gg_core.thing_arn, connectivity_info.host_address,
 connectivity_info.port))
                    mqtt_connection = mqtt_connection_builder.mtls_from_path(
                        endpoint=connectivity_info.host_address,
                        port=connectivity_info.port,
                        cert_filepath=args.input_cert,
                        pri_key_filepath=args.input_key,

 ca_bytes=gg_group.certificate_authorities[0].encode('utf-8'),
                        on_connection_interrupted=on_connection_interupted,
                        on_connection_resumed=on_connection_resumed,
                        client_id=args.input_thing_name,
                        clean_session=False,
                        keep_alive_secs=30)

                    connect_future = mqtt_connection.connect()
                    connect_future.result()
                    print('Connected!')
                    return mqtt_connection
```

```
                except Exception as e:
                    print('Connection failed with exception {}'.format(e))
                    continue

    exit('All connection attempts failed')

# Function for gracefully quitting this sample
def exit(msg_or_exception):
    if isinstance(msg_or_exception, Exception):
        print("Exiting sample due to exception.")
        traceback.print_exception(msg_or_exception.__class__, msg_or_exception,
 sys.exc_info()[2])
    else:
        print("Exiting sample:", msg_or_exception)

    with locked_data.lock:
        if not locked_data.disconnect_called:
            print("Disconnecting...")
            locked_data.disconnect_called = True
            future = mqtt_connection.disconnect()
            future.add_done_callback(on_disconnected)

def on_disconnected(disconnect_future):
    # type: (Future) -> None
    print("Disconnected.")

    # Signal that sample is finished
    is_sample_done.set()

def on_get_shadow_accepted(response):
    # type: (iotshadow.GetShadowResponse) -> None
    try:
        with locked_data.lock:
            # check that this is a response to a request from this session
            try:
                locked_data.request_tokens.remove(response.client_token)
            except KeyError:
                return

            print("Finished getting initial shadow state.")
            if locked_data.shadow_value is not None:
                print("  Ignoring initial query because a delta event has
 already been received.")
```

```
                return

        if response.state:
            if response.state.delta:
                value = response.state.delta.get(shadow_property)
                if value:
                    print("  Shadow contains delta value '{}'.".format(value))
                    change_shadow_value(value)
                    return

            if response.state.reported:
                value = response.state.reported.get(shadow_property)
                if value:
                    print("  Shadow contains reported value
 '{}'.".format(value))

 set_local_value_due_to_initial_query(response.state.reported[shadow_property])
                    return

        print("  Shadow document lacks '{}' property. Setting
 defaults...".format(shadow_property))
        change_shadow_value(SHADOW_VALUE_DEFAULT)
        return

    except Exception as e:
        exit(e)

def on_get_shadow_rejected(error):
    # type: (iotshadow.ErrorResponse) -> None
    try:
        # check that this is a response to a request from this session
        with locked_data.lock:
            try:
                locked_data.request_tokens.remove(error.client_token)
            except KeyError:
                return

        if error.code == 404:
            print("Thing has no shadow document. Creating with defaults...")
            change_shadow_value(SHADOW_VALUE_DEFAULT)
        else:
            exit("Get request was rejected. code:{} message:'{}'".format(
                error.code, error.message))
```

```
    except Exception as e:
        exit(e)

def on_shadow_delta_updated(delta):
    # type: (iotshadow.ShadowDeltaUpdatedEvent) -> None
    try:
        print("Received shadow delta event.")
        if delta.state and (shadow_property in delta.state):
            value = delta.state[shadow_property]
            if value is None:
                print("  Delta reports that '{}' was deleted. Resetting
 defaults...".format(shadow_property))
                change_shadow_value(SHADOW_VALUE_DEFAULT)
                return
            else:
                print("  Delta reports that desired value is '{}'. Changing
 local value...".format(value))
                if (delta.client_token is not None):
                    print ("  ClientToken is: " + delta.client_token)
                change_shadow_value(value, delta.client_token)
        else:
            print("  Delta did not report a change in
 '{}'".format(shadow_property))

    except Exception as e:
        exit(e)

def on_publish_update_shadow(future):
    #type: (Future) -> None
    try:
        future.result()
        print("Update request published.")
    except Exception as e:
        print("Failed to publish update request.")
        exit(e)

def on_update_shadow_accepted(response):
    # type: (iotshadow.UpdateShadowResponse) -> None
    try:
        # check that this is a response to a request from this session
        with locked_data.lock:
            try:
                locked_data.request_tokens.remove(response.client_token)
            except KeyError:
```

```
                return

        try:
            if response.state.reported != None:
                if shadow_property in response.state.reported:
                    print("Finished updating reported shadow value to
 '{}'.".format(response.state.reported[shadow_property])) # type: ignore
                else:
                    print ("Could not find shadow property with name:
 '{}'.".format(shadow_property)) # type: ignore
            else:
                print("Shadow states cleared.") # when the shadow states are
 cleared, reported and desired are set to None
        except:
            exit("Updated shadow is missing the target property")

    except Exception as e:
        exit(e)

def on_update_shadow_rejected(error):
    # type: (iotshadow.ErrorResponse) -> None
    try:
        # check that this is a response to a request from this session
        with locked_data.lock:
            try:
                locked_data.request_tokens.remove(error.client_token)
            except KeyError:
                return

        exit("Update request was rejected. code:{} message:'{}'".format(
            error.code, error.message))

    except Exception as e:
        exit(e)

def set_local_value_due_to_initial_query(reported_value):
    with locked_data.lock:
        locked_data.shadow_value = reported_value

def change_shadow_value(value, token=None):
    with locked_data.lock:
        if locked_data.shadow_value == value:
            print("Local value is already '{}'.".format(value))
            return
```

```
        print("Changed local shadow value to '{}'.".format(value))
        locked_data.shadow_value = value

        print("Updating reported shadow value to '{}'...".format(value))

        reuse_token = token is not None
        # use a unique token so we can correlate this "request" message to
        # any "response" messages received on the /accepted and /rejected
topics
        if not reuse_token:
            token = str(uuid4())

        # if the value is "clear shadow" then send a UpdateShadowRequest with
None
        # for both reported and desired to clear the shadow document
completely.
        if value == "clear_shadow":
            tmp_state = iotshadow.ShadowState(reported=None, desired=None,
reported_is_nullable=True, desired_is_nullable=True)
            request = iotshadow.UpdateShadowRequest(
                thing_name=shadow_thing_name,
                state=tmp_state,
                client_token=token,
            )
        # Otherwise, send a normal update request
        else:
            # if the value is "none" then set it to a Python none object to
            # clear the individual shadow property
            if value == "none":
                value = None

            request = iotshadow.UpdateShadowRequest(
            thing_name=shadow_thing_name,
            state=iotshadow.ShadowState(
                reported={ shadow_property: value }
                ),
                client_token=token,
            )

        future = shadow_client.publish_update_shadow(request,
mqtt.QoS.AT_LEAST_ONCE)

        if not reuse_token:
```

```
            locked_data.request_tokens.add(token)

        future.add_done_callback(on_publish_update_shadow)


if __name__ == '__main__':
    tls_options =
 io.TlsContextOptions.create_client_with_mtls_from_path(args.input_cert,args.input_key)
    if args.input_ca:
        tls_options.override_default_trust_store_from_path(None,
 args.input_ca)
    tls_context = io.ClientTlsContext(tls_options)

    socket_options = io.SocketOptions()

    print('Performing greengrass discovery...')
    discovery_client =
 DiscoveryClient(io.ClientBootstrap.get_or_create_static_default(),
 socket_options, tls_context, args.input_region)
    resp_future = discovery_client.discover(args.input_thing_name)
    discover_response = resp_future.result()

    print(discover_response)
    if args.print_discover_resp_only:
        exit(0)

    mqtt_connection = try_iot_endpoints()
    shadow_client = iotshadow.IotShadowClient(mqtt_connection)

    try:
        # Subscribe to necessary topics.
        # Note that is **is** important to wait for "accepted/rejected"
 subscriptions
        # to succeed before publishing the corresponding "request".
        print("Subscribing to Update responses...")
        update_accepted_subscribed_future, _ =
 shadow_client.subscribe_to_update_shadow_accepted(

 request=iotshadow.UpdateShadowSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_update_shadow_accepted)

        update_rejected_subscribed_future, _ =
 shadow_client.subscribe_to_update_shadow_rejected(
```

```
request=iotshadow.UpdateShadowSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_update_shadow_rejected)

        # Wait for subscriptions to succeed
        update_accepted_subscribed_future.result()
        update_rejected_subscribed_future.result()

        print("Subscribing to Get responses...")
        get_accepted_subscribed_future, _ =
shadow_client.subscribe_to_get_shadow_accepted(

request=iotshadow.GetShadowSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_get_shadow_accepted)

        get_rejected_subscribed_future, _ =
shadow_client.subscribe_to_get_shadow_rejected(

request=iotshadow.GetShadowSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_get_shadow_rejected)

        # Wait for subscriptions to succeed
        get_accepted_subscribed_future.result()
        get_rejected_subscribed_future.result()

        print("Subscribing to Delta events...")
        delta_subscribed_future, _ =
shadow_client.subscribe_to_shadow_delta_updated_events(

request=iotshadow.ShadowDeltaUpdatedSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_shadow_delta_updated)

        # Wait for subscription to succeed
        delta_subscribed_future.result()

        # The rest of the sample runs asynchronously.

        # Issue request for shadow's current state.
        # The response will be received by the on_get_accepted() callback
        print("Requesting current shadow state...")
```

```
        with locked_data.lock:
            # use a unique token so we can correlate this "request" message to
            # any "response" messages received on the /accepted and /rejected
topics
            token = str(uuid4())

            publish_get_future = shadow_client.publish_get_shadow(

request=iotshadow.GetShadowRequest(thing_name=shadow_thing_name,
client_token=token),
                qos=mqtt.QoS.AT_LEAST_ONCE)

            locked_data.request_tokens.add(token)

        # Ensure that publish succeeds
        publish_get_future.result()

    except Exception as e:
        exit(e)

    # Wait for the sample to finish (user types 'quit', or an error occurs)
    is_sample_done.wait()
```

Diese Python-Anwendung macht Folgendes:

- Verwendet Greengrass Discovery, um das Kerngerät zu erkennen und eine Verbindung herzustellen.
- Fordert das Shadow-Dokument vom Kerngerät an, um den ursprünglichen Status der Eigenschaft abzurufen.
- Abonniert Shadow-Delta-Ereignisse, die das Kerngerät sendet, wenn der Wert der Eigenschaft von ihrem `desired` `reported` Wert abweicht. Wenn die Anwendung ein Shadow-Delta-Ereignis empfängt, ändert sie den Wert der Eigenschaft und sendet eine Aktualisierung an das Kerngerät, um den neuen Wert als `reported` Wert festzulegen.

Diese Anwendung kombiniert die Greengrass-Entdeckung und die Schattenproben aus der AWS IoT Device SDK Version 2.

b. Führen Sie die Beispielanwendung aus. Diese Anwendung erwartet Argumente, die den Namen des Dings auf dem Client-Gerät, die zu verwendende Shadow-Eigenschaft und die Zertifikate angeben, mit denen die Verbindung authentifiziert und gesichert wird.

- *MyClientDevice1*Ersetzen Sie es durch den Ding-Namen des Client-Geräts.
- *~/certs/AmazonRootCA1.pem*Ersetzen Sie durch den Pfad zum Amazon-Root-CA-Zertifikat auf dem Client-Gerät.
- *~/certs/device.pem.crt*Ersetzen Sie durch den Pfad zum Gerätezertifikat auf dem Client-Gerät.
- *~/certs/private.pem.key*Ersetzen Sie durch den Pfad zur privaten Schlüsseldatei auf dem Client-Gerät.
- *us-east-1*Ersetzen Sie 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 Shadow-Themen und wartet auf den Empfang von Shadow-Delta-Ereignissen vom Kerngerät. Wenn die Ausgabe anzeigt, dass die Anwendung Shadow-Delta-Ereignisse empfängt und darauf reagiert, kann das Client-Gerät erfolgreich mit seinem Shadow auf dem Kerngerät 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 Behebung von Problemen mit der Greengrass-Erkennung.

Sie können auch die Greengrass-Protokolle auf dem Kerngerät einsehen, um zu überprüfen, ob das Client-Gerät erfolgreich eine Verbindung herstellt und Nachrichten sendet. Weitere Informationen finden Sie unter AWS IoT Greengrass Protokolle überwachen.

5. Sehen Sie sich die Komponentenprotokolle erneut an, um sicherzustellen, dass die Komponente Shadow-Update-Bestätigungen 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 Meldungen, 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 Shadow des Client-Geräts jedoch nicht mit AWS IoT Core. Sie können einen Shadow mit synchronisieren AWS IoT Core , um beispielsweise den Status aller Geräte in Ihrer Flotte anzuzeigen oder zu ändern. Weitere Informationen zur Konfiguration der Shadow-Manager-Komponente zum Synchronisieren von Schatten finden Sie unterSynchronisieren Sie lokale Geräteschatten mit AWS IoT Core. 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 Greengrass-Komponenten und empfängt Shadow-Updates vom Core-Gerät. Weitere Informationen zu den in diesem Tutorial behandelten Themen finden Sie im Folgenden:

- Client-Geräte zuordnen
- Endpunkte von Kerngeräten verwalten
- Testen Sie die Kommunikation mit Client-Geräten
- Greengrass-Entdeckungs-API RESTful
- MQTT-Nachrichten zwischen Client-Geräten weiterleiten und AWS IoT Core
- Interagieren Sie mit Client-Geräten in Komponenten
- Interagiere mit Geräteschatten
- Interagieren Sie mit Schatten auf Client-Geräten und synchronisieren Sie sie

Tutorial: Schützen Sie Greengrass Nucleus mit Trusted Platform Module (TPM)

> **Note**
>
> Der Mechanismus in diesem Tutorial wird nur mit unterstützt. Installieren Sie die AWS IoT Greengrass Core-Software mit manueller Ressourcenbereitstellung

Diese Tutorials enthalten Anweisungen zur Verwendung des TPM2 Chips als Hardware-Sicherheitsmodul (HSM) zur Erstellung eines privaten Schlüssels und einer CSR. Welches ist es gewohnt. Erstellen Sie das Ding-Zertifikat

In diesen Tutorials erfahren Sie, wie Sie die Gerätesicherheit verbessern können, indem Sie die AWS IoT Greengrass Core-Software (Greengrass Nucleus) mit einem Trusted Platform Module (TPM) über die PKCS #11 -Schnittstelle konfigurieren. Diese TPM-Integration stellt sicher, dass private Schlüssel und Zertifikate, die für die Geräteidentität und die Verbindung verwendet werden, sicher auf AWS IoT Core manipulationssicherer Hardware gespeichert werden, sodass eine Entnahme aufgrund von Identitätsmissbrauch oder anderen böswilligen Aktivitäten verhindert wird.

Wenn Sie diese Integration abgeschlossen haben, verwendet Ihr Greengrass-Core-Gerät TPM-geschützte private Schlüssel für seine Identität und Kommunikation mit Diensten. AWS IoT

Weitere Informationen zur Sicherheit auf Greengrass-Geräten finden Sie unterSicherheit in AWS IoT Greengrass.

Voraussetzungen

Zum Durcharbeiten dieses Tutorials ist Folgendes erforderlich:

- Ein Linux-kompatibles Gerät mit TPM 2.0-Hardware oder Firmware TPM 2.0.
- Die Anweisungen in diesem Tutorial sind für Ubuntu 24.04 LTS definiert.
  - Jede Linux-Distribution, die den TPM2 Linux-Software-Stack unterstützen kann, kann diesen Mechanismus unterstützen.
- Ein Entwicklercomputer, der AWS CLI installiert und konfiguriert ist und über folgende Berechtigungen verfügt:
  - AWS IoT Ressourcen erstellen und verwalten

  - Erstellen und verwalten Sie IAM-Rollen und -Richtlinien
- Java Runtime Environment (JRE) Version 8 oder höher ist auf Ihrem Gerät installiert.
- Die folgenden Softwarepakete sind auf Ihrem Gerät installiert:
  - curl
  - jq
- Root- oder Sudo-Rechte auf dem Gerät.

Schritt 1: Installieren Sie TPM2 Tools und Abhängigkeiten

In diesem Schritt installieren Sie die erforderlichen TPM2 Softwaretools und Bibliotheken.

1. Aktualisieren Sie Ihren Paketmanager und installieren Sie die TPM2 Tools und Abhängigkeiten, indem Sie den folgenden Befehl ausführen.

```
sudo apt-get update && sudo apt-get install tpm2-tools \
tpm2-abrmd \
tpm2-tss-engine-tools \
gnutls-bin \
libtpm2-pkcs11-1 \
libtpm2-pkcs11-tools \
libtpm2-pkcs11-1-dev \
python3-tpm2-pkcs11-tools \
libengine-pkcs11-openssl \
libtss2-tcti-tabrmd0
```

2. Installieren Sie die TPM2 OpenSSL-Provider-Pakete auf Ubuntu 24.04, die OpenSSL Engine 3 verwenden.

```
sudo apt-get install tpm2-openssl
```

Schritt 2: Initialisieren Sie den PKCS #11 -Speicher und erstellen Sie einen Slot

1. Erstellen Sie ein Verzeichnis zum Speichern von Daten.

```
sudo mkdir -p /etc/tpm2_pkcs11
```

2. Legen Sie den Speicherort als Umgebungsvariable fest. Weitere Informationen zur Speicherhierarchie finden Sie unter Initialisierung.

```
export TPM2_PKCS11_STORE=/etc/tpm2_store
```

3. Initialisieren Sie das TPM2 Token mit dem primären Objekt.

```
sudo tpm2_ptool init
```

Die verfügbaren Optionen sind die folgenden:

`hierarchy-auth HIERARCHY_AUTH`

Das Autorisierungskennwort für das Hinzufügen eines primären Objekts zur Hierarchie.

`primary-auth PRIMARY_AUTH`

Autorisierungswert für ein vorhandenes Primärschlüsselobjekt.

Der Standardwert ist ein leerer Authentifizierungswert.

`primary-handle [PRIMARY_HANDLE]`

Verwenden Sie ein vorhandenes Primärschlüsselobjekt.

Standard: `0x81000001`

`transient-parents`

Verwenden Sie ein transientes Primärobjekt einer bestimmten Vorlage.

Werte:`tpm2-tools-default`,, `tpm2-tools-ecc-default` `tss2-engine-key`

`path PATH`

Der Speicherort des Speicherverzeichnisses. Falls angegeben, muss das Verzeichnis existieren. Wenn nicht angegeben, führt es eine Suche durch, indem es die Umgebungsvariable betrachtet`TPM2_PKCS11_STORE`. Wenn diese Umgebungsvariable nicht gesetzt ist, sucht sie nach`/etc/tpm2_pkcs11`. Wenn dieses Verzeichnis nicht gefunden wird oder nicht erstellt werden kann, wird standardmäßig das aktuelle Arbeitsverzeichnis verwendet.

Schritt 3: Erstellen Sie ein Token und einen Schlüssel

1. Erstellen Sie ein PKCS #11 -Token.

```
sudo tpm2_ptool addtoken –pid=1 –userpin=USERPIN –sopin=SOPIN –label=greengrass
```

Die verfügbaren Optionen sind die folgenden:

`--pid PID`

Die primäre Objekt-ID, die mit diesem Token verknüpft werden soll.

`--sopin SOPIN`

Die Administrator-PIN. Diese PIN wird für die Objektwiederherstellung verwendet.

`--userpin USERPIN`

Die Benutzer-PIN. Diese PIN wird zur Authentifizierung für die Verwendung von Objekten verwendet.

`--label LABEL`

Ein eindeutiges Etikett zur Identifizierung des verwendeten Profils muss eindeutig sein.

`--hierarchy-auth HIERARCHY_AUTH`

2. Erstellen Sie ein ECC-Schlüsselobjekt.

```
sudo tpm2_ptool addkey –algorithm=ecc256 –label=greengrass –userpin=****** –key-
label=greenkey
```

`--label LABEL`

Das Token-Label, um auch den Schlüssel zu importieren.

`--key-label KEY_LABEL`

Die Bezeichnung des importierten Schlüssels. Der Standardwert ist ein ganzzahliger Wert.

`--id ID`

Die Schlüssel-ID. Der Standardwert ist eine zufällige Hexadezimalzahl von 8 Byte.

`--attr-always-authenticate`

Setzt das CKA_ALWAYS_AUTHENTICATE-Attribut auf CK_TRUE.

`--hierarchy-auth HIERARCHY_AUTH`

Die Hierarchyauth, die für transiente Objekte erforderlich ist.

`--sopin SOPIN`

Die Administrator-PIN.

`--userpin USERPIN`

Die Benutzer-PIN.

`--algorithm`

{rsa1024, rsa2048, rsa3072, rsa4096, aes128, aes256, ecc224, ecc256, ecc384, ecc521, hmac:sha1, hmac:sha256, hmac:sha384, hmac:sha512}

Der Typ des Schlüssels.

3. Exportieren Sie das TPM2 -TSS-Objekt aus dem Token, um die Authentifizierungsdaten zu erfassen.

```
yaml_ecc0=$(sudo tpm2_ptool export –label="greengrass" –key-label="greenkey" –
userpin="******")
```

Beispielausgabe:

```
> echo $yaml_ecc0
object-auth: 706c1cad8a5238871b30149705255926
primary-object:
  auth: ''
  hierarchy: owner
  is_transient: false
```

> **Note**
>
> Sie finden auch eine Datei mit dem Namen `greenkey.pem`, bei der es sich um den TSS2 privaten Schlüssel handelt, in dem Verzeichnis, in dem Sie diesen Befehl ausgeführt haben. Verwenden Sie dies, um die CSR mit dem tpm2 openssl Provider zu generieren. Die TSS2 PRIVATE KEY-Datei ist durch das TPM geschützt und kann nicht auf einem anderen Computer verwendet werden. Weitere Informationen zu TSS2 Schlüsseln mit OpenSSL finden Sie unter Speichern des privaten oder öffentlichen Schlüssels.

4. Erfassen Sie die Authentifizierungsdaten für den privaten TSS-Schlüssel.

```
auth_ecc0=$(echo "$yaml_ecc0" | grep "object-auth" | cut -d' ' -f2-)
```

Schritt 4: Generieren Sie eine Zertifikatsignieranforderung (CSR)

In diesem Schritt verwenden Sie den TPM2 -geschützten privaten Schlüssel, um eine CSR zu generieren.

1. Generieren Sie mithilfe des Anbieters eine CSR. TPM2

```
sudo openssl req -new -provider tpm2 -provider base -key greenkey.pem -passin "pass:
$auth_ecc0" -out "$H"$HOSTNAME".csr
```

   Wenn Sie dazu aufgefordert werden, geben Sie die erforderlichen Informationen für Ihre CSR ein, einschließlich:

   - Name des Landes (2-Buchstaben-Code)

- State or Province Name
- Ortsname
- Name der Organisation
- Organizational Unit Name
- Common Name
- Email Address

Alternativ können Sie eine OpenSSL-Konfigurationsdatei für die unbeaufsichtigte Generierung bereitstellen. Weitere Informationen finden Sie in der OpenSSL-Dokumentation.

2. Wenn Sie die CSR nicht auf demselben Computer generieren, kopieren Sie die generierte CSR auf einen Computer, für den Anmeldeinformationen konfiguriert sind. AWS

Schritt 5: Erstellen Sie das Ding-Zertifikat

Erstellen Sie ein AWS IoT Ding-Zertifikat. Weitere Informationen zum Erstellen eines Ding-Zertifikats finden Sie unterErstellen Sie das Ding-Zertifikat.

Schritt 6: Importieren Sie das Ding-Zertifikat in TPM

1. Kopieren Sie das Ding-Zertifikat auf das Gerät.
2. Fügen Sie das Ding-Zertifikat dem Greengrass-Token hinzu.

```
sudo tpm2_ptool addcert --label=greengrass --key-label=greenkey device.pem.crt
```

Die verfügbaren Optionen sind die folgenden:

`--help`

Diese Hilfemeldung anzeigen und beenden.

`--label LABEL`

Die Profilbezeichnung, die entfernt werden soll.

`--key-label KEY_LABEL`

Die zugehörige Bezeichnung für den privaten Schlüssel.

`--key-id KEY_ID`

Die zugehörige private Schlüssel-ID im Hexadezimalformat.

`cert`

Das x509-PEM-Zertifikat, das hinzugefügt werden soll.

Schritt 7: Erfassen Sie die PKCS #11 -Objekt-URL

Wir werden die aus dem `gnutls-bin` Paket `p11tool` bereitgestellten Daten verwenden, um die PKCS #11 -Token-URL und das Objekt abzurufen. URLs

1. Erfassen Sie die Token-URL für das Greengrass-Token.

```
TOKEN=sudo p11tool --list-token-urls | grep "token=greengrass"
```

2. Ruft das Objekt URLs für das Greengrass-Token ab. Verwenden Sie dieselbe PIN, die Sie in Schritt 3 verwendet haben.

```
sudo p11tool --login --list-all "${TOKEN}"
```

Beispielausgabe:

```
Token 'greengrass' with URL
 'pkcs11:model=SLB9672%00%00%00%00%00%00%00%00;manufacturer=Infineon;serial=0000000000000
 requires user PIN
Enter PIN:
WARNING: Needed CKA_VALUE but didn't find encrypted blob
Object 0:
        URL:
 pkcs11:model=SLB9672%00%00%00%00%00%00%00%00;manufacturer=Infineon;serial=00000000000000
%39%32%37%65%61%64%61%39%31%32%35%31%61%35%37%31;object=greenkey;type=private
        Type: Private key (EC/ECDSA-SECP256R1)
        Label: greenkey
        Flags: CKA_PRIVATE; CKA_NEVER_EXTRACTABLE; CKA_SENSITIVE;
        ID: 39:32:37:65:61:64:61:39:31:32:35:31:61:35:37:31

Object 1:
```

```
        URL:
 pkcs11:model=SLB9672%00%00%00%00%00%00%00%00%00;manufacturer=Infineon;serial=0000000000000
%39%32%37%65%61%64%61%39%31%32%35%31%61%35%37%31;object=greenkey;type=public
        Type: Public key (EC/ECDSA-SECP256R1)
        Label: greenkey
        ID: 39:32:37:65:61:64:61:39:31:32:35:31:61:35:37:31

Object 2:
        URL:
 pkcs11:model=SLB9672%00%00%00%00%00%00%00%00%00;manufacturer=Infineon;serial=0000000000000
%39%32%37%65%61%64%61%39%31%32%35%31%61%35%37%31;object=greenkey;type=cert
        Type: X.509 Certificate (EC/ECDSA-SECP256R1)
        Expires: Fri Dec 31 18:59:59 2049
        Label: greenkey
        ID: 39:32:37:65:61:64:61:39:31:32:35:31:61:35:37:31
```

3. Erfassen Sie die Objekt-URL für den privaten Schlüssel und das Zertifikat.

Schritt 8: Greengrass mit TPM2 Support konfigurieren und installieren

1. Konfigurieren Sie das Ding-Zertifikat. Weitere Informationen finden Sie unter Das Ding-Zertifikat konfigurieren.
2. Folgen Sie den Anweisungen zur Installation der AWS IoT Greengrass Core-Software mit dem privaten Schlüssel und dem Zertifikat in einem HSM inInstallieren Sie die Core-Software AWS IoT Greengrass. Gehen Sie dann wie folgt vor, um Ihre Installation so zu konfigurieren, dass sie TPM2 über die PKCS #11 -Schnittstelle genutzt werden kann.
3. Stellen Sie sicher, dass Sie die PKCS #11 -Provider-Komponente heruntergeladen und in Ihrem Greengrass-Installationsverzeichnis gespeichert haben.
4. Verwenden Sie einen Texteditor, um eine Konfigurationsdatei mit dem Namen zu erstellen, die dem Installationsprogramm `config.yaml` 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
   ```

5. Kopieren Sie den folgenden YAML-Inhalt in die Datei. Diese Teilkonfigurationsdatei spezifiziert Systemparameter, Greengrass-Nukleus-Parameter und PKCS #11 -Anbieterparameter.

   ```
   ---
   ```

```
system:
  certificateFilePath: "pkcs11:model=SW%20%20%20TPM
%00%00%00%00%00%00%00%00;manufacturer=IBM;serial=0000000000000000;token=greengrass;id=
%34%35;object=greenkey;type=cert"
  privateKeyPath: "pkcs11:model=SW%20%20%20TPM
%00%00%00%00%00%00%00%00;manufacturer=IBM;serial=0000000000000000;token=greengrass;id=
%34%35;object=greenkey;type=private"
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
  rootpath: "/greengrass/v2"
  thingName: "myThing"
services:
  aws.greengrass.Nucleus:
    componentType: "NUCLEUS"
    version: "2.14.0"
    configuration:
      awsRegion: "us-east-1"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
  aws.greengrass.crypto.Pkcs11Provider:
    configuration:
      name: "tpm2_pkcs11"
      library: "/usr/lib/x86_64-linux-gnu/pkcs11/libtpm2_pkcs11.so"
      slot: 1
      userPin: "123456"
```

6. Bearbeiten Sie die Datei mit den für Ihre Installation spezifischen Parametern.

   a. Aktualisieren Sie certificateFilePath und privateKeyPath mit dem Update certificateFilePath und privateKeyPath mit der in Schritt 7 erfassten PKCS #11 -URL.

   b. Aktualisieren Sie Ihr iotDataEndpoint und iotCredEndpoint basierend auf Ihrem AWS IoT Endpunkt.

   c. Aktualisieren Sie in der `aws.greengrass.crypto.Pkcs11Provider` Konfiguration die Bibliothek basierend auf Ihrer Plattform.

   > **Note**
   >
   > Das gezeigte Beispiel bezieht sich auf X86_64. Der Dateipfad wird für das Gerät ähnlich sein. ARM64

7. Schließen Sie die Greengrass-Installationsschritte unter ab. Installieren Sie die Core-Software AWS IoT Greengrass

Schritt 9: Überprüfen Sie die Installation

In diesem Schritt überprüfen Sie, ob Greengrass mit der TPM2 Integration ordnungsgemäß ausgeführt wird.

1. Überprüfen Sie den Greengrass-Servicestatus.

```
sudo systemctl status greengrass.service
```

2. Sehen Sie sich die Greengrass-Protokolle an, um sicherzustellen, dass keine Fehler vorliegen.

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

3. Vergewissern Sie sich, dass Ihr Gerät in der AWS IoT Konsole als verbunden angezeigt wird.
   a. Melden Sie sich bei der AWS IoT Greengrass -Konsole an.
   b. Erweitern Sie unter Verwalten die Option Greengrass-Geräte und wählen Sie Core-Geräte aus.
   c. Vergewissern Sie sich, dass Ihr Gerät angeschlossen ist. Der Gerätestatus zeigt an`HEALTHY`, ob es verbunden ist. Weitere Informationen finden Sie unter Überprüfen Sie den Status des Greengrass Core-Geräts.

Fehlerbehebung

Wenn Sie bei der Einrichtung oder dem Betrieb Ihres TPM2 -fähigen Greengrass-Geräts auf Probleme stoßen, versuchen Sie es mit den folgenden Schritten zur Fehlerbehebung.

- Überprüfen Sie die Haupt-Protokolldatei von Greengrass.

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

- Überprüfen Sie die PKCS #11 -Anbieterkonfiguration.

```
sudo cat /greengrass/v2/config/effectiveConfig.yaml
```

- Stellen Sie sicher, dass der TPM2 Dienst läuft.

```
sudo systemctl status tpm2-abrmd.service
```

- Stellen Sie sicher, dass auf den TPM2 Schlüssel zugegriffen werden kann.

```
sudo pkcs11-tool –module /usr/lib/x86_64-linux-gnu/pkcs11/libtpm2_pkcs11.so -l -p
 123456 –list-objects
```

- Wenn Ihr Betriebssystem mit vollständiger Festplattenverschlüsselung mit TPM2 Speicher-Root-Schlüsseln wie Clevis oder systemd-cryptenroll eingerichtet ist, stellen Sie sicher, dass Sie nicht dasselbe persistente Handle verwenden, das von diesen Tools verwendet wird. Die Verwendung desselben persistenten Benutzernamens kann sich auf Ihren Festplattenverschlüsselungsmechanismus auswirken. Führen Sie den folgenden Befehl aus, um alle erstellten und verwendeten persistenten Handles zu überprüfen

```
sudo tpm2_getcap handles-persistent
```

Nächste Schritte

Nachdem Sie Ihr Greengrass-Core-Gerät erfolgreich integriert haben TPM2, können Sie:

- Stellen Sie Komponenten auf Ihrem sicheren Greengrass-Gerät bereit. Weitere Informationen finden Sie unter AWS IoT Greengrass Komponenten auf Geräten bereitstellen.
- Richten Sie zusätzliche Greengrass-Geräte mit TPM2 Integration ein.

Weitere Informationen zur Sicherheit auf Greengrass-Geräten finden Sie unterSicherheit in AWS IoT Greengrass.

Tutorial: Schützen Sie AWS IoT Greengrass Nucleus Lite mit Trusted Platform Module (TPM)

In diesem Tutorial wird erklärt, wie Sie die TPM-Unterstützung (Trusted Platform Module) für AWS IoT Greengrass Nucleus Lite aktivieren und konfigurieren. TPM bietet eine hardwarebasierte Vertrauensbasis für die sichere Speicherung von Schlüsseln. Diese Sicherheitsfunktion schützt kryptografische Operationen und vertrauliche Anmeldeinformationen und verbessert so die Gerätesicherheit und -integrität.

Wenn Sie diese Integration abgeschlossen haben, verwendet Ihr AWS IoT Greengrass Kerngerät TPM-geschützte private Schlüssel für seine Identität und Kommunikation mit Diensten. AWS IoT

Weitere Informationen zur Sicherheit von AWS IoT Greengrass Geräten finden Sie unter. Sicherheit in AWS IoT Greengrass

> **Important**
>
> Dieser Mechanismus wird nur bei der Installation von AWS IoT Greengrass Nucleus Lite mit manueller Ressourcenbereitstellung unterstützt.

Voraussetzungen

Zum Durcharbeiten dieses Tutorials ist Folgendes erforderlich:

- Ein Linux-kompatibles Gerät mit TPM 2.0-Hardware oder NitroTPM
- Ein Entwicklercomputer, auf dem Greengrass Nucleus Lite installiert ist. Weitere Informationen finden Sie unter Installieren der AWS IoT Greengrass Core-Software (Konsole).
- Die Anweisungen in diesem Tutorial sind für Ubuntu 24.04 LTS definiert.
- Jede Linux-Distribution, die den TPM2 Linux-Software-Stack unterstützen kann, kann diesen Mechanismus unterstützen.
- Ein Entwicklercomputer, der AWS CLIinstalliert und konfiguriert ist und über folgende Berechtigungen verfügt:
  - AWS IoT Ressourcen erstellen und verwalten
  - Erstellen und verwalten Sie IAM-Rollen und -Richtlinien
- Root- oder Sudo-Rechte auf dem Gerät.

Dieses Tutorial enthält Anweisungen zur Verwendung des TPM2 Chips als Hardware-Sicherheitsmodul (HSM) zur Erstellung eines privaten Schlüssels und einer CSR, mit der AWS IoT das Ding-Zertifikat erstellt wird.

Schritt 1: Richten Sie eine NitroTPM-Instanz ein

1. Richten Sie eine NitroTPM-Instanz ein. Weitere Informationen finden Sie unter NitroTPM.
2. Starten Sie die letzte Instance mit dem benutzerdefinierten AMI, das im vorherigen Schritt erstellt wurde.

> **Important**
>
> Wenn Sie eine Verbindung mit SSH herstellen, verwenden Sie den `ubuntu` Benutzer anstelle von`root`.

3. Überprüfen Sie, ob das TPM-Gerät vorhanden ist und funktioniert, indem Sie den folgenden Befehl ausführen:

```
ls -la /dev/tpm*
```

Sie sollten `/dev/tpmrm0` Geräte sehen`/dev/tpm0`.

Schritt 2: TPM-Tools installieren und konfigurieren

1. Installieren Sie die erforderlichen Pakete, indem Sie den folgenden Befehl ausführen:

```
sudo apt update
sudo apt install tpm2-openssl tpm2-tools tpm2-abrmd libtss2-tcti-tabrmd0
```

2. Überprüfen Sie die Berechtigungen des TPM-Geräts, indem Sie den folgenden Befehl ausführen:

```
ls -l /dev/tpm0    # Should be owned by tss:root with permissions 0660
ls -l /dev/tpmrm0  # Should be owned by tss:tss with permissions 0660
```

Schritt 3: Den OpenSSL-Anbieter TPM2 konfigurieren

1. Bearbeiten Sie die OpenSSL-Konfigurationsdatei:

```
sudo vi /etc/ssl/openssl.cnf
```

2. Fügen Sie die folgende Konfiguration hinzu:

```
[openssl_init]
providers = provider_sect

[provider_sect]
default = default_sect
```

```
tpm2 = tpm2_sect

[default_sect]
activate = 1

[tpm2_sect]
identity = tpm2
module = /usr/local/lib64/tpm2.so
activate = 1
```

3. Passen Sie den Modulpfad nach Bedarf an. Sie können den richtigen Pfad finden, indem Sie:

```
find /usr -name "tpm2.so"
```

Schritt 4: Generieren Sie persistente TPM-Schlüssel

1. Erstellen Sie einen Primärschlüssel, indem Sie den folgenden Befehl ausführen:

```
sudo tpm2_createprimary -C o -c primary.ctx
```

2. Erstellen Sie ein ECC-Schlüsselobjekt, indem Sie den folgenden Befehl ausführen:

```
sudo tpm2_create -C primary.ctx -g sha256 -G ecc256 -r device.priv -u device.pub
```

3. Laden Sie den Schlüssel, indem Sie den folgenden Befehl ausführen:

```
sudo tpm2_load -C primary.ctx -r device.priv -u device.pub -c device.ctx
```

4. Machen Sie den Schlüssel persistent, indem Sie den folgenden Befehl ausführen:

```
sudo tpm2_evictcontrol -C o -c device.ctx 0x81000002
```

   Dadurch wird ein persistenter Schlüssel mit dem Handle (like`0x81000002`) erstellt.

Schritt 5: Generieren Sie eine Zertifikatssignieranforderung (CSR)

In diesem Schritt verwenden Sie den TPM2 -geschützten privaten Schlüssel, um eine Certificate Signing Request (CSR) zu generieren.

1. Generieren Sie eine CSR mithilfe des TPM-Schlüssels:

```
openssl req -new -provider tpm2 -key "handle:0x81000002" \
    -out device.csr \
    -subj "/CN=TPMThing"
```

2. `0x81000002`Ersetzen Sie es durch den von Ihnen gewählten Handle-Wert und `TPMThing` durch den gewünschten Ding-Namen.

Schritt 6: Erstellen Sie das Zertifikat aus CSR

1. Erstellen Sie auf Ihrem Entwicklungscomputer einen Ordner, in den Sie das Zertifikat für das AWS IoT Ding heruntergeladen haben.

```
mkdir greengrass-v2-certs
```

2. 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://path_to_device.csr \
  --certificate-pem-outfile greengrass-v2-certs/device.pem.crt
```

   Wenn die Anfrage erfolgreich ist, sieht die Antwort dem folgenden Beispiel ähnlich:

```
{
"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-----"
}
```

Schritt 7: Greengrass Nucleus Lite mit TPM-Unterstützung einrichten

Um die TPM-Unterstützung in Greengrass Nucleus Lite zu aktivieren, nehmen Sie die folgenden Änderungen vor:

1. Konfigurieren Sie Benutzerberechtigungen, indem Sie den `ggcore` Benutzer der `tss` Gruppe für den TPM-Zugriff hinzufügen:

   ```
   sudo usermod -a -G tss ggcore
   ```

2. Aktualisieren Sie das Verzeichnis mit den Anmeldeinformationen, indem Sie die folgenden Schritte ausführen:

   - Entfernen Sie die Datei mit dem privaten Schlüssel aus dem Verzeichnis mit den Anmeldeinformationen.
     - Da wir persistente TPM-Schlüssel verwenden, muss keine private Schlüsseldatei kopiert werden.
   - Kopieren Sie die neue Datei `device.pem.crt` in das Verzeichnis mit den Anmeldeinformationen.

3. Bearbeiten Sie `config.yaml` mit der folgenden TPM-spezifischen Konfiguration:

   ```
   system:
       privateKeyPath: "handle:0x81000002" # Use your chosen handle
       certificateFilePath: "" # Replace with the path of device.pem.crt
       ...
   ```

4. Starten Sie Ihren Greengrass Nucleus Lite neu, indem Sie den folgenden Befehl ausführen:

```
systemctl restart greengrass-lite.target
```

Fehlerbehebung

Falls bei der Einrichtung oder dem Betrieb Ihres TPM2 -fähigen AWS IoT Greengrass Geräts Probleme auftreten, versuchen Sie es mit den folgenden Schritten zur Problembehebung:

Das TPM-Gerät wurde nicht gefunden (bei Verwendung von NitroTPM)

Wenn `/dev/tpm0` es nicht vorhanden ist, führen Sie die folgenden Schritte aus:

1. Stellen Sie sicher, dass Sie einen von NitroTPM unterstützten Instance-Typ verwenden.
2. Stellen Sie sicher, dass das AMI mit erstellt wurde`--tpm-support v2.0`.
3. Stellen Sie sicher, dass die Instance über das benutzerdefinierte AMI gestartet wurde.

Fehler bei abgelehnter Berechtigung

Wenn Sie auf TPM-Zugriffsfehler stoßen, gehen Sie wie folgt vor:

1. Stellen Sie sicher, dass der Benutzer in der `tss` Gruppe ist:`groups $USER`.
2. Überprüfen Sie die Berechtigungen des TPM-Geräts mit dem folgenden Befehl:

   ```
   ls -l /dev/tpm*
   ```
3. Stellen Sie sicher, dass Sie sich nach dem Hinzufügen zur `tss` Gruppe ab- und wieder angemeldet haben.

Probleme mit OpenSSL-Anbietern

Wenn der TPM-Anbieter nicht gefunden wird, gehen Sie wie folgt vor:

1. Überprüfen Sie `tpm2.so` den Pfad in der`/etc/ssl/openssl.cnf`.
2. Überprüfen Sie die Installation des Anbieters, indem Sie den folgenden Befehl ausführen:

   ```
   openssl list -providers
   ```
3. Stellen Sie sicher, dass das `tpm2-openssl` Paket ordnungsgemäß installiert ist.

Nächste Schritte

Nachdem Sie Ihr AWS IoT Greengrass Kerngerät erfolgreich integriert haben TPM2, können Sie:

- Komponenten auf Ihrem sicheren AWS IoT Greengrass Gerät bereitstellen
- Richten Sie zusätzliche AWS IoT Greengrass Geräte mit TPM2 Integration ein

Weitere Informationen zur Sicherheit von AWS IoT Greengrass Geräten finden Sie unterSicherheit in AWS IoT Greengrass.

Tutorial: Erste Schritte mit SageMaker AI Edge Manager

> **Important**
>
> SageMaker AI Edge Manager wurde am 26. April 2024 eingestellt. Weitere Informationen zur weiteren Bereitstellung Ihrer Modelle auf Edge-Geräten finden Sie unter Ende der Nutzungsdauer von SageMaker AI Edge Manager.

Amazon SageMaker AI Edge Manager ist ein Softwareagent, der auf Edge-Geräten ausgeführt wird. SageMaker AI Edge Manager bietet Modellmanagement für Edge-Geräte, sodass Sie mit Amazon SageMaker AI Neo kompilierte Modelle direkt auf Greengrass-Core-Geräten verpacken und verwenden können. Mithilfe von SageMaker AI 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 AI Edge Manager auf Greengrass-Kerngeräten finden Sie unterVerwenden Sie Amazon SageMaker AI Edge Manager auf Greengrass-Kerngeräten.

Dieses Tutorial zeigt Ihnen, wie Sie mit der Verwendung von SageMaker AI Edge Manager beginnen können, und zwar AWS anhand der bereitgestellten Beispielkomponenten auf einem vorhandenen Kerngerät. Diese Beispielkomponenten verwenden die SageMaker AI Edge Manager-Komponente als Abhängigkeit, um den Edge Manager-Agenten bereitzustellen und Inferenzen mithilfe vortrainierter Modelle durchzuführen, die mit AI Neo kompiliert wurden. SageMaker Weitere Informationen zum SageMaker AI Edge Manager-Agenten finden Sie unter SageMaker AI Edge Manager im Amazon SageMaker AI Developer Guide.

AWS Enthält Beispielcode, mit dem Sie die folgenden Inferenz- und Modellkomponenten erstellen können, um den SageMaker AI Edge Manager-Agenten auf einem vorhandenen Greengrass-Core-Gerät einzurichten und zu verwenden.

- Klassifizierung von Bildern
  - `com.greengrass.SageMakerEdgeManager.ImageClassification`
  - `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`
- Objekterkennung
  - `com.greengrass.SageMakerEdgeManager.ObjectDetection`
  - `com.greengrass.SageMakerEdgeManager.ObjectDetection.Model`

Dieses Tutorial zeigt Ihnen, wie Sie die Beispielkomponenten und den SageMaker AI Edge Manager-Agenten bereitstellen.

Themen

- Voraussetzungen
- Richten Sie Ihr Greengrass-Core-Gerät in SageMaker AI Edge Manager ein
- Erstellen Sie die Beispielkomponenten
- Führen Sie ein Beispiel für die Inferenz zur Bildklassifizierung aus

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 amzn-s3-demo-bucket1 verwendet, um auf diesen Bucket zu verweisen.
  - Ein S3-Bucket, den Sie mit Ihrer KI-Edge-Geräteflotte verknüpfen. SageMaker SageMaker AI 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 amzn-s3-demo-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.

  JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
  {
    "Action": [
      "s3:GetObject"
    ],
    "Resource": [
      "arn:aws:s3:::amzn-s3-demo-bucket1/*"
    ],
    "Effect": "Allow"
  }
  ]
}
```

Richten Sie Ihr Greengrass-Core-Gerät in SageMaker AI Edge Manager ein

Edge-Geräteflotten in SageMaker AI Edge Manager sind Sammlungen logisch gruppierter Geräte. Um SageMaker AI 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 AI Edge Manager-Agent bereitstellen. Anschließend müssen Sie das Kerngerät 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 Amazon SageMaker AI-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 Amazon SageMaker AI-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 AI Edge Manager-Komponente zu erleichtern, AWS bietet es ein Python-Skript 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 amzn-s3-demo-bucket
   ```

   *region*Ersetzen Sie durch den AWS-Region Ort, an dem Sie Ihr Greengrass-Core-Gerät erstellt haben, und ersetzen Sie amzn-s3-demo-bucket1 durch den Namen des S3-Buckets, in dem Sie Ihre Komponentenartefakte speichern möchten.

   > **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 AI 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 AI Edge Manager-Agenten auszuführen, müssen Sie diese Komponenten auf Ihrem Kerngerät bereitstellen. Durch die Bereitstellung dieser Komponenten wird ein von SageMaker AI Neo kompiliertes, vortrainiertes Resnet-50-Modell heruntergeladen und der SageMaker AI Edge Manager-Agent auf Ihrem Gerät installiert. Der SageMaker AI 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 Bereitstellungsseite 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": "amzn-s3-demo-bucket"
```

```
}
```

*device-fleet-name*Ersetzen Sie es durch den Namen der Edge-Geräteflotte, die Sie erstellt haben, und *amzn-s3-demo-bucket* ersetzen Sie es 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 aus

Zur Bereitstellung Ihrer Komponenten (AWS CLI)

1. Erstellen Sie auf Ihrem Entwicklungscomputer eine `deployment.json` Datei, um die Bereitstellungskonfiguration für Ihre SageMaker AI 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\":\"amzn-
s3-demo-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 es dann *`amzn-s3-demo-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 passieren, 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 Inferenz für maschinelles Lernen.

Tutorial: Führen Sie mit Lite eine Inferenz zur Klassifizierung von Beispielbildern durch TensorFlow

Dieses Tutorial zeigt Ihnen, wie Sie die TensorFlow Lite-Komponente für die Bildklassifizierung verwenden, um eine Beispielinferenz für die Bildklassifizierung auf einem Greengrass-Core-Gerät durchzuführen. Diese Komponente umfasst die folgenden Komponentenabhängigkeiten:

- TensorFlow Lite-Komponente zum Speichern von Bildklassifizierungsmodellen
- TensorFlow Lite-Runtime-Komponente

Wenn Sie diese Komponente bereitstellen, lädt sie ein vortrainiertes MobileNet v1-Modell herunter und installiert die TensorFlow Lite-Runtime und ihre Abhängigkeiten. Diese Komponente veröffentlicht Inferenzergebnisse zu diesem Thema. `ml/tflite/image-classification` Um diese Inferenzergebnisse anzusehen, verwenden Sie den AWS IoT MQTT-Client in der AWS IoT Konsole, um dieses Thema zu abonnieren.

In diesem Tutorial setzen Sie die Beispiel-Inferenzkomponente ein, um die Bildklassifizierung für das Beispielbild durchzuführen, das von bereitgestellt wird. AWS IoT Greengrass Nachdem Sie dieses Tutorial abgeschlossen haben, können Sie es abschließen. Es zeigt IhnenTutorial: Führen Sie mit Lite eine Beispielinferenz zur Bildklassifizierung von Bildern aus einer Kamera durch TensorFlow ,

wie Sie die Beispiel-Inferenzkomponente modifizieren, um Bilder von einer Kamera lokal auf einem Greengrass-Core-Gerät zu klassifizieren.

Weitere Informationen zum maschinellen Lernen auf Greengrass-Geräten finden Sie unterDurchführen von Machine Learning-Inferenzen.

Themen

- Voraussetzungen
- Schritt 1: Abonnieren Sie das Thema Standardbenachrichtigungen
- Schritt 2: Stellen Sie die TensorFlow Lite-Komponente zur Bildklassifizierung bereit
- Schritt 3: Inferenzergebnisse anzeigen
- Nächste Schritte

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 Kerngerät muss die folgenden Anforderungen erfüllen:
  - 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.

Schritt 1: Abonnieren Sie das Thema Standardbenachrichtigungen

In diesem Schritt konfigurieren Sie den AWS IoT MQTT-Client in der AWS IoT Konsole so, dass er MQTT-Nachrichten überwacht, die von der TensorFlow Lite-Komponente zur Bildklassifizierung veröffentlicht wurden. Standardmäßig veröffentlicht die Komponente Inferenzergebnisse zu diesem Thema. `ml/tflite/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.
2. Geben **`ml/tflite/image-classification`**Sie unter Thema abonnieren in das Feld Themenname den Text ein.
3. Wählen Sie Subscribe (Abonnieren) aus.

Schritt 2: Stellen Sie die TensorFlow Lite-Komponente zur Bildklassifizierung bereit

In diesem Schritt stellen Sie die TensorFlow Lite-Komponente zur Imageklassifizierung auf Ihrem Kerngerät bereit:

So stellen Sie die TensorFlow Lite-Komponente zur Bildklassifizierung bereit (Konsole)

1. Wählen Sie im Navigationsmenü der AWS IoT Greengrass Konsole die Option Komponenten aus.
2. Wählen Sie auf der Seite Komponenten auf der Registerkarte Öffentliche Komponenten die Option `aws.greengrass.TensorFlowLiteImageClassification` aus.
3. Auf dem aws.greengrass.TensorFlowLiteImageClassificationWählen Sie auf der Seite Deploy 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. Vergewissern Sie sich auf der Seite Komponenten auswählen unter Öffentliche Komponenten, dass die `aws.greengrass.TensorFlowLiteImageClassification` Komponente ausgewählt ist, und klicken Sie auf Weiter.
7. Behalten Sie auf der Seite Komponenten konfigurieren die Standardkonfigurationseinstellungen bei und wählen Sie 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

So stellen Sie die TensorFlow Lite-Komponente zur Bildklassifizierung 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`*
   - Dieses Tutorial verwendet die Komponentenversion 2.1.0. 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-Komponente zur Bildklassifizierung 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: Inferenzergebnisse anzeigen

Nachdem Sie die Komponente 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. Schritt 1: Abonnieren Sie das Thema Standardbenachrichtigungen

- 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**`ml/tflite/image-classification`**.

     Sie sollten Meldungen sehen, die dem folgenden Beispiel ähneln.

```
{
  "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-Core-Gerät aus.

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

  Sie sollten Ergebnisse sehen, die dem folgenden Beispiel ähneln.

  ```
  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 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 Inferenz für maschinelles Lernen.

Nächste Schritte

Wenn Sie über ein Greengrass-Core-Gerät mit einer unterstützten Kameraschnittstelle verfügen, können Sie den Vorgang abschließen. Hier erfahren SieTutorial: Führen Sie mit Lite eine Beispielinferenz zur Bildklassifizierung von Bildern aus einer Kamera durch TensorFlow , wie Sie die Beispiel-Inferenzkomponente modifizieren, um die Bildklassifizierung von Bildern von einer Kamera durchzuführen.

Um die Konfiguration der TensorFlow Lite-Beispielkomponente für die Inferenz zur Bildklassifizierung genauer zu untersuchen, versuchen Sie Folgendes:

- Ändern Sie den `InferenceInterval` Konfigurationsparameter, um zu ändern, wie oft der Inferenzcode ausgeführt wird.
- Ändern Sie die `ImageName` und die `ImageDirectory` Konfigurationsparameter in der Konfiguration der Inferenzkomponente, um ein benutzerdefiniertes Bild anzugeben, das für die Inferenz verwendet werden soll.

Informationen zum Anpassen der Konfiguration öffentlicher Komponenten oder zum Erstellen benutzerdefinierter Komponenten für maschinelles Lernen finden Sie unter. Passen Sie Ihre Komponenten für maschinelles Lernen an

Tutorial: Führen Sie mit Lite eine Beispielinferenz zur Bildklassifizierung von Bildern aus einer Kamera durch TensorFlow

In diesem Tutorial erfahren Sie, wie Sie mit der TensorFlow Lite-Komponente zur Bildklassifizierung eine Beispielinferenz zur Bildklassifizierung für Bilder von einer Kamera lokal auf einem Greengrass-Core-Gerät durchführen können. Diese Komponente umfasst die folgenden Komponentenabhängigkeiten:

- TensorFlow Lite-Komponente zum Speichern von Bildklassifizierungsmodellen
- TensorFlow Lite-Runtime-Komponente

> **Note**
>
> Dieses Tutorial greift auf das Kameramodul für Raspberry Pi - oder NVIDIA Jetson Nano-Geräte zu, AWS IoT Greengrass unterstützt jedoch auch 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 maschinellen Lernen auf Greengrass-Geräten finden Sie unterDurchführen von Machine Learning-Inferenzen.

Themen

- Voraussetzungen
- Schritt 1: Konfigurieren Sie das Kameramodul auf Ihrem Gerät
- Schritt 2: Bestätigen Sie Ihr Abonnement für das Thema Standardbenachrichtigungen
- Schritt 3: Ändern Sie die Konfiguration der TensorFlow Lite-Image-Klassifizierungskomponente und stellen Sie sie bereit
- Schritt 4: Inferenzergebnisse anzeigen
- Nächste Schritte

Voraussetzungen

Um dieses Tutorial abzuschließen, müssen Sie es zuerst abschließenTutorial: Führen Sie mit Lite eine Inferenz zur Klassifizierung von Beispielbildern durch TensorFlow .

Sie benötigen außerdem Folgendes:

- Ein Linux-Greengrass-Core-Gerät mit einer Kameraschnittstelle. In diesem Tutorial wird auf das Kameramodul auf einem der folgenden unterstützten Geräte zugegriffen:
  - Raspberry Pi mit Raspberry Pi OS (früher Raspbian genannt)
  - NVIDIA Jetson Nano

  Informationen zur Einrichtung eines Greengrass-Core-Geräts finden Sie unterTutorial: 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 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.
- Für Raspberry Pi- oder NVIDIA Jetson Nano-Geräte, Raspberry Pi-Kameramodul V2 — 8 Megapixel, 1080p. Weitere Informationen zum Einrichten der Kamera finden Sie unter Verbinden der Kamera in der Raspberry Pi-Dokumentation.

Schritt 1: Konfigurieren Sie das Kameramodul 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. Stellen Sie sicher, dass 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`python3.7`, verwenden Sie `python3.7` stattdessen die Befehle in diesem Tutorial. `python3` 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 das Kamera-Setup 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, der es der Inferenzkomponente ermöglicht, von der virtuellen Umgebung aus, die von der Runtime-Komponente erstellt wurde, auf Ihre Kamera zuzugreifen.

   ```
   sudo ln -s /usr/lib/python3/dist-packages/picamera "MLRootPath/
   greengrass_ml_tflite_venv/lib/python3.7/site-packages"
   ```

   Der Standardwert *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 Tutorial: Führen Sie mit Lite eine Inferenz zur Klassifizierung von Beispielbildern durch TensorFlow bereitstellen.

Jetson Nano (Armv8)

1. Führen Sie den folgenden Befehl aus, um das Kamera-Setup 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 Bild mit dem Namen test.jpg Ihres aktuellen Verzeichnisses erfasst und gespeichert.

2. (Optional) Starten Sie das Gerät neu. Wenn bei der Ausführung des gst-launch Befehls im vorherigen Schritt Probleme auftreten, können diese Probleme möglicherweise durch einen Neustart des Geräts behoben werden.

   ```
   sudo reboot
   ```

> **Note**
>
> Bei Armv8 (AArch64) -Geräten, wie z. B. einem Jetson Nano, müssen Sie keinen Symlink erstellen, damit die Inferenzkomponente von der virtuellen Umgebung aus, die von der Runtime-Komponente erstellt wurde, auf die Kamera zugreifen kann.

Schritt 2: Bestätigen Sie Ihr Abonnement für das Thema Standardbenachrichtigungen

In haben Sie konfiguriertTutorial: Führen Sie mit Lite eine Inferenz zur Klassifizierung von Beispielbildern durch TensorFlow , dass der AWS IoT MQTT-Client in der AWS IoT Konsole so konfiguriert ist, dass er MQTT-Nachrichten überwacht, die von der TensorFlow Lite-Komponente zur Bildklassifizierung zu diesem Thema veröffentlicht wurden. `ml/tflite/image-classification` Stellen Sie in der AWS IoT Konsole sicher, dass dieses Abonnement existiert. Falls nicht, folgen Sie den Schritten unter, Schritt 1: Abonnieren Sie das Thema Standardbenachrichtigungen um dieses Thema zu abonnieren, bevor Sie die Komponente auf Ihrem Greengrass-Core-Gerät bereitstellen.

Schritt 3: Ändern Sie die Konfiguration der TensorFlow Lite-Image-Klassifizierungskomponente und stellen Sie sie bereit

In diesem Schritt konfigurieren Sie die TensorFlow Lite-Komponente zur Imageklassifizierung und stellen sie auf Ihrem Kerngerät bereit:

So konfigurieren und implementieren Sie die TensorFlow Lite-Komponente zur Bildklassifizierung (Konsole)

1. Wählen Sie im Navigationsmenü der AWS IoT Greengrass Konsole die Option Komponenten aus.
2. Wählen Sie auf der Seite Komponenten auf der Registerkarte Öffentliche Komponenten die Option `aws.greengrass.TensorFlowLiteImageClassification` aus.
3. Auf dem aws.greengrass.TensorFlowLiteImageClassificationWählen Sie auf der Seite Deploy 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. Vergewissern Sie sich auf der Seite Komponenten auswählen unter Öffentliche Komponenten, dass die `aws.greengrass.TensorFlowLiteImageClassification` Komponente ausgewählt ist, und klicken Sie auf Weiter.
7. Gehen Sie auf der Seite Komponenten konfigurieren wie folgt vor:

   a. Wählen Sie die Inferenzkomponente und anschließend Komponente konfigurieren aus.

   b. Geben Sie unter Konfigurationsupdate das folgende Konfigurationsupdate in das Feld Zusammenzufügende Konfiguration ein.

   ```
   {
     "InferenceInterval": "60",
     "UseCamera": "true"
   }
   ```

   Mit diesem Konfigurationsupdate greift die Komponente auf das Kameramodul auf Ihrem Gerät zu und führt Rückschlüsse auf die von der Kamera aufgenommenen Bilder durch. 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

So konfigurieren und implementieren Sie die TensorFlow Lite-Komponente zur Bildklassifizierung (AWS CLI)

1. Erstellen Sie eine `deployment.json` Datei, um die Bereitstellungskonfiguration für die TensorFlow Lite-Imageklassifizierungskomponente 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`*
- Dieses Tutorial verwendet die Komponentenversion 2.1.0. 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 Rückschlüsse auf die von der Kamera aufgenommenen Bilder durch. Der Inferenzcode wird alle 60 Sekunden ausgeführt. Ersetzen Sie die folgenden Werte

2. Führen Sie den folgenden Befehl aus, um die TensorFlow Lite-Komponente zur Bildklassifizierung 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: Inferenzergebnisse anzeigen

Nachdem Sie die Komponente 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. Schritt 2: Bestätigen Sie Ihr Abonnement für das Thema Standardbenachrichtigungen

- 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**`ml/tflite/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/aws.greengrass.TensorFlowLiteImageClassification.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 erforderlichen 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 Inferenz für maschinelles Lernen.

Nächste Schritte

In diesem Tutorial erfahren Sie, wie Sie die TensorFlow Lite-Komponente zur Bildklassifizierung mit benutzerdefinierten Konfigurationsoptionen verwenden, um anhand von Bildern, die mit einer Kamera aufgenommen wurden, anhand von Beispielen eine Bildklassifizierung durchzuführen.

Weitere Informationen zum Anpassen der Konfiguration öffentlicher Komponenten oder zum Erstellen benutzerdefinierter Komponenten für maschinelles Lernen finden Sie unterPassen Sie Ihre Komponenten für maschinelles Lernen an.

Komponenten

AWS IoT Greengrass Komponenten sind Softwaremodule, die Sie auf Greengrass-Kerngeräten bereitstellen. Komponenten können Anwendungen, Runtime-Installationsprogramme, Bibliotheken oder beliebigen Code darstellen, den Sie auf einem Gerät ausführen würden. Sie können Komponenten definieren, die von anderen Komponenten abhängen. Sie könnten 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 einsetzen, stellt Greengrass nur die Softwaremodule bereit, die Ihre Geräte benötigen.

Themen

- AWS-mitgelieferte Komponenten
- Von Publishern unterstützte Komponenten
- Komponenten der Gemeinschaft
- AWS IoT Greengrass Entwicklungstools
- AWS IoT Greengrass Komponenten entwickeln
- AWS IoT Greengrass Komponenten auf Geräten bereitstellen

AWS-mitgelieferte Komponenten

AWS IoT Greengrass stellt vorgefertigte Komponenten bereit und verwaltet sie, 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 für ihre eigenständige Funktionalität auf Ihren Geräten bereitstellen oder sie als Abhängigkeiten in Ihren benutzerdefinierten Greengrass-Komponenten verwenden.

> **Note**
>
> Einige der AWS bereitgestellten Komponenten hängen von bestimmten Nebenversionen des Greengrass-Kerns ab. Aufgrund dieser Abhängigkeit müssen Sie diese Komponenten aktualisieren, wenn Sie den Greengrass Nucleus auf eine neue Nebenversion aktualisieren. Informationen zu den spezifischen Versionen von Nucleus, von denen jede Komponente abhängt, finden Sie im entsprechenden Komponententhema. Weitere Informationen zur

Aktualisierung des Nucleus finden Sie unterAktualisieren Sie die AWS IoT Greengrass Core-Software (OTA).

Wenn eine Komponente einen Komponententyp sowohl generisch als auch Lambda hat, ist die aktuelle Version der Komponente der generische Typ und eine frühere Version der Komponente ist der Lambda-Typ.

Komponente	Description	Komponentyp	Unterstützes Betriebssystem	Open-Source	Nucleus Lite-kompatibel
Grüngraskern	Der AWS IoT Greengrass Kern der Core-Software. Verwenden Sie diese Komponente, um die Software auf Ihren Kerngeräten zu konfigurieren und zu aktualisieren.	Nucleus	Linux, Windows	Ja	Nein
Nucleus lite von Greengrass	Ein leichter Kern für Geräte mit beschränkten Ressourcen,	NucleusLite	Linux	Ja	Nein

Komponente	Description	Komponent entyp	Unterstützes Betriebssystem	Open-Source	Nucleus Lite-kompatibel
	optimiert für kostengünstige Edge-Geräte und großvolumige Anwendungen				
Authentifizierung auf Client-Geräten	Ermöglicht es lokalen IoT-Geräten, so genannten Client-Geräten, eine Verbindung zum Kerngerät herzustellen.	Plug-In	Linux, Windows	Ja	Nein
CloudWatch Metriken	Veröffentlicht benutzerdefinierte Metriken auf Amazon CloudWatch.	Generisch, Lambda	Linux, Windows	Ja	Ja

Komponente	Description	Komponententyp	Unterstützes Betriebssystem	Open-Source	Nucleus Lite-kompatibel
AWS IoT Device Defender	Informiert Administratoren über Änderungen im Status des Greengrass-Core-Geräts, um ungewöhnliches Verhalten zu erkennen.	Generisch, Lambda	Linux, Windows	Ja	Nein

Komponente	Description	Komponententyp	Unterstützes Betriebssystem	Open-Source	Nucleus Lite-kompatibel
Festplattenspooler	Aktiviert eine persistente Speicheroption für Nachrichten, an die von Greengrass-Core-Geräten gespoolt wurden. AWS IoT Core Diese Komponente speichert diese ausgehenden Nachrichten auf der Festplatte.	Plug-In	Linux, Windows	Ja	Nein

Komponente	Description	Komponententyp	Unterstützes Betriebssystem	Open-Source	Nucleus Lite-kompatibel
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	Nein

Komponente	Description	Komponententyp	Unterstützes Betriebssystem	Open-Source	Nucleus Lite-kompatibel
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	Nein

Komponente	Description	Komponententyp	Unterstützes Betriebssystem	Open-Source	Nucleus Lite-kompatibel
Greengrass CLI	Stellt eine Befehlszeilenschnittstelle bereit, mit der Sie lokale Bereitstellungen erstellen und mit dem Greengrass-Kerngerät und seinen Komponenten interagieren können.	Plug-In	Linux, Windows	Ja	Nein

Komponente	Description	Komponententyp	Unterstützes Betriebssystem	Open-Source	Nucleus Lite-kompatibel
IP-Detektor	Meldet Verbindungsinformationen für den MQTT-Broker an AWS IoT Greengrass, sodass Client-Geräte herausfinden können, wie eine Verbindung hergestellt werden kann.	Plug-In	Linux, Windows	Ja	Nein
Firehose	Veröffentlicht Daten über Amazon Data Firehose-Lieferdatenströme an Ziele in der AWS Cloud.	Lambda	Linux	Nein	Nein

Komponente	Description	Komponententyp	Unterstützes Betriebssystem	Open-Source	Nucleus Lite-kompatibel
Lambda-Launcher	Verwaltet Prozesse und Umgebungskonfigurationen für Lambda-Funktionen.	Generisch	Linux	Nein	Nein
Lambda-Manager	Verwaltet die Kommunikation zwischen Prozessen und die Skalierung für Lambda-Funktionen.	Plug-In	Linux	Nein	Nein
Lambda-Laufzeiten	Stellt Artefakte für jede Lambda-Laufzeit bereit.	Generisch	Linux	Nein	Nein

Komponente	Description	Komponententyp	Unterstützes Betriebssystem	Open-Source	Nucleus Lite-kompatibel
Legacy-Abonnement-Router	Verwaltet Abonnements für Lambda-Funktionen, die auf AWS IoT Greengrass V1 ausgeführt werden.	Generisch	Linux	Nein	Nein
Lokale Debug-Konsole	Stellt eine lokale Konsole bereit, mit der Sie das Greengrass-Core-Gerät und seine Komponenten debuggen und verwalten können.	Plug-In	Linux, Windows	Ja	Nein

Komponente	Description	Komponententyp	Unterstützes Betriebssystem	Open-Source	Nucleus Lite-kompatibel
Protokollmanager	Sammelt Protokolle und lädt sie auf das Greengrass-Core-Gerät hoch.	Plug-In	Linux, Windows	Ja	Nein
Komponenten für maschinelles Lernen	Stellt Modelle für maschinelles Lernen und Beispiel-Inferenzcode bereit, mit denen Sie Inferenzen für maschinelles Lernen auf Greengrass-Kerngeräten durchführen können.	Siehe Komponenten für maschinelles Lernen.			Nein

Komponente	Description	Komponententyp	Unterstützes Betriebssystem	Open-Source	Nucleus Lite-kompatibel
Modbus-RTU-Protokolladapter	Ruft Informationen von lokalen Modbus RTU-Geräten ab.	Lambda	Linux	Nein	Nein
Nucleus-Telemetrie-Sender	Publiziert Telemetriedaten zur Systemintegrität, die vom Nucleus gesammelt wurden, zu einem lokalen Thema oder zu einem MQTT-Thema. AWS IoT Core	Plug-In	Linux, Windows	Ja	Nein

Komponente	Description	Komponententyp	Unterstützes Betriebssystem	Open-Source	Nucleus Lite-kompatibel
MQTT-Brücke	Leitet MQTT-Nachrichten zwischen Client-Geräten weiter, veröffentlicht lokal AWS IoT Greengrass , abonniert und. AWS IoT Core	Plug-In	Linux, Windows	Ja	Nein
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	Nein

Komponente	Description	Komponententyp	Unterstützes Betriebssystem	Open-Source	Nucleus Lite-kompatibel
MQTT 5-Broker (EMAX)	Führt einen MQTT 5-Broker aus, der Nachrichten zwischen Client-Geräten und dem Core-Gerät verarbeitet.	Generisch	Linux, Windows	Nein	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	Nein

Komponente	Description	Komponententyp	Unterstützes Betriebssystem	Open-Source	Nucleus Lite-kompatibel
Geheimer Manager	Stellt Secrets from AWS Secrets Manager Secrets bereit, sodass Sie Anmeldeinformationen wie Passwörter sicher in benutzerdefinierten Komponenten auf dem Greengrass-Core-Gerät verwenden können.	Plug-In	Linux, Windows	Ja	Nein

Komponente	Description	Komponententyp	Unterstützes Betriebssystem	Open-Source	Nucleus Lite-kompatibel
Sicheres Tunneling	Ermöglicht AWS IoT sichere Tunnelverbindungen, mit denen Sie bidirektionale Kommunikation mit Greengrass-Core-Geräten herstellen können, die sich hinter eingeschränkten Firewalls befinden.	Generisch	Linux	Nein	Ja

Komponente	Description	Komponententyp	Unterstützes Betriebssystem	Open-Source	Nucleus Lite-kompatibel
Schattenmanager	Ermöglicht die Interaktion mit Schatten auf dem Kerngerät. Es verwaltet die Speicherung von Shadow-Dokumenten und auch die Synchronisation lokaler Shadow-Zustände mit dem AWS IoT Device Shadow-Dienst.	Plug-In	Linux, Windows	Ja	Nein
Amazon SNS	Veröffentlicht Nachrichten zu Amazon SNS SNS-Themen.	Lambda	Linux	Nein	Nein

Komponente	Description	Komponententyp	Unterstützes Betriebssystem	Open-Source	Nucleus Lite-kompatibel
Stream-Manager	Streamt große Datenmengen aus lokalen Quellen in die. AWS Cloud	Generisch	Linux, Windows	Nein	Ja
Systemprotokollweiterleitung	Laden Sie systemd-journald-Protokolle in den hoch. AWS Cloud	Generisch	Linux	Ja	Ja
Systemmanager-Agent	Verwalten Sie das Kerngerät mit AWS Systems Manager, mit dem Sie Geräte patchen, Befehle ausführen und vieles mehr können.	Generisch	Linux	Ja	Nein

Komponente	Description	Komponententyp	Unterstützes Betriebssystem	Open-Source	Nucleus Lite-kompatibel
Token-Austauschdienst	Stellt AWS Anmeldeinformationen bereit, die Sie für die Interaktion mit AWS Diensten verwenden können.	Generisch	Linux, Windows	Nein	Nein
SiteWise IoT-OPC-UA-Kollektor	Sammelt Daten von OPC-UA-Servern.	Generisch	Linux, Windows	Nein	Nein
IoT SiteWise OPC UA-Datenquellensimulator	Führt einen lokalen OPC-UA-Server aus, der Beispieldaten generiert.	Generisch	Linux, Windows	Nein	Nein
SiteWise IoT-Herausgeber	Veröffentlicht Daten in der AWS Cloud.	Generisch	Linux, Windows	Nein	Nein

Komponente	Description	Komponententyp	Unterstützes Betriebssystem	Open-Source	Nucleus Lite-kompatibel
SiteWise IoT-Prozessor	Verarbeitet Daten auf den Greengrass-Kerngeräten.	Generisch	Linux, Windows	Nein	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.

> **Note**
>
> Ab der Greengrass-Version 2.14.0 ist eine für den Speicherbedarf optimierte Version der Nucleus-Gerätelaufzeit für Geräte mit eingeschränkten Randbedingungen verfügbar. Weitere Informationen zu Konfiguration und Verwendung finden Sie unter Greengrass Nucleus Lite.

> **Important**
>
> Wenn sich die Version der Nucleus-Komponente ändert oder wenn Sie bestimmte Konfigurationsparameter ändern, wird die AWS IoT Greengrass Core-Software — zu der auch 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 Sie die AWS IoT Greengrass Core-Software (OTA).

Themen

- Versionen
- Anforderungen an Speichergeräte
- Unterstützte Plattformen
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Herunterladen und Installation
- Konfiguration
- Lokale Protokolldatei
- Änderungsprotokoll

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.16.x
- 2.15.x
- 2.14.x
- 2.13.x
- 2.12.x
- 2.11.x
- 2.10.x

- 2.9.x
- 2.8.x
- 2.7.x
- 2.6.x
- 2.5.x
- 2.4.x
- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Anforderungen an Speichergeräte

> **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 AWS Cloud kommunizieren kann. 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 Geräteverkehr über einen Proxy oder eine Firewall zulassen.
- 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:

  - (Fakultativ)`systemctl`. Dieser Befehl wird verwendet, um die AWS IoT Greengrass Core-Software als Systemdienst einzurichten.
  - (Optional) `useraddgroupadd`, und`usermod`. Diese Befehle werden verwendet, um den `ggc_user` Systembenutzer und die `ggc_group` Systemgruppe einzurichten.
  - (Fakultativ)`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 Geräteverkehr über einen Proxy oder eine Firewall zulassen.

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 erhalten Sie unter Kompatibilität der Greengrass-Funktionen und Überlegungen zu Funktionen.

Überlegungen zu Funktionen

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 Kompatibilität der Greengrass-Funktionen.

Um ein benutzerdefiniertes Linux-basiertes Betriebssystem zu erstellen, können Sie das BitBake Rezept für AWS IoT Greengrass 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 installiert, konfiguriert und führt die AWS IoT Greengrass Core-Software automatisch auf Ihrem Gerät aus.

Linux-Plattformen können auch AWS IoT Greengrass in einem Docker-Container ausgeführt werden. Weitere Informationen finden Sie unter Führen Sie die AWS IoT Greengrass Core-Software in einem Docker-Container aus.

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 AWS-mitgelieferte 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 unterKonfiguration der 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 eingestellt `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 die von Festplattenspooler Greengrass bereitgestellte Version eingestellt `aws.greengrass.DiskSpooler` und verwendet diese.

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.
- `DUALSTACK`— Die vorsignierte S3-URL verwendet den Dual-Stack-Endpunkt.

Standard: `GLOBAL`

`fipsMode`

(Optional) Veranlasst Greengrass, FIPS-Endpunkte zu verwenden. Weitere Informationen zur Aktivierung von FIPS-Endpunkten finden Sie unter FIPS-Endpunkte.

Wählen Sie aus den folgenden Optionen aus:

- `true`Wenn die Einstellung auf true gesetzt ist, verwenden die Endgeräte den FIPS-Endpunkt.
- `false`Wenn der Wert falsch ist, verwenden die Endpunkte keinen FIPS-Endpunkt.

Standard: `false`

`logging`

(Optional) Die Protokollierungskonfiguration für das Kerngerät. Weitere Informationen zur Konfiguration und Verwendung von Greengrass-Protokollen finden Sie unterAWS IoT Greengrass Protokolle überwachen.

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 angebe`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. Erfassung von Telemetriedaten zur Systemintegrität von AWS IoT Greengrass Kerngeräten

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 Zeitspanne (in Millisekunden), die auf das Öffnen einer Verbindung gewartet werden soll, 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 bei der Verbindung ein Timeout eintritt.

Standard: `30000` (30 Sekunden)

`deploymentConfigurationTimeSource`

Dieser Parameter ist in Version 2.15.0 und höher dieser Komponente verfügbar.

(Optional) Der Zeitstempel, der bei der Verarbeitung einer Bereitstellung verwendet werden soll. Die Standardeinstellung ist der`deploymentCreationTime`.

Dieses Objekt enthält die folgenden Werte:

`deploymentCreationTime`

Der Standardwert von`deploymentConfigurationTimeSource`. Das Gerät verwendet den Zeitstempel für die Erstellung der Bereitstellung, um Konflikte mit den Konfigurationsschlüsseln während der Verarbeitung zu lösen. Wenn dieses Verhalten ausgewählt ist, hat die lokale Gerätekonfiguration, die im Nucleus gespeichert ist, möglicherweise einen längeren Zeitstempel als der der eingehenden Bereitstellung und lehnt eingehende Konfigurationsänderungen ab, die jetzt als veraltet gelten.

`deploymentProcessingTime`

Das Gerät verwendet seinen lokalen Zeitstempel, um Konflikte mit den Konfigurationsschlüsseln während der Bereitstellung zu lösen. Bei der Verarbeitung aktualisiert das Gerät die Konfigurationen anhand des Verarbeitungszeitstempels und nicht anhand des Zeitstempels für die Erstellung der Bereitstellung. Dieses Verhalten setzt voraus, dass die Uhr des Geräts ordnungsgemäß kalibriert ist.

Konfigurieren Sie diese Nucleus-Einstellung in Ihrem ersten Geräte-Image oder Ihrer ersten Installation und nicht im Rahmen einer Bereitstellung, wenn Sie möchten, dass neue Geräte dieses Verhalten bei der ersten Verbindung verwenden. Verwenden Sie für diese Konfiguration die `--init-config`Option des Nucleus Classic-Installationsprogramms.

Diese Erstkonfiguration ist wichtig, da Geräte mehrere Bereitstellungen in beliebiger Reihenfolge verarbeiten. Ohne eine korrekte Erstkonfiguration verarbeitet ein Gerät möglicherweise Bereitstellungen mit dem `deploymentCreationTime` Standardverhalten, bevor es die Bereitstellung empfängt, die die Nucleus-Konfiguration auf festlegt. `deploymentProcessingTime`

Example Beispiel: Aktualisierung der Zusammenführung von Konfigurationen

```
{
  "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
2.16.1	Fehlerkorrekturen und Verbesserungen • Fügt die Konfiguration für Wiederholungsintervalle für Anmeldeinformationen nach Ausfällen des Token-Exchange-Dienstes hinzu.
2.16.0	Neue Features • Nucleus unterstützt jetzt cgroups V2. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem der Nucleus veraltete Bereitstellungen nicht bereinigte, wenn eine Bereitstellung storniert wurde.
2.15.1	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem der Nucleus die Komponentenprozesse nach dem 30-sekündigen Shutdown-Timeout während Bootstrap-Bereitstellungen nicht bereinigen konnte.
2.15.0	Neue Features • Fügt eine Telemetriefunktion hinzu, um Hostsysteminformationen wie CPU- und Betriebssystemdetails einzubeziehen. Weitere Informationen finden Sie auf der Seite zur Telemetriekomponente. • Fügt die `deploymentConfigurationTimeSource` Konfiguration hinzu. Weitere Informationen finden Sie unter Konfiguration der Nucleus-Komponenten. Fehlerkorrekturen und Verbesserungen • Nucleus priorisiert jetzt standardmäßig die Verwendung lokaler Versionen und Artefakte, wodurch die Konsistenz der Bereitstellung verbessert und externe Abhängigkeiten reduziert werden. Benutzer

Version	Änderungen
	können dieses Verhalten außer Kraft setzen, indem sie im Bereitste llungsdokument alternative Einstellungen angeben. • Fügt ein neues Warnprotokoll hinzu, wenn ein bereitgestellter Windows-Benutzer nicht dem normalerweise von Windows akzeptierten Zeichensatz entspricht.
2.14.3	Fehlerkorrekturen und Verbesserungen • Ermöglicht es dem Token-Austauschdienst, bei Änderungen der Portkonfi guration neu zu starten. • Behebt ein Problem, bei dem der Flottenstatusdienst keine Meldungen zur Änderung des Komponentenstatus für nicht angeheftete Lambdas sendet, wenn diese nicht ausgelöst wurden. • Behebt ein Problem, bei dem Komponenten nicht ordnungsgemäß heruntergefahren werden, wenn eine neue Komponentenversion bereitges tellt wird. • Behebt ein Problem, bei dem die Injection-Hooks integrierter Plugins zweimal ausgeführt werden, was zu zusätzlichen Lebenszyklusereignissen und doppelter Protokollierung führt. • Verbessert die Protokollierung des Komponentenlebenszyklus auf Windows-Geräten.
2.14.2	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem ein HTTP-Client nicht mit gegenseitiger Authentifizierung konfiguriert ist.
2.14.1	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem Komponenten bei Neuinstallationen von Greengrass nicht richtig gestoppt wurden.

Version	Änderungen
2.14.0	**Warning** Diese Version ist nicht mehr verfügbar. Die Verbesserungen in dieser Version sind in späteren Versionen dieser Komponente verfügbar. Neue Features • Die neue Unterstützung von Dual-Stack-Endpunkten ermöglicht die IPv6 Netzwerkkommunikation. • Verbesserte Widerstandsfähigkeit gegen Nucleus-Neustartfehler und `Launch` Verzeichnisbeschädigung. Fehlerkorrekturen und Verbesserungen • Behebt Speicherlecks bei Schließungen von PubSub IPC-Abonnements. • Behebt den Ausführungslebenszyklus der Komponente, bei der sie aufgrund eines Start-Timeouts in den Status ERRORED übergeht, wenn die Skipif-Bedingung erfüllt ist. • Behebt ein Problem, bei dem das Core-Gerät keine Verbindung herstelle n kann, AWS IoT Core wenn die TLS-Richtlinie auf _1_3_2022_10 gesetzt ist. TLS13
2.13.0	Neue Features • Support den FIPS-Endpunkt in Nucleus. Fehlerkorrekturen und Verbesserungen • Bereitstellungsverbesserungen abbrechen: Bereitstellungen können jetzt abgebrochen werden, während eine neue Konfiguration zusammeng eführt wird und auf den Start der Dienste gewartet wird.
2.12.6	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, das bei bestimmten ARMv8 Prozessoren, einschlie ßlich des Jetson Nano, beim Start zu einem Absturz führt.

Version	Änderungen
2.12.5	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.
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 Protokoll ierung und MQTT-Veröffentlichung führt.

Version	Änderungen
2.12.0	Neue Features • Ermöglicht es Ihnen, die Bootstrap-Lebenszyklusschritte als Teil einer Rollback-Bereitstellung auszuführen.
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 nicht in Statusaktualisierungen für die Bereitstellung 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.

Version	Änderungen
2.10.3	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem Greengrass keine Bereitstellungsben achrichtigungen abonniert, wenn der PKCS #11 -Anbieter verwendet wird.
2.10.2	Fehlerkorrekturen und Verbesserungen • Ermöglicht die Analyse von Komponenten-Lebenszyklen ohne Berücksic htigung 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, einschlie ßlich des Jetson Nano, zu einem Absturz beim Start 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 hinzu. MQTT5 • Fügt einen Mechanismus hinzu, mit dem Plugin-Komponenten schnell und ohne Scannen geladen werden können. • 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-Clients 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 Fehlercodes für die Bereitstellung. • 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 Statuscodes für Komponenten. • 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 publish/subscribe Themen abonnieren. Weitere Informationen erhalten Sie unter Lokale Nachrichten veröffentlichen/abonnieren und SubscribeToTopic. • Fügt Unterstützung für Rezeptvariablen in Komponentenkonfigurationen hinzu, mit Ausnahme der *component_dependency_name* `:configuration:` *json_pointer* Rezeptvariablen. 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 interpolateComponentConfigurationKonfigurationsoption auf`true`. Weitere Informationen erhalten 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 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 • Behebt ein Problem, bei dem abhängige Komponenten in bestimmten Szenarien nicht reagierten, wenn ihre festen Abhängigkeiten neu gestartet wurden oder ihren Status änderten. • Verbessert die Fehlermeldungen, die das Kerngerät an den AWS IoT Greengrass Cloud-Dienst meldet, wenn eine Bereitstellung fehlschlägt.

Version	Änderungen
	• 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 Greengras s 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:ListThingGroupsF orCoreDevice` 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 AWS IoT Greengrass Komponenten auf Geräten bereitstellen. • Behebt ein Problem mit Systemumgebungsvariablen, die der Greengras s-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. Mit diesem Parameter können Sie den Standardbenutzer angeben, 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 erhalten 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 China 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

Nucleus lite von Greengrass

Der Greengrass Nucleus Lite (`aws.greengrass.NucleusLite`) ist eine Gerätelaufzeit für Geräte mit eingeschränktem Netzwerkrand, die für minimalen Speicherbedarf optimiert ist (verwendet weniger als 5 MB RAM). Es wurde mit AWS IoT Greengrass Version 2.14.0 eingeführt und ist abwärtskompatibel mit AWS IoT Greengrass generischen Komponenten, der Greengrass V2-API und dem SDK.

Der Greengrass Nucleus lite wird als Alternative zum herkömmlichen Greengrass Nucleus (`aws.greengrass.Nucleus`) angeboten und kann in heterogenen Flotten von Greengrass-Geräten eingesetzt werden.

Themen

- Versionen
- Betriebssystem
- Voraussetzungen
- Kompatibilität
- Herunterladen und Installation
- 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 auf Kerngeräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux (Distributionen mit Systemd)

Weitere Informationen finden Sie unter Greengrass Nucleus.

Voraussetzungen

Geräte müssen bestimmte Anforderungen erfüllen, um AWS IoT Greengrass Nucleus Lite und die AWS IoT Greengrass Core-Software installieren und ausführen zu können. Weitere Informationen finden Sie in der Einrichtungsanleitung.

- 5 MB RAM-Speicherplatz für die Nucleus-Laufzeit.
- 5 MB Speicher (Festplatte/Flash).

Zusätzliche Systemabhängigkeiten sind im Setup-Handbuch dokumentiert.

Die Greengrass Nucleus-Komponente wird für die Ausführung in einer VPC unterstützt. Um diese Komponente in einer VPC bereitzustellen, ist Folgendes erforderlich:

- Der Greengrass-Kern muss über Konnektivität zu AWS IoT Daten, AWS IoT Anmeldeinformationen und Amazon S3 verfügen.

Kompatibilität

Der AWS IoT Greengrass Nucleus Lite ist mit der AWS IoT Greengrass v2-API (Teilmenge von) kompatibel und wird unterstützt. SDKs Es hängt nicht von einer bestimmten Sprache ab, runtimes/VMs aber Komponenten, die zu einer Bereitstellung hinzugefügt werden, können das Vorhandensein bestimmter Laufzeiten erfordern (z. B.: Java JVM, Python). Weitere Informationen darüber, welche Funktionen von Greengrass Nucleus Lite unterstützt werden, finden Sie unterKompatibilität der Greengrass-Funktionen.

Herunterladen und Installation

Sie können ein APT-Paket herunterladen, aus dem Quellcode bauen, eine Yocto-Ebene verwenden oder ein vorgefertigtes Yocto-Image für ein kompatibles Gerät herunterladen (z. B.). RaspberryPi Von der AWS IoT Core Konsole aus können Sie ein Verbindungskit herunterladen, das alle Anmeldeinformationen und die Erstkonfiguration für Ihr Gerät enthält. Anweisungen zur Installation sind in jeder spezifischen Verteilungsmethode enthalten.

Sie können auch einem Tutorial folgen, um AWS IoT Greengrass Nucleus Lite 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

Der Nucleus bietet die folgenden Konfigurationsparameter. Einige Parameter setzen voraus, dass die AWS IoT Greengrass Core-Software neu gestartet wird, um wirksam zu werden.

`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 von Kerngeräten für die Interaktion mit AWS Diensten.

`iotDataEndpoint`

Der AWS IoT Datenendpunkt für Ihren AWS-Konto.

`iotCredEndpoint`

Der Endpunkt der AWS IoT Anmeldeinformationen für Ihren AWS-Konto.

`greengrassDataPlanePort`

Der Port, der für Datenebenenverbindungen verwendet werden soll. Weitere Informationen finden Sie unter Connect über Port 443 oder über einen Netzwerk-Proxy herstellen.

> **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 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.

`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 Änderung nach dem nächsten Neustart der AWS IoT Greengrass Core-Software wirksam.

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`http://host:port`.

- `scheme`— Das Schema, das sein muss`http`.
- `host`— Der Hostname oder die IP-Adresse des Proxyservers.
- `port`— (Optional) Die Portnummer. Wenn Sie den Port nicht angeben, verwendet das Greengrass-Core-Gerät den folgenden Standardwert:
  - `http`— 80

Lokale Protokolldatei

Nachrichten werden in der Standardausgabe protokolliert und die Protokolldateien werden von Systemd verarbeitet.

Um die Logs dieser Komponente einzusehen

- Dient `journalctl` zum Anzeigen von Protokollen.

Änderungsprotokoll

Version	Änderungen
2.3.0	Neue Features • Support für die Verwendung von TPM 2.0 für die IoT Core MQTT-Auto risierung. • Die apt-Beispielpakete unterstützen jetzt mehr Betriebssysteme: Ubuntu 22.04, Ubuntu 24.04, Debian 12 und Debian 13. • RestartComponent IPC wird jetzt unterstützt.

Version	Änderungen
	Fehlerkorrekturen und Verbesserungen • Lokale Bereitstellungen benötigen keinen Internetzugang mehr. • GetConfiguration wurde aktualisiert, um dem Laufzeitverhalten von Greengrass Nucleus zu entsprechen. (Bahnbrechende Veränderung) • Allgemeine Fehlerbehebungen und Verbesserungen.
2.2.2	Fehlerkorrekturen und Verbesserungen • Behebt überarbeitete Bereitstellungen mit unveränderten Komponent enversionen, die fehlschlugen, wenn auf einer unveränderten Komponente eine ausführbare Datei als Artefakt ausgeführt wurde. • Behebt die Interpolation von Rezeptvariablen, wenn der interpolierte Wert mehr als vier verschachtelte Unterobjekte enthält. • Behebt die Interpolation von Rezeptvariablen, einschließlich Anführung szeichen und Shell-Sonderzeichen. • Behebt das Leck von FDS, wenn MQTT-Verbindungen fehlschlagen.
2.2.1	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem der Nucleus keine TES-Anmeldeinforma tionen abrufen kann.
2.2.0	Neue Features • Fügt Unterstützung für URIs Container-Image-Artefakte hinzu. Fehlerkorrekturen und Verbesserungen • Allgemeine Fehlerbehebungen und Verbesserungen.

Version	Änderungen
2.1.0	Neue Features • Fügt HTTP-Proxyunterstützung hinzu, die mit der `networkProxy` Konfigurationsoption konfiguriert werden kann. Fehlerkorrekturen und Verbesserungen • Senkt die Anforderung `libcurl` von 7.86 auf 7.82, um Geräten mit älteren Versionen `libcurl` gerecht zu werden. • Aktualisiert die `journalctl` Protokolle, sodass sie Komponenten und nicht Recipe-Runner zugeordnet werden. • Verbessert die Fehlerantworten bei IPC-Aufrufen. • Fügt Wiederholungsversuche für S3-Download-Versuche für generische Komponentenartefakte hinzu. • Kleinere Fehlerbehebungen.
2.0.2	Fehlerkorrekturen und Verbesserungen • Behebt die Abhängigkeiten der einzuschließenden APT-Paket`ecgroup-tools` .
2.0.1	Fehlerkorrekturen und Verbesserungen • Fügt Unterstützung für die Interpolation von Rezeptvariablen für den Timeout-Abschnitt des Greengrass-Rezepts hinzu. • Fügt Unterstützung für den ValidateAuthorizationToken IPC-Befehl für den Stream-Manager hinzu. • Behebt Warnungen beim Fleet Provisioning. • Fügt Wiederholungen und Backoff zum Job-Listener hinzu. • Allgemeine Fehlerbehebungen und Verbesserungen.
2.0.0	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 Sie mit lokalen IoT-Geräten.

Themen

- Versionen
- Typ
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Konfiguration
- Lokale Protokolldatei
- Änderungsprotokoll

Versionen

> **Note**
>
> Version 2.3.0 für die Client-Geräteauthentifizierung wurde eingestellt. Es wird dringend empfohlen, auf Version 2.3.1 oder höher für die Client-Geräteauthentifizierung zu aktualisieren.

Diese Komponente hat die folgenden Versionen:

- 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 Kerngerät ändern.

Diese Komponente verwendet dieselbe Protokolldatei wie der Greengrass-Kern. Weitere Informationen finden Sie unter AWS IoT Greengrass Protokolle überwachen.

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 folgenden MQTT-Themenfilter enthalten: ARNs
    - `$aws/things/coreDeviceThingName*-gci/shadow/update/delta`
    - `$aws/things/coreDeviceThingName*-gci/shadow/get/accepted`
  - `iot:Receive`, wobei die Ressourcen die folgenden MQTT-Themen beinhalten: ARNs
    - `$aws/things/coreDeviceThingName*-gci/shadow/update/delta`
    - `$aws/things/coreDeviceThingName*-gci/shadow/get/accepted`

  Weitere Informationen erhalten Sie unter AWS IoT Richtlinien für den Betrieb 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 Geräteverkehr über einen Proxy oder eine Firewall zulassen.

Endpoint	Port	Erforderlich	Description
`iot.region.amazonaws.com`	443	Ja	Wird verwendet, um Informationen über AWS IoT Ding-Zert

Endpoint	Port	Erforderlich	Description
			ifikate 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.

2.5.5

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.5.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2,6,0 <2,17,0	Weich

2.5.4

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.5.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.6.0 <2.16.0	Weich

2.5.2 – 2.5.3

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.5.2 und 2.5.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.6.0 <2.15.0	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	>=2.6.0 <2.14.0	Weich

2.4.4 - 2.5.0

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.
> Für die Komponente MQTT 3.1.1 Broker (Moquette) wird sie standardmäßig wöchentlich neu gestartet, wenn sich das Serverzertifikat ändert und die Clients gezwungen sind, sich erneut zu autorisieren. 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 – 2.5.4

`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 (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. *deviceClientId*Ersetzen Sie es durch die zu verwendende Client-ID.
- `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 es *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 keine Platzhalter für MQTT-Themen.

`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 ist 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, die die Komponente zum Starten benötigt. Der Status der Komponente ändert sich auf, `ERRORED` wenn dieser Timeout überschritten wird.

Standard: `120`

Example Beispiel: Aktualisierung der Zusammenführung von Konfigurationen (unter Verwendung einer restriktiven Richtlinie)

In der folgenden Beispielkonfiguration wird angegeben, dass Client-Geräte, deren Namen mit beginnen, eine Verbindung herstellen dürfen`MyClientDevice`, und publish/subscribe zwar zu allen Themen.

```
{
```

```
  "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)

Die folgende Beispielkonfiguration gibt an, dass alle Client-Geräte eine Verbindung herstellen können, und zwar zu allen publish/subscribe Themen.

```
{
  "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 enden`MyClientDevice`.

```
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. *`deviceClientId`*Ersetzen Sie es durch die zu verwendende Client-ID.
- `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 es *`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 ist 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, die die Komponente zum Starten benötigt. Der Status der Komponente ändert sich auf, `ERRORED` wenn dieser Timeout überschritten wird.

Standard: `120`

Example Beispiel: Aktualisierung der Zusammenführung von Konfigurationen (unter Verwendung einer restriktiven Richtlinie)

In der folgenden Beispielkonfiguration wird angegeben, dass Client-Geräte, deren Namen mit beginnen, eine Verbindung herstellen dürfen`MyClientDevice`, und publish/subscribe zwar zu allen Themen.

```
{
  "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)

Die folgende Beispielkonfiguration gibt an, dass alle Client-Geräte eine Verbindung herstellen können, und zwar zu allen publish/subscribe Themen.

```
{
  "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. *deviceClientId*Ersetzen Sie es durch die zu verwendende Client-ID.
- `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 es *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 ist 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, die die Komponente zum Starten benötigt. Der Status der Komponente ändert sich auf, `ERRORED` wenn dieser Timeout überschritten wird.

Standard: `120`

Example Beispiel: Aktualisierung der Zusammenführung von Konfigurationen (unter Verwendung einer restriktiven Richtlinie)

In der folgenden Beispielkonfiguration wird angegeben, dass Client-Geräte, deren Namen mit beginnen, eine Verbindung herstellen dürfen`MyClientDevice`, und publish/subscribe zwar zu allen Themen.

```
{
  "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)

Die folgende Beispielkonfiguration gibt an, dass alle Client-Geräte eine Verbindung herstellen können, und zwar zu allen publish/subscribe Themen.

```
{
  "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. *`deviceClientId`*Ersetzen Sie es durch die zu verwendende Client-ID.

- `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 es *`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 ist 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 angegeben, dass Client-Geräte, deren Namen mit beginnen, eine Verbindung herstellen dürfen`MyClientDevice`, und publish/subscribe zwar zu allen Themen.

```
{
  "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)

Die folgende Beispielkonfiguration gibt an, dass alle Client-Geräte eine Verbindung herstellen können, und zwar zu allen publish/subscribe Themen.

```
{
  "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. *`deviceClientId`*Ersetzen Sie es durch die zu verwendende Client-ID.
- `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 es *`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.

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 ist 1.

Example Beispiel: Update zur Zusammenführung von Konfigurationen (unter Verwendung einer restriktiven Richtlinie)

In der folgenden Beispielkonfiguration wird angegeben, dass Client-Geräte, deren Namen mit beginnen, eine Verbindung herstellen dürfen`MyClientDevice`, und publish/subscribe zwar zu allen Themen.

```
{
  "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)

Die folgende Beispielkonfiguration gibt an, dass alle Client-Geräte eine Verbindung herstellen können, und zwar zu allen publish/subscribe Themen.

```
{
  "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. *deviceClientId*Ersetzen Sie es durch die zu verwendende Client-ID.

- `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 es *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

Example Beispiel: Update zur Zusammenführung von Konfigurationen (unter Verwendung einer restriktiven Richtlinie)

In der folgenden Beispielkonfiguration wird angegeben, dass Client-Geräte, deren Namen mit beginnen, eine Verbindung herstellen dürfen`MyClientDevice`, und publish/subscribe zwar zu allen Themen.

```
{
  "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)

Die folgende Beispielkonfiguration gibt an, dass alle Client-Geräte eine Verbindung herstellen können, und zwar zu allen publish/subscribe Themen.

```
{
  "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. *deviceClientId*Ersetzen Sie es durch die zu verwendende Client-ID.
- `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 es *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)

Example Beispiel: Update zur Zusammenführung von Konfigurationen (unter Verwendung einer restriktiven Richtlinie)

In der folgenden Beispielkonfiguration wird angegeben, dass Client-Geräte, deren Namen mit beginnen, eine Verbindung herstellen dürfen`MyClientDevice`, und publish/subscribe zwar zu allen Themen.

```
{
  "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)

Die folgende Beispielkonfiguration gibt an, dass alle Client-Geräte eine Verbindung herstellen können, und zwar zu allen publish/subscribe Themen.

```
{
  "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. *deviceClientId*Ersetzen Sie es durch die zu verwendende Client-ID.
- `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 es *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 angegeben, dass Client-Geräte, deren Namen mit beginnen, eine Verbindung herstellen dürfen`MyClientDevice`, und publish/subscribe zwar zu allen Themen.

```
{
  "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)

Die folgende Beispielkonfiguration gibt an, dass alle Client-Geräte eine Verbindung herstellen können, und zwar zu allen publish/subscribe Themen.

```
{
  "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
2.5.5	Die Version wurde für die Version 2.16.0 von Greengrass Nucleus aktualisiert.
2.5.4	Die Version wurde für die Version 2.15.0 von Greengrass Nucleus aktualisiert.
2.5.3	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem Client-Geräte aufgrund veralteter Client-Zertifikate keine Verbindung zum Kerngerät herstellen können.

Version	Änderungen
2.5.2	Die Version wurde für die Version 2.14.0 von Greengrass Nucleus aktualisiert.
2.5.1	Fehlerkorrekturen und Verbesserungen • Unterstützt den FIPS-Endpunkt.
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.
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.

Version	Änderungen
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.
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-Ge räten, 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 Zertifizi erungsstellen, 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.

Version	Änderungen
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.
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.

Version	Änderungen
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 CloudWatch Amazon-Metrikenkomponente (`aws.greengrass.Cloudwatch`) veröffentlicht benutzerdefinierte Metriken von Greengrass-Kerngeräten für Amazon CloudWatch. Die Komponente ermöglicht es den Komponenten, CloudWatch Metriken zu veröffentlichen, mit denen Sie die Umgebung des Greengrass-Core-Geräts überwachen und analysieren können. Weitere Informationen finden Sie unter Verwenden von CloudWatch Amazon-Metriken im CloudWatch Amazon-Benutzerhandbuch.

Um eine CloudWatch Metrik mit dieser Komponente zu veröffentlichen, veröffentlichen Sie eine Nachricht zu einem Thema, das diese Komponente abonniert. Standardmäßig abonniert diese Komponente das `cloudwatch/metric/put` lokale Thema „Veröffentlichen/Abonnieren“. Sie können andere Themen angeben, einschließlich AWS IoT Core MQTT-Themen, wenn Sie diese Komponente bereitstellen.

Diese Komponente stapelt Metriken, die sich im selben Namespace befinden, und veröffentlicht sie in regelmäßigen CloudWatch Abständen.

> **Note**
>
> Diese Komponente bietet ähnliche Funktionen wie der CloudWatch Metrik-Konnektor in AWS IoT Greengrass V1. Weitere Informationen finden Sie unter CloudWatch Metrik-Konnektor im AWS IoT Greengrass V1-Entwicklerhandbuch.

Themen

- Versionen
- Typ
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Konfiguration
- Eingabedaten
- Ausgabedaten
- Lizenzen
- Lokale Protokolldatei
- Änderungsprotokoll
- Weitere Informationen finden Sie auch unter

Versionen

Diese Komponente hat die folgenden Versionen:

- 3.2.x
- 3.1.x
- 3.0.x
- 2.1.x
- 2.0.x

Informationen zu den Änderungen in den einzelnen Versionen der Komponente finden Sie im Changelog.

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 mithilfe der Lambda-Launcher-Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

v3.x

Diese Komponente kann auf Kerngeräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

v2.x

Diese Komponente kann nur auf Linux-Kerngeräten installiert werden.

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

3.x

- Python-Version 3.7 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.
- Die Greengrass-Geräterolle muss die `cloudwatch:PutMetricData` Aktion zulassen, wie in der folgenden Beispiel-IAM-Richtlinie gezeigt.

JSON

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

Weitere Informationen finden Sie in der CloudWatchAmazon-Berechtigungsreferenz im CloudWatch Amazon-Benutzerhandbuch.

2.x

- 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.
- Python-Version 3.7 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.
- Die Greengrass-Geräterolle muss die `cloudwatch:PutMetricData` Aktion zulassen, wie in der folgenden Beispiel-IAM-Richtlinie gezeigt.

  JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Effect": "Allow",
      "Resource": "*"
```

```
    }
  ]
}
```

Weitere Informationen finden Sie in der CloudWatchAmazon-Berechtigungsreferenz im CloudWatch Amazon-Benutzerhandbuch.

- Um Ausgabedaten von dieser Komponente zu erhalten, müssen Sie das folgende Konfigurationsupdate für die ältere Abonnement-Router-Komponente (`aws.greengrass.LegacySubscriptionRouter`) zusammenführen, wenn Sie diese Komponente bereitstellen. Diese Konfiguration gibt das Thema an, zu 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"
      }
    }
  }
  ```

  - *`region`*Ersetzen Sie es durch AWS-Region das, was Sie verwenden.

- *version*Ersetzen Sie durch die Version der Lambda-Funktion, die diese Komponente ausführt. Um die Version der Lambda-Funktion zu finden, müssen Sie sich das Rezept für die Version dieser Komponente ansehen, 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 älteren Abonnement-Router jedes Mal aktualisieren, wenn Sie diese Komponente bereitstellen. Dadurch wird sichergestellt, dass Sie die richtige Lambda-Funktionsversion für die Komponentenversion verwenden, die Sie bereitstellen.

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 Endpunkten und Ports, die für den Basisbetrieb erforderlich sind. Weitere Informationen finden Sie unter Geräteverkehr über einen Proxy oder eine Firewall zulassen.

Endpoint	Port	Erforderlich	Description
`monitoring. region.amazonaws.com`	443	Ja	Metriken hochladen. CloudWatch

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.

3.2.0

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 3.2.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <3.0.0	Weich
Token-Austauschdienst	>=0.0.0	Hart

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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <3.0.0	Weich
Token-Austauschdienst	>=0.0.0	Hart

2.1.4 - 2.1.9

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.1.4 bis 2.1.9 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <3.0.0	Hart
Lambda-Launcher	^2.0.0	Hart

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^2.0.0	Hart

2.1.4 - 2.1.8

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.1.4 und 2.1.8 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <3.0.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^2.0.0	Hart

2.1.2 - 2.1.3

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.1.2 und 2.1.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.8.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.7.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.6.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.5.0	Hart
Lambda-Launcher	^2.0.0	Hart

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.4.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.3.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.2.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.3 <2.1.0	Hart
Lambda-Launcher	>=1.0.0	Hart
Lambda-Laufzeiten	>=1,0.0	Weich
Token-Austauschdienst	>=1.0.0	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.

v3.x

`PublishInterval`

(Optional) Die maximale Wartezeit in Sekunden, bis die Komponente gebündelte Metriken für einen bestimmten Namespace veröffentlicht. Um die Komponente so zu konfigurieren, dass

sie Metriken so veröffentlicht, wie sie sie empfängt, d. h. ohne Batchverarbeitung, geben Sie Folgendes an. `0`

Die Komponente veröffentlicht CloudWatch erst, nachdem sie 20 Metriken im selben Namespace empfangen hat, oder nach dem von Ihnen angegebenen Intervall.

> **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 in allen Namespaces, die im Speicher gespeichert werden sollen, bevor die Komponente sie durch neuere Metriken ersetzt.

Dieses Limit gilt, wenn das Kerngerät keine Verbindung zum Internet hat, sodass die Komponente die Metriken zwischenspeichert, 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 Kerngeräts passieren.

Dieser Wert muss mindestens 2.000 Metriken umfassen.

Standard: 5.000 Metriken

`InputTopic`

(Optional) Das Thema, das die Komponente abonniert, um Nachrichten zu empfangen. Wenn Sie `true` for 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) Zeichenkettenwert, der definiert, ob AWS IoT Core MQTT-Themen veröffentlicht und abonniert werden sollen. Unterstützte Werte sind `true` und `false`.

Standard: `false`

`LogLevel`

(Optional) Die Protokollierungsebene für die Komponente. Wählen Sie aus den folgenden Protokollebenen, die hier in der Reihenfolge der Stufen aufgeführt sind:

- `DEBUG`
- `INFO`
- `WARNING`
- `ERROR`
- `CRITICAL`

Standard: `INFO`

`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 verwenden zu können, müssen Sie die folgenden Bibliotheken, einschließlich aller Abhängigkeiten, installieren und sie dem standardmäßigen Greengrass-Systembenutzer zur Verfügung stellen.

- AWS IoT Device SDK v2 für Python
- AWS SDK für Python (Boto3)

Standard: `true`

`PublishRegion`

(Optional) Der Ort AWS-Region , an dem 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 umfasst Lambda-Funktionsparameter. Wir empfehlen, dass Sie nur die folgenden Parameter 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 Wartezeit in Sekunden, bis die Komponente gebündelte Metriken für einen bestimmten Namespace veröffentlicht. Um die Komponente so zu konfigurieren, dass sie Metriken so veröffentlicht, wie sie sie empfängt, d. h. ohne Batchverarbeitung, geben Sie Folgendes an. `0`

Die Komponente veröffentlicht CloudWatch erst, nachdem sie 20 Metriken im selben Namespace empfangen hat, oder nach dem von Ihnen angegebenen Intervall.

> **Note**
>
> Die Komponente garantiert nicht die Reihenfolge, in der Ereignisse veröffentlicht werden.

Dieser Wert kann höchstens 900 Sekunden betragen.

Standard: 10 Sekunden

`MAX_METRICS_TO_RETAIN`

(Optional) Die maximale Anzahl von Metriken in allen Namespaces, die im Speicher gespeichert werden sollen, bevor die Komponente sie durch neuere Metriken ersetzt.

Dieses Limit gilt, wenn das Kerngerät keine Verbindung zum Internet hat, sodass die Komponente die Metriken zwischenspeichert, 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 Kerngeräts passieren.

Dieser Wert muss mindestens 2.000 Metriken umfassen.

Standard: 5.000 Metriken

PUBLISH_REGION

(Optional) Der Ort AWS-Region , für den die 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 läuft in einer isolierten Laufzeitumgebung innerhalb des AWS IoT Greengrass Containers.

Standard: GreengrassContainer

containerParams

(Optional) Ein Objekt, das die Container-Parameter für diese Komponente enthält. Die Komponente verwendet diese Parameter, wenn Sie GreengrassContainer für angebencontainerMode.

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 Themen abonniert. publish/subscribe

Dieses Objekt enthält die folgenden Informationen:

0— Dies ist ein Array-Index als Zeichenfolge.

Ein Objekt, das die folgenden Informationen enthält:

`type`

(Optional) Der publish/subscribe Messaging-Typ, den diese Komponente verwendet, um Nachrichten zu abonnieren. 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 aus, 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 aus 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 Container-Modus)

```
{
  "containerMode": "NoContainer"
}
```

Eingabedaten

Diese Komponente akzeptiert Metriken zum folgenden Thema und veröffentlicht die Metriken unter CloudWatch. Standardmäßig abonniert diese Komponente lokale publish/subscribe Nachrichten.

Weitere Informationen zum Veröffentlichen von Nachrichten aus Ihren benutzerdefinierten Komponenten in dieser Komponente finden Sie unterLokale Nachrichten veröffentlichen/abonnieren.

Ab der 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 zu 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 des `PutMetricData`Vorgangs entsprechen.

Typ: `object` der die folgenden Informationen enthält:

`namespace`

Der benutzerdefinierte Namespace für die Metrikdaten in dieser Anfrage. CloudWatch verwendet Namespaces als Container für metrische Datenpunkte.

> **Note**
>
> Sie können keinen Namespace angeben, der mit der reservierten Zeichenfolge beginnt. `AWS/`

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 zu kleine oder zu große Werte ab. Der Wert muss zwischen `8.515920e-109` und `1.174271e+108` (Basis 10) oder `2e-360` und `2e360` (Basis 2) liegen. CloudWatch unterstützt 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 dem Namen des Kerngeräts entspricht.

Typ: `array` von Objekten, die jeweils die folgenden Informationen enthalten:

`name`

(Optional) Der Name der Dimension.

Typ: `string`

`value`

(Optional) Der Dimensionswert.

Typ: `string`

`timestamp`

(Optional) Die Zeit, zu der die metrischen Daten empfangen wurden, ausgedrückt in Sekunden in Unix-Epochenzeit.

Standardmäßig wird der Zeitpunkt verwendet, zu dem die Komponente die Nachricht empfängt.

Typ: `double`

> **Note**
>
> Wenn Sie diese Komponente zwischen den Versionen 2.0.3 und 2.0.7 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`,`Milliseconds`,`Bytes`,`Kilobytes`,`Megabytes`,`Gigabytes`,`Terabyt`
`Second`,`Kilobytes/Second`,`Megabytes/Second`,`Gigabytes/Second`,`Terabytes/`
`Second`,`Bits/Second`,`Kilobits/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:
>
> - 40 KB Limit für die API-Nutzlast

- 20 Metriken pro API-Anforderung
- 150 Transaktionen pro Sekunde (TPS) für die `PutMetricData`-API

Weitere Informationen finden Sie unter CloudWatch Servicekontingenten 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 publish/subscribe Thema. Weitere Informationen zum Abonnieren von Nachrichten zu diesem Thema in Ihren benutzerdefinierten Komponenten finden Sie unterLokale Nachrichten veröffentlichen/abonnieren.

Sie können diese Komponente optional so konfigurieren, dass sie in einem MQTT-Thema veröffentlicht wird, indem Sie den `PubSubToIoTCore` Konfigurationsparameter auf `true` setzen. 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**
>
> Die Komponentenversionen 2.0.x veröffentlichen Antworten standardmäßig als Ausgabedaten zu einem MQTT-Thema. Sie müssen das Thema `subject` in der Konfiguration für die ältere Abonnement-Router-Komponente als das angeben.

Standardthema: `cloudwatch/metric/put/status`

Example Beispielausgabe: Erfolg

Die Antwort umfasst den Namespace der Metrikdaten und das `RequestId` Feld aus der CloudWatch Antwort.

```
{
  "response": {
    "cloudwatch_rid": "70573243-d723-11e8-b095-75ff2EXAMPLE",
    "namespace": "Greengrass",
    "status": "success"
  }
}
```

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, versucht sie erneut, im nächsten Batch zu veröffentlichen.

Lizenzen

Diese Komponente umfasst die folgende Software/Lizenzierung von Drittanbietern:

- AWS SDK für Python (Boto3)/Apache-Lizenz 2.0
- botocore/Apache-Lizenz 2.0
- dateutil/PSF-Lizenz
- docutils/BSD-Lizenz, GNU General Public License (GPL), Python Software Foundation License, Public Domain
- jmespath/MIT-Lizenz
- s3transfer/Apache-Lizenz 2.0
- urllib3/MIT-Lizenz

Diese Komponente wird im Rahmen der Greengrass Core Software-Lizenzvereinbarung 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
```

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.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 den einzelnen Versionen der Komponente beschrieben.

v3.x

Version	Änderungen
3.2.0	Neue Features • Rezeptstützen für Greengrass Nucleus Lite hinzufügen
3.1.0	Fehlerkorrekturen und Verbesserungen • Fügt Unterstützung für HTTPS-Netzwerk-Proxykonfigurationen hinzu. Weitere Informationen erhalten 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 Version 2.x auf Version 3.x durchführen möchten, müssen Sie die Konfiguration der Komponente aktualisieren. Weitere Informationen finden Sie unter Konfiguration der Metrikkomponenten. CloudWatch Neue Features • Integriert die Unterstützung für Kerngeräte, auf denen Windows ausgeführt wird. • Ändert den Komponententyp von einer Lambda-Komponente in eine generische Komponente. Diese Komponente ist jetzt nicht mehr von der älteren Abonnement-Router-Komponente abhängig, um Abonnements zu erstellen.

Version	Änderungen
	• 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. Fehlerkorrekturen und Verbesserungen Integriert die Unterstützung für doppelte Zeitstempel in Eingabedaten.

v2.x

Version	Änderungen
2.1.8	Die Version wurde für die Version 2.13.0 von Greengrass Nucleus aktualisiert.
2.1.3	Die Version wurde für die Version 2.11.0 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 HTTPS-Netzwerk-Proxykonfigurationen hinzu. Weitere Informationen erhalten 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	Fehlerkorrekturen und Verbesserungen • Integriert die Unterstützung für doppelte Zeitstempel in Eingabedaten. • Die Version wurde für die Version 2.5.0 von Greengrass Nucleus aktualisiert.
2.0.7	Die Version wurde für die Version 2.4.0 von Greengrass Nucleus aktualisiert.
2.0.6	Die Version wurde für die Version 2.3.0 von Greengrass Nucleus aktualisiert.
2.0.5	Die Version wurde für die Version 2.2.0 von Greengrass Nucleus aktualisiert.
2.0.4	Die Version wurde für die Version 2.1.0 von Greengrass Nucleus aktualisiert.
2.0.3	Erste Version

Weitere Informationen finden Sie auch unter

- Verwendung von CloudWatch Amazon-Metriken im CloudWatch Amazon-Benutzerhandbuch
- PutMetricDatain der Amazon CloudWatch API-Referenz

AWS IoT Device Defender

Die AWS IoT Device Defender Komponente (`aws.greengrass.DeviceDefender`) benachrichtigt Administratoren über Änderungen im Status der Greengrass-Core-Geräte. 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 Kerngerät. Anschließend veröffentlicht sie die Metriken auf AWS IoT Device Defender. Weitere Informationen zum Lesen und Interpretieren der

von dieser Komponente gemeldeten Metriken finden Sie im AWS IoT Core Entwicklerhandbuch unter Spezifikation der Gerätemetriken.

> **Note**
>
> Diese Komponente bietet ähnliche Funktionen wie der Device Defender-Connector 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 den Änderungen in den einzelnen Versionen der Komponente finden Sie im Changelog.

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 mithilfe der Lambda-Launcher-Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

v3.x

Diese Komponente kann auf Kerngeräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

v2.x

Diese Komponente kann nur auf Linux-Kerngeräten installiert werden.

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

v3.x

- Python-Version 3.7 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.
- AWS IoT Device Defender konfiguriert, um die Erkennungsfunktion zur Überwachung von Verstößen zu verwenden. Weitere Informationen finden Sie unter Detect im AWS IoT Core Entwicklerhandbuch.

v2.x

- 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.
- Python-Version 3.7 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.
- AWS IoT Device Defender konfiguriert, um die Erkennungsfunktion zur Überwachung von Verstößen zu verwenden. Weitere Informationen finden Sie unter Detect im AWS IoT Core Entwicklerhandbuch.
- Die Psutil-Bibliothek ist auf dem Core-Gerät installiert. Version 5.7.0 ist die neueste Version, von der bestätigt wurde, dass sie mit der Komponente funktioniert.
- Die auf dem Kerngerät installierte CBOR-Bibliothek. Version 1.0.0 ist die neueste Version, von der bestätigt wurde, dass sie mit der Komponente funktioniert.
- Um Ausgabedaten von dieser Komponente zu erhalten, müssen Sie bei der Bereitstellung dieser Komponente das folgende Konfigurationsupdate für die ältere Abonnement-Router-Komponente (`aws.greengrass.LegacySubscriptionRouter`) zusammenführen. Diese Konfiguration gibt das Thema an, zu 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"
    }
  }
}
```

- *region*Ersetzen Sie es durch AWS-Region das, was Sie verwenden.
- *version*Ersetzen Sie durch die Version der Lambda-Funktion, die diese Komponente ausführt. Um die Version der Lambda-Funktion zu finden, müssen Sie sich das Rezept für die Version dieser Komponente ansehen, 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 älteren Abonnement-Router jedes Mal aktualisieren, wenn Sie diese Komponente bereitstellen. Dadurch wird sichergestellt, dass Sie die richtige Lambda-Funktionsversion für die Komponentenversion verwenden, die Sie bereitstellen.

Weitere Informationen finden Sie unter Erstellen von Bereitstellungen.

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.

3.1.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 3.1.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <3.0.0	Weich
Token-Austauschdienst	>=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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <3.0.0	Weich
Token-Austauschdienst	>=0.0.0	Hart

2.0.12 - 2.0.17

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.0.12 bis 2.0.17 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <3.0.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^2.0.0	Hart

2.0.12 - 2.0.16

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.16 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <3.0.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^2.0.0	Hart

2.0.10 - 2.0.11

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.0.10 und 2.0.11 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.8.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.7.0	Hart
Lambda-Launcher	^2.0.0	Hart

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.6.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.5.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.4.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.3.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.2.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.3 <2.1.0	Hart
Lambda-Launcher	>=1.0.0	Hart
Lambda-Laufzeiten	>=1,0.0	Weich
Token-Austauschdienst	>=1.0.0	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.

v3.x

`PublishRetryCount`

Gibt an, wie oft die Veröffentlichung wiederholt wird. Diese Funktion ist in Version 3.1.1 verfügbar.

Das Minimum ist 0.

Das Maximum ist 72.

Standard: 5

`SampleIntervalSeconds`

(Optional) Die Zeitspanne in Sekunden zwischen den einzelnen Zyklen, in denen die Komponente Messwerte 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 verwendet werden soll, um die Abhängigkeiten dieser Komponente zu installieren.

Setzen Sie diesen Wert auf, `false` wenn Sie ein benutzerdefiniertes Skript verwenden möchten, um Abhängigkeiten zu installieren, oder wenn Sie Laufzeitabhängigkeiten in ein vorgefertigtes Linux-Image aufnehmen möchten. Um diese Komponente verwenden zu können, müssen Sie die folgenden Bibliotheken, einschließlich aller Abhängigkeiten, installieren und sie dem standardmäßigen Greengrass-Systembenutzer zur Verfügung stellen.

- AWS IoT Device SDK v2 für Python
- CBOR-Bibliothek. Version 1.0.0 ist die neueste Version, von der bestätigt wurde, dass sie mit der Komponente funktioniert.
- Psutil-Bibliothek. Version 5.7.0 ist die neueste Version, von der bestätigt wurde, dass sie mit der Komponente funktioniert.

> **Note**
>
> Wenn Sie Version 3.0.0 oder 3.0.1 dieser Komponente auf Kerngeräten 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 keinen Betrieb hinter einem HTTPS-Proxy.

Standard: `true`

v2.x

> **Note**
>
> Die Standardkonfiguration dieser Komponente umfasst Lambda-Funktionsparameter. Wir empfehlen, dass Sie nur die folgenden Parameter 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 ohne Container-Modus auszuführen, geben Sie `/proc` für diesen Parameter Folgendes an.

Standard: `/host-proc`. Dies ist der Standardpfad, in dem diese Komponente den `/proc` Ordner im Container mountet.

> **Note**
>
> Diese Komponente hat nur Lesezugriff auf diesen Ordner.

`SAMPLE_INTERVAL_SECONDS`

(Optional) Die Zeitspanne in Sekunden zwischen den einzelnen Zyklen, in denen die Komponente Messwerte 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 läuft nicht in einer isolierten Laufzeitumgebung.

  Wenn Sie diese Option angeben, müssen Sie sie `/proc` für die `PROCFS_PATH` Umgebungsvariable 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 Speichermenge (in Kilobyte), die der Komponente zugewiesen werden soll.

Der Standardwert ist 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 Themen abonniert. publish/subscribe

Dieses Objekt enthält die folgenden Informationen:

`0`— Dies ist ein Array-Index als Zeichenfolge.

Ein Objekt, das die folgenden Informationen enthält:

`type`

(Optional) Der publish/subscribe Messaging-Typ, den diese Komponente verwendet, um Nachrichten zu abonnieren. 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 aus, 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 aus 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 Container-Modus)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "PROCFS_PATH": "/proc"
    }
  },
  "containerMode": "NoContainer"
}
```

Eingabedaten

Diese Komponente akzeptiert keine Nachrichten als Eingabedaten.

Ausgabedaten

Diese Komponente veröffentlicht Sicherheitsmetriken zu dem folgenden reservierten Thema für AWS IoT Device Defender. Diese Komponente wird *coreDeviceName* bei der Veröffentlichung der Messwerte durch den Namen des Kerngeräts ersetzt.

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 in der 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
```

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.DeviceDefender.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.DeviceDefender.log -Tail 10 -
  Wait
  ```

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.

v3.x

Version	Änderungen
3.1.1	Fehlerkorrekturen und Verbesserungen • Fügt Wiederholungsversuche für die Client-Verbindung hinzu, wenn die Verbindung nach einem Netzwerkausfall nicht wiederhergestellt werden kann.

Version	Änderungen
	• Fügt einen konfigurierbaren Wiederholungsversuch für die Veröffentlichung von Messwerten hinzu.
3.1.0	Fehlerkorrekturen und Verbesserungen • Fügt Unterstützung für HTTPS-Netzwerk-Proxykonfigurationen hinzu. Weitere Informationen erhalten 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 mit der Berechnung von Deltawerten für Metriken durch die Komponente.
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
2.0.17	Die Version wurde für die Version 2.14.0 von Greengrass Nucleus aktualisiert.
2.0.16	Die Version wurde für die Version 2.13.0 von Greengrass Nucleus aktualisiert.
2.0.11	Die Version wurde für die Version 2.11.0 von Greengrass Nucleus aktualisiert.
2.0.10	Die Version wurde für die Version 2.7.0 von Greengrass Nucleus aktualisiert.

Version	Änderungen
2.0.9	Die Version wurde für die Version 2.6.0 von Greengrass Nucleus aktualisiert.
2.0.8	Die Version wurde für die Version 2.5.0 von Greengrass Nucleus aktualisiert.
2.0.7	Die Version wurde für die Version 2.4.0 von Greengrass Nucleus aktualisiert.
2.0.6	Die Version wurde für die Version 2.3.0 von Greengrass Nucleus aktualisiert.
2.0.5	Die Version wurde für die Version 2.2.0 von Greengrass Nucleus aktualisiert.
2.0.4	Die Version wurde für die Version 2.1.0 von Greengrass Nucleus aktualisiert.
2.0.3	Erste Version

Festplattenspooler

Die Festplatten-Spooler-Komponente (`aws.greengrass.DiskSpooler`) bietet eine persistente Speicheroption für Nachrichten, auf die von Greengrass-Kerngeräten gespoolt wird. AWS IoT Core Diese Komponente speichert diese ausgehenden Nachrichten auf der Festplatte.

Themen

- Versionen
- Typ
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Usage
- Lokale Protokolldatei

- Änderungsprotokoll

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 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 AWS IoT Greengrass Protokolle überwachen.

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:

- `storageType`sollte so eingestellt sein, dass `Disk` diese Komponente verwendet werden soll. Sie können dies in der Greengrass-Nukleus-Konfiguration einstellen.
- `maxSizeInBytes`darf nicht so konfiguriert sein, dass es den verfügbaren Speicherplatz auf dem Gerät übersteigt. Sie können dies in der Greengrass-Nukleus-Konfiguration einstellen.
- Die Festplattenspooler-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.

1.0.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.0.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.11.0 <2.17.0	Hart

1.0.

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.0.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.11.0 <2.16.0	Hart

1.0.5

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.0.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.11.0 <2.15.0	Hart

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	>=2.11.0 <2.14.0	Hart

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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=2.11.0 <2.12.0	Hart

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zu den Komponentenrezepten.

Usage

Um die Disk-Spooler-Komponente verwenden zu können, `aws.greengrass.DiskSpooler` muss diese bereitgestellt werden.

Um diese Komponente zu konfigurieren und zu verwenden, müssen Sie den `pluginName` Wert auf setzen. `aws.greengrass.DiskSpooler`

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
1.0.7	Die Version wurde für die Version 2.16.0 von Greengrass Nucleus aktualisiert.
1.0.6	Die Version wurde für die Version 2.15.0 von Greengrass Nucleus aktualisiert.
1.0.5	Die Version wurde für die Version 2.14.0 von Greengrass Nucleus aktualisiert.

Version	Änderungen
1.0.4	Fehlerkorrekturen und Verbesserungen Allgemeine Fehlerkorrekturen.
1.0.3	Fehlerkorrekturen und Verbesserungen Verbessert die Leistung durch die Wiederverwendung von Datenbank verbindungen.
1.0.2	Fehlerkorrekturen und Verbesserungen Behebt ein Problem, bei dem das Feld für das MQTT-Nachrichtenformat in bestimmten Fällen nicht dauerhaft angezeigt wird.
1.0.1	Die Version wurde für die Version 2.12.0 von Greengrass Nucleus aktualisiert.
1.0.0	Erste Version

Docker-Anwendungsmanager

Die Docker-Anwendungsmanager-Komponente (`aws.greengrass.DockerApplicationManager`) ermöglicht AWS IoT Greengrass das Herunterladen von Docker-Images aus öffentlichen Image-Registern und privaten Registern, die auf Amazon Elastic Container Registry (Amazon ECR) gehostet werden. Es ermöglicht auch AWS IoT Greengrass die automatische Verwaltung von Anmeldeinformationen, um Bilder sicher aus privaten Repositorys in Amazon ECR herunterzuladen.

Wenn Sie eine benutzerdefinierte Komponente entwickeln, die einen Docker-Container ausführt, fügen Sie den Docker-Anwendungsmanager als Abhängigkeit hinzu, um die Docker-Images herunterzuladen, die in Ihrer Komponente als Artefakte angegeben sind. Weitere Informationen finden Sie unter Führen Sie einen Docker-Container aus.

Themen

- Versionen
- Typ
- Betriebssystem

- Voraussetzungen
- Abhängigkeiten
- Konfiguration
- Lokale Protokolldatei
- Änderungsprotokoll
- Weitere Informationen finden Sie auch unter

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 Kerngeräten 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 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.
- Docker-Images, die in einer der folgenden unterstützten Bildquellen gespeichert sind:
  - Öffentliche und private Image-Repositorys in Amazon Elastic Container Registry (Amazon ECR)
  - Öffentliches Docker Hub-Repository
  - Vertrauenswürdiges öffentliches Docker-Register
- Docker-Images, die als Artefakte in Ihren benutzerdefinierten Docker-Container-Komponenten enthalten sind. Verwenden Sie die folgenden URI-Formate, um Ihre Docker-Images zu spezifizieren:
  - 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 finden Sie unter Führen Sie einen Docker-Container aus.

  > **Note**
  >
  > Wenn Sie das Image-Tag oder den Image-Digest nicht in der Artefakt-URI für ein Image angeben, ruft der Docker-Anwendungsmanager die neueste verfügbare Version dieses Images ab, wenn Sie Ihre benutzerdefinierte Docker-Container-Komponente bereitstellen. Um sicherzustellen, dass auf allen Ihren Kerngeräten dieselbe Version eines Images ausgeführt wird, empfehlen wir Ihnen, das Image-Tag oder den Image-Digest in die Artefakt-URI aufzunehmen.
- 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 ausggc_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 Benutzerggc_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 Benutzerggc_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
```

- Wenn Sie die AWS IoT Greengrass Core-Software für die Verwendung eines Netzwerk-Proxys konfigurieren, müssen Sie Docker für die Verwendung desselben Proxyservers konfigurieren.
- Wenn Ihre Docker-Images in einer privaten Amazon ECR-Registry 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.

  JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
```

```
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

- Die Docker-Anwendungsmanager-Komponente wird für die Ausführung in einer VPC unterstützt. Um diese Komponente in einer VPC bereitzustellen, ist Folgendes erforderlich.
  - Die Docker-Anwendungsmanager-Komponente muss über Konnektivität zum Herunterladen von Bildern verfügen. 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 Endpunkten und Ports, die für den Basisbetrieb erforderlich sind. Weitere Informationen finden Sie unter Geräteverkehr über einen Proxy oder eine Firewall zulassen.

Endpoint	Port	Erforderlich	Description
`ecr.region.amazonaws.com`	443	Nein	Erforderlich, wenn Sie Docker-Images von Amazon

Endpoint	Port	Erforderlich	Description
			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 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.

2.0.15

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.15 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.1.0 <2.17.0	Weich

2.0.14

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.14 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.1.0 <2.16.0	Weich

2.0.13

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.13 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.1.0 <2.15.0	Weich

2.0.12

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.12 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.1.0 <2.14.0	Weich

2.0.11

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.11 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=2.1.0 <2.2.0	Weich

Weitere Informationen zu Komponentenabhängigkeiten finden Sie in der Referenz zu den Komponentenrezepten.

Konfiguration

Diese Komponente hat keine Konfigurationsparameter.

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
2.0.15	Die Version wurde für die Version 2.16.0 von Greengrass Nucleus aktualisiert.
2.0.14	Die Version wurde für die Version 2.15.0 von Greengrass Nucleus aktualisiert.
2.0.13	Die Version wurde für die Version 2.14.0 von Greengrass Nucleus aktualisiert.
2.0.12	Die Version wurde für die Version 2.13.0 von Greengrass Nucleus aktualisiert.
2.0.11	Die Version wurde für die Version 2.12.0 von Greengrass Nucleus aktualisiert.
2.0.10	Die Version wurde für die Version 2.11.0 von Greengrass Nucleus aktualisiert.
2.0.9	Die Version wurde für die Version 2.10.0 von Greengrass Nucleus aktualisiert.
2.0.8	Die Version wurde für die Version 2.9.0 von Greengrass Nucleus aktualisiert.
2.0.7	Die Version wurde für die Version 2.8.0 von Greengrass Nucleus aktualisiert.

Version	Änderungen
2.0.6	Die Version wurde für die Version 2.7.0 von Greengrass Nucleus aktualisiert.
2.0.5	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	Die Version wurde für die Version 2.4.0 von Greengrass Nucleus aktualisiert.
2.0.2	Die Version wurde für die Version 2.3.0 von Greengrass Nucleus aktualisiert.
2.0.1	Die Version wurde für die Version 2.2.0 von Greengrass Nucleus 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-Zielvideostreams 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**
>
> Die Komponente Edge-Connector für Kinesis Video Streams ist nur in den folgenden AWS-Regionen Fällen verfügbar:
>
> - USA Ost (Nord-Virginia)
> - USA West (Oregon)
> - Europe (Frankfurt)
> - Europa (Irland)
> - Asien-Pazifik (Singapur)
> - Asien-Pazifik (Tokio)
> - Asien-Pazifik (Seoul)
> - Asien-Pazifik (Sydney)
> - Asien-Pazifik (Mumbai)
> - China (Peking)

Themen

- Versionen
- Typ
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Konfiguration
- Lizenzen
- Usage
- Lokale Protokolldatei
- Änderungsprotokoll
- Weitere Informationen finden Sie auch unter

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.

- GStreamerVersion 1.18.4 oder höher ist auf dem Kerngerät installiert. Weitere Informationen finden Sie unter Installation. GStreamer

  Auf einem Gerät mit `apt` können Sie die folgenden Befehle zur Installation ausführen GStreamer.

  ```
  sudo apt install -y libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev
   gstreamer1.0-plugins-base-apps
  sudo apt install -y gstreamer1.0-libav
  sudo apt install -y gstreamer1.0-plugins-bad gstreamer1.0-plugins-good gstreamer1.0-
  plugins-ugly gstreamer1.0-tools
  ```

- 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 der Schlüssel und der Wert die URL für die Kamera ist. `RTSPStreamUrl` 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.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Effect": "Allow",
      "Resource": [
      "arn:aws:secretsmanager:us-east-1:123456789012:secret:IPCamera1Url",
  "arn:aws:secretsmanager:us-east-1:123456789012:secret:IPCamera2Url"
      ]
    },
    {
      "Action": [
        "iotsitewise:BatchPutAssetPropertyValue",
        "iotsitewise:DescribeAsset",
        "iotsitewise:DescribeAssetModel",
        "iotsitewise:DescribeAssetProperty",
        "iotsitewise:GetAssetPropertyValue",
        "iotsitewise:ListAssetRelationships",
        "iotsitewise:ListAssets",
        "iotsitewise:ListAssociatedAssets",
        "kinesisvideo:CreateStream",
        "kinesisvideo:DescribeStream",
        "kinesisvideo:GetDataEndpoint",
        "kinesisvideo:PutMedia",
```

```
        "kinesisvideo:TagStream"
      ],
      "Effect": "Allow",
      "Resource": [
        "*"
      ]
    }
  ]
}
```

> **Note**
>
> 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 Geräteverkehr über einen Proxy oder eine Firewall zulassen.

Endpoint	Port	Erforderlich	Description
`kinesisvideo.` *`region`*`.amazonaws.com`	443	Ja	Laden Sie Daten in Kinesis Video Streams hoch.
`data.iotsitewise.` *`region`*`.amazonaws.com`	443	Ja	Veröffentlichen Sie Videostream-Metadaten auf

Endpoint	Port	Erforderlich	Description
			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 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

Usage

Um diese Komponente zu konfigurieren und mit ihr zu interagieren, können Sie Eigenschaften für die Komponenten festlegen, die das AWS IoT SiteWise Kerngerät und die IP-Kameras repräsentieren, mit denen es verbunden ist. Sie können auch Videostreams in Grafana-Dashboards 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 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
- Secrets im AWS Secrets Manager Benutzerhandbuch 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
```

Mit dem folgenden Befehl wird beispielsweise eine Komponente mit dem Namen `com.example.HelloWorld` neu gestartet.

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
- Typ
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Konfiguration
- Lokale Protokolldatei
- Änderungsprotokoll

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.16.x
- 2.15.x
- 2.14.x
- 2.13.x
- 2.12.x
- 2.11.x
- 2.10.x
- 2.9.x

- 2.8.x
- 2.7.x
- 2.6.x
- 2.5.x
- 2.4.x
- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

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 AWS IoT Greengrass Protokolle überwachen.

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.

2.16.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.16.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.12.0 <2.17.0	Weich

2.15.1

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.15.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.12.0 <2.16.0	Weich

2.15.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.15.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.12.0 <2.16.0	Weich

2.14.0 – 2.14.3

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.14.0 und 2.14.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.12.0 <2.15.0	Weich

2.13.0

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.13.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.12.0 <2.14.0	Weich

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 - 2.14.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 angeben. IDs `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 angeben. IDs `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 Zusammenführung von Konfigurationen

Die folgende Beispielkonfiguration legt fest, dass drei POSIX-Systemgruppen (`group11002`, und`group3`) und zwei Windows-Benutzergruppen (`Device 0peratorsundQA 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 angeben. IDs `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 (`group11002`, 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
2.16.1	Die Version wurde für die Version 2.16.1 von Greengrass Nucleus aktualisiert.
2.16.0	Die Version wurde für die Version 2.16.0 von Greengrass Nucleus aktualisiert.
2.15.1	Die Version wurde für die Version 2.15.1 von Greengrass Nucleus aktualisiert.
2.15.0	Fehlerkorrekturen und Verbesserungen • Fügt einen Fix hinzu, sodass die Komponente nicht neu installiert wird, wenn sich die Authentifizierungskonfiguration ändert, sondern trotzdem neu gestartet wird. • Allgemeine Fehlerbehebungen und Verbesserungen.
2.14.3	Die Version wurde für die Version 2.14.3 von Greengrass Nucleus aktualisiert.
2.14.2	Die Version wurde für die Version 2.14.2 von Greengrass Nucleus aktualisiert.
2.14.1	Die Version wurde für die Version 2.14.1 von Greengrass Nucleus aktualisiert.

Version	Änderungen
2.14.0	**Warning** Diese Version ist nicht mehr verfügbar. Die Verbesserungen in dieser Version sind in späteren Versionen dieser Komponente verfügbar. Fehlerkorrekturen und Verbesserungen • Überprüfen Sie den Zielparameter für die Bereitstellung im CLI-Befehl.
2.13.0	Die Version wurde für die Version 2.13.0 von Greengrass Nucleus aktualisiert.
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.
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.

Version	Änderungen
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.
2.7.0	Die Version wurde für die Version 2.7.0 von Greengrass Nucleus aktualisiert.

Version	Änderungen
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.
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.

Version	Änderungen
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 verbunden sind.

> **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 Sie 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.

Themen

- Versionen
- Typ
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Konfiguration
- Lokale Protokolldatei
- Änderungsprotokoll

Versionen

Diese Komponente hat die folgenden Versionen:

- 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 AWS IoT Greengrass Protokolle überwachen.

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 erhalten Sie unter AWS IoT Richtlinien für den Betrieb 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 Endpunkte von Kerngeräten verwalten.
- 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.

2.2.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.2.3 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.2.0 <2.17.0	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 <2.16.0	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 <2.15.0	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	>=2.2.0 <2.14.0	Weich

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.

2.2.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, über die 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 IPv4 verknüpfungslokale 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`

`includeIPv6LoopbackAddrs`

(Optional) Sie können diese Option aktivieren, um IPv6 Loopback-Adressen zu erkennen und zu melden. Dies sind IP-Adressen, über die 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. Sie müssen `includeIPv4Addrs` auf `false` und auf einstellen`includeIPv6Addrs`, `true` um diese Option verwenden zu können.

Standard: `false`

`includeIPv6LinkLocalAddrs`

(Optional) Sie können diese Option aktivieren, um IPv6 verknüpfungslokale 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. Sie müssen `includeIPv4Addrs` auf `false` und auf einstellen, `includeIPv6Addrs` um diese Option verwenden `true` zu können.

Standard: `false`

`includeIPv4Addrs`

(Optional) Die Standardeinstellung ist auf eingestellt`true`. Sie können diese Option aktivieren, um IPv4 Adressen zu veröffentlichen, die auf dem Core-Gerät gefunden wurden.

Standard: `true`

`includeIPv6Addrs`

(Optional) Sie können diese Option aktivieren, um IPv6 Adressen zu veröffentlichen, die auf dem Kerngerät gefunden wurden. Stellen Sie `includeIPv4Addrs` auf `false` ein, um diese Option zu verwenden.

Standard: `false`

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, über die 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 IPv4 verknüpfungslokale 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, über die 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 IPv4 verknüpfungslokale 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
2.2.3	Die Version wurde für die Version 2.16.0 von Greengrass Nucleus aktualisiert.
2.2.2	Die Version wurde für die Version 2.15.0 von Greengrass Nucleus aktualisiert.
2.2.1	Die Version wurde für die Version 2.14.0 von Greengrass Nucleus aktualisiert.
2.2.0	Die Version wurde für die Version 2.13.0 von Greengrass Nucleus aktualisiert. Neue Features • Fügt Unterstützung für hinzu. IPv6 Sie können es jetzt IPv6 für lokales Messaging verwenden.
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.

Version	Änderungen
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.
2.0.0	Erste Version

Firehose

Die Firehose-Komponente (`aws.greengrass.KinesisFirehose`) veröffentlicht Daten über Amazon Data Firehose-Lieferdatenströme an Ziele wie Amazon S3, Amazon Redshift und Amazon Service. OpenSearch Weitere Informationen finden Sie unter Was ist Amazon Data Firehose? im Amazon Data Firehose Developer Guide.

Um mit dieser Komponente in einem Kinesis-Lieferstream zu veröffentlichen, veröffentlichen Sie eine Nachricht zu einem Thema, das diese Komponente abonniert. Standardmäßig abonniert diese Komponente die Themen „Publizieren/Abonnieren“ `kinesisfirehose/message` und „`kinesisfirehose/message/binary/#`Lokales Veröffentlichen“. Sie können andere Themen angeben, einschließlich AWS IoT Core MQTT-Themen, wenn Sie diese Komponente bereitstellen.

> **Note**
>
> Diese Komponente bietet ähnliche Funktionen wie der Firehose-Anschluss in AWS IoT Greengrass V1. Weitere Informationen finden Sie unter Firehose-Anschluss im AWS IoT Greengrass V1-Entwicklerhandbuch.

Themen

- Versionen
- Typ
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Konfiguration
- Eingabedaten
- Ausgabedaten
- Lokale Protokolldatei
- 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 mithilfe der Lambda-Launcher-Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann nur auf Linux-Kerngeräten installiert werden.

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.
- Python-Version 3.7 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.
- Die Greengrass-Geräterolle muss die `firehose:PutRecordBatch` Aktionen `firehose:PutRecord` und zulassen, wie in der folgenden IAM-Beispielrichtlinie gezeigt.

  JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "firehose:PutRecord",
        "firehose:PutRecordBatch"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:firehose:us-east-1:123456789012:deliverystream/stream-name"
      ]
    }
  ]
}
```

Sie können den Standardzustellungsstream 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 bei der Bereitstellung dieser Komponente das folgende Konfigurationsupdate für die ältere Abonnement-Router-Komponente (`aws.greengrass.LegacySubscriptionRouter`) zusammenführen. Diese Konfiguration gibt das Thema an, zu 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"
      }
    }
  }
  ```

  - *region*Ersetzen Sie es durch AWS-Region das, was Sie verwenden.
  - *version*Ersetzen Sie durch die Version der Lambda-Funktion, die diese Komponente ausführt. Um die Version der Lambda-Funktion zu finden, müssen Sie sich das Rezept für die Version dieser Komponente ansehen, 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 älteren Abonnement-Router jedes Mal aktualisieren, wenn Sie diese Komponente bereitstellen. Dadurch wird sichergestellt, dass Sie die richtige Lambda-Funktionsversion für die Komponentenversion verwenden, die Sie bereitstellen.

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 eine Konnektivität verfügen`firehose.region.amazonaws.com`, zu der der VPC-Endpunkt ist. `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 Endpunkten und Ports, die für den Basisbetrieb erforderlich sind. Weitere Informationen finden Sie unter Geräteverkehr über einen Proxy oder eine Firewall zulassen.

Endpoint	Port	Erforderlich	Description
`firehose.` *`region`*`.amazonaws.com`	443	Ja	Laden Sie Daten auf Firehose hoch.

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.

2.1.10

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 <2.16.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^2.0.0	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 <2.15.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^2.0.0	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 <2.14.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.13.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.12.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.11.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.10.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.9.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.8.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.7.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.6.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.5.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.4.0	Hart
Lambda-Launcher	^2.0.0	Hart

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.3.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.2.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.3 <2.1.0	Hart
Lambda-Launcher	>=1.0.0	Hart
Lambda-Laufzeiten	>=1,0.0	Weich
Token-Austauschdienst	>=1.0.0	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**
>
> Die Standardkonfiguration dieser Komponente umfasst Lambda-Funktionsparameter. Wir empfehlen, dass Sie nur die folgenden Parameter 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 Standard-Firehose-Lieferstreams, in den die Komponente Daten sendet. Sie können den Zielstream mit der `delivery_stream_arn` Eigenschaft in der Nutzlast der Eingabenachricht überschreiben.

> **Note**
>
> Die zentrale Geräterolle muss die erforderlichen Aktionen für alle Zielzustellungsstreams zulassen. Weitere Informationen finden Sie unter Voraussetzungen.

`PUBLISH_INTERVAL`

(Optional) Die maximale Anzahl von Sekunden, die gewartet werden soll, bevor die Komponente Batchdaten in Firehose veröffentlicht. Um die Komponente so zu konfigurieren, dass sie Metriken so veröffentlicht, wie sie sie empfängt, d. h. ohne Batchverarbeitung, geben Sie Folgendes 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 Lieferstream zurückweist.

Dieser Wert muss mindestens 2.000 Datensätze betragen.

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 läuft in einer isolierten Laufzeitumgebung innerhalb des AWS IoT Greengrass Containers.

Standard: `GreengrassContainer`

`containerParams`

(Optional) Ein Objekt, das die Container-Parameter 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 Themen abonniert. publish/subscribe

Dieses Objekt enthält die folgenden Informationen:

`0`— Dies ist ein Array-Index als Zeichenfolge.

Ein Objekt, das die folgenden Informationen enthält:

`type`

(Optional) Der publish/subscribe Messaging-Typ, den diese Komponente verwendet, um Nachrichten zu abonnieren. 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 aus, 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 aus 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 Container-Modus)

```
{
  "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 die Inhalte an den Ziel-Lieferstream. 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 Firehose-Ziel-Lieferstreams. Geben Sie diese Eigenschaft an, um den Standard-Lieferstream 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, setzt die Komponente die `id` Eigenschaft im Antwortobjekt auf diesen Wert.

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 (lokales Veröffentlichen/Abonnieren): `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 () `/` einzufügen.

Ausgabedaten

Diese Komponente veröffentlicht standardmäßig Antworten als Ausgabedaten zum folgenden MQTT-Thema. Sie müssen dieses Thema `subject` in der Konfiguration für die ältere Abonnement-Router-Komponente angeben. Weitere Informationen zum Abonnieren von Nachrichten zu diesem Thema in Ihren benutzerdefinierten Komponenten finden Sie unterMQTT-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, versucht sie erneut, im nächsten Batch zu veröffentlichen.

Lokale Protokolldatei

Diese Komponente verwendet die folgende Protokolldatei.

```
/greengrass/v2/logs/aws.greengrass.KinesisFirehose.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.KinesisFirehose.log
  ```

Lizenzen

Diese Komponente umfasst die folgende Software/Lizenzierung von Drittanbietern:

- AWS SDK für Python (Boto3)/Apache-Lizenz 2.0
- botocore/Apache-Lizenz 2.0
- dateutil/PSF-Lizenz
- docutils/BSD-Lizenz, GNU General Public License (GPL), Python Software Foundation License, Public Domain
- jmespath/MIT-Lizenz
- s3transfer/Apache-Lizenz 2.0
- urllib3/MIT-Lizenz

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
2.1.10	Die Version wurde für die Version 2.15.0 von Greengrass Nucleus aktualisiert.
2.1.9	Die Version wurde für die Version 2.14.0 von Greengrass Nucleus aktualisiert.
2.1.8	Die Version wurde für die Version 2.13.0 von Greengrass Nucleus aktualisiert.
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	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 HTTPS-Netzwerk-Proxykonfigurationen hinzu. Weitere Informationen erhalten Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy und Ermöglichen Sie dem Kerngerät, einem HTTPS-Proxy zu vertrauen.
2.0.8	Die Version wurde für die Version 2.5.0 von Greengrass Nucleus aktualisiert.
2.0.7	Die Version wurde für die Version 2.4.0 von Greengrass Nucleus aktualisiert.
2.0.6	Die Version wurde für die Version 2.3.0 von Greengrass Nucleus aktualisiert.
2.0.5	Die Version wurde für die Version 2.2.0 von Greengrass Nucleus aktualisiert.

Version	Änderungen
2.0.4	Die Version wurde für die Version 2.1.0 von Greengrass Nucleus aktualisiert.
2.0.3	Erste Version

Weitere Informationen finden Sie auch unter

- Was ist Amazon Data Firehose? im Amazon Data Firehose Developer Guide

Lambda-Launcher

Die Lambda-Launcher-Komponente (`aws.greengrass.LambdaLauncher`) startet und stoppt AWS Lambda Funktionen auf AWS IoT Greengrass Kerngeräten. Diese Komponente richtet auch jegliche Containerisierung ein und führt Prozesse unter den von Ihnen angegebenen Benutzern aus.

> **Note**
>
> Wenn Sie eine Lambda-Funktionskomponente auf einem Kerngerät bereitstellen, umfasst die Bereitstellung auch diese Komponente. Weitere Informationen finden Sie unter AWS Lambda Funktionen ausführen.

Themen

- Versionen
- Typ
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Konfiguration
- Lokale Protokolldatei
- Änderungsprotokoll

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 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-Launcher-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.

2.0.11 – 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	Art der Abhängigkeit
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	Art der Abhängigkeit
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	Art der Abhängigkeit
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	Art der Abhängigkeit
Lambda-Manager	>=2.0.3 <2.1.0	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.

```
/greengrass/v2/logs/lambdaFunctionComponentName.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 und *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 den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
2.0.13	Fehlerkorrekturen und Verbesserungen Allgemeine Fehlerbehebungen und Verbesserungen.
2.0.12	Fehlerkorrekturen und Verbesserungen Behebt ein Problem, bei dem der Lambda-Launcher einen Fehler ausgeben konnte, wenn der vorherige Prozess nicht ordnungsgemäß gestoppt wurde.

Version	Änderungen
2.0.11	Support für Lambda Manager 2.3.0.
2.0.10	Fehlerkorrekturen und Verbesserungen • Allgemeine Fehlerbehebungen und Verbesserungen.
2.0.9	Die Version wurde für die Version 2.5.0 von Greengrass Nucleus aktualisiert.
2.0.8	Die Version wurde für die Version 2.4.0 von Greengrass Nucleus aktualisiert.
2.0.7	Die Version wurde für die Version 2.3.0 von Greengrass Nucleus aktualisiert.
2.0.6	Allgemeine Leistungssteigerungen und Bugfixes.
2.0.4	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem die Komponente nicht korrekt `AddGroupOwner` an den Lambda-Funktionscontainer übergeben wird.
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 AWS Lambda Funktionen ausführen.

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 AWS IoT Greengrass Protokolle überwachen.

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.

2.3.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.3.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.17.0	Weich

2.3.6

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.3.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.16.0	Weich

2.3.5

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.3.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.15.0	Weich

2.3.4

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.3.4 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.14.0	Weich

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
2.3.7	Die Version wurde für die Version 2.16.0 von Greengrass Nucleus aktualisiert.
2.3.6	Die Version wurde für die Version 2.15.0 von Greengrass Nucleus aktualisiert.
2.3.5	Fehlerkorrekturen und Verbesserungen • Verbessert die Leistung, indem epoll anstelle von no verwendet wird, sofern verfügbar.
2.3.4	Die Version wurde für die Version 2.13.0 von Greengrass Nucleus aktualisiert.
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.

Version	Änderungen
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` . 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.

Version	Änderungen
2.2.5	Neue Features • Fügt Unterstützung für MQTT-Themen-Platzhalter in Ereignisquellen hinzu, in denen Sie lokale Nachrichten abonnieren. publish/subscribe 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.
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

Version	Änderungen
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.
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-Laufzeitkomponente (`aws.greengrass.LambdaRuntimes`) stellt die Laufzeiten bereit, die Greengrass-Kerngeräte zum Ausführen von Funktionen verwenden. AWS Lambda

> **Note**
>
> Wenn Sie eine Lambda-Funktionskomponente auf einem Kerngerät bereitstellen, umfasst die Bereitstellung auch diese Komponente. Weitere Informationen finden Sie unter AWS Lambda Funktionen ausführen.

Themen

- Versionen
- Typ
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten

- Konfiguration
- Lokale Protokolldatei
- Änderungsprotokoll

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 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-Laufzeitkomponente 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 den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
2.0.9	Fehlerkorrekturen und Verbesserungen Behebt eine Syntaxwarnung mit Python 3.12
2.0.8	Die Version wurde für die Version 2.5.0 von Greengrass Nucleus aktualisiert.
2.0.7	Die Version wurde für die Version 2.4.0 von Greengrass Nucleus aktualisiert.
2.0.6	Die Version wurde für die Version 2.3.0 von Greengrass Nucleus aktualisiert.
2.0.5	Die Version wurde für die Version 2.2.0 von Greengrass Nucleus aktualisiert.
2.0.4	Die Version wurde für die Version 2.1.0 von Greengrass Nucleus aktualisiert.
2.0.3	Erste Version

Legacy-Abonnement-Router

Der ältere 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-Messaging auf einem Kerngerät 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 Connector-Komponenten und Lambda-Funktionskomponenten zu aktivieren, die das AWS IoT Greengrass Core SDK verwenden.

> **Note**
>
> Die ältere Abonnement-Router-Komponente 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 IPC-Schnittstelle (Interprocess Communication) in AWS IoT Device SDK V2 zu verwenden, müssen Sie die ältere Abonnement-Router-Komponente nicht bereitstellen. Weitere Informationen finden Sie in den folgenden Interprozesskommunikationsdiensten:
>
> - Lokale Nachrichten veröffentlichen/abonnieren
> - MQTT-Nachrichten veröffentlichen/abonnieren AWS IoT Core

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 nur auf Linux-Core-Geräten installiert werden.

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

- Der ältere Abonnement-Router wird für die Ausführung in einer VPC unterstützt.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt er 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.

2.1.15

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.15 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.17.0	Weich

2.1.14

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.14 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.16.0	Weich

2.1.13

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.13 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.15.0	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 <2.14.0	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 <2.13.0	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

2.0.3

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.3 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.

v2.1.x

`subscriptions`

(Optional) Die Abonnements, die auf dem Kerngerät aktiviert werden sollen. Dabei handelt es sich um ein Objekt, bei dem jeder Schlüssel eine eindeutige ID ist und jeder Wert ein Objekt ist, das das Abonnement für diesen Connector definiert. Sie müssen ein Abonnement konfigurieren, wenn Sie eine V1-Connectorkomponente 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 Kerngerät.
  Geben Sie den Komponentennamen mit dem `component:` Präfix an, z. B. **`component:com.example.HelloWorldLambda`**
- Der Amazon-Ressourcenname (ARN) einer Lambda-Funktion auf dem Kerngerät.

> **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 dem Abonnement entspricht.
> 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-Connector-Komponente bereitzustellen, geben Sie den Namen der Komponente oder den ARN der Lambda-Funktion der Connector-Komponente an.

`subject`

Das MQTT-Thema oder der Themenfilter, auf dem Quelle und Ziel Nachrichten veröffentlichen und empfangen können. Dieser Wert unterstützt die Platzhalter + und # Topic.

`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 Kerngerät veröffentlicht. Geben Sie eines der folgenden Elemente an:

- `cloud`. Die `source` Funktion veröffentlicht MQTT-Nachrichten an AWS IoT Core.

- Der Name einer Lambda-Funktionskomponente auf dem Kerngerät.
  Geben Sie den Komponentennamen mit dem `component:` Präfix an, z. B. **`component:com.example.HelloWorldLambda`**
- Der Amazon-Ressourcenname (ARN) einer Lambda-Funktion auf dem Kerngerät.

> **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 dem Abonnement entspricht.
> 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 spezifiziert, dass die `com.example.HelloWorldLambda` Lambda-Funktionskomponente eine MQTT-Nachricht zu diesem AWS IoT Core Thema veröffentlicht. `hello/world`

```
{
  "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 spezifiziert, dass die `com.example.HelloWorldLambda` Lambda-Funktionskomponente MQTT-Nachrichten zu diesem Thema an die `com.example.MessageRelay` Lambda-Funktionskomponente veröffentlicht. `hello/world`

```
{
  "subscriptions": {
    "Greengrass_HelloWorld_to_MessageRelay": {
      "id": "Greengrass_HelloWorld_to_MessageRelay",
      "source": "component:com.example.HelloWorldLambda",
      "subject": "hello/world",
      "target": "component:com.example.MessageRelay"
    }
  }
}
```

v2.0.x

`subscriptions`

(Optional) Die Abonnements, die auf dem Core-Gerät aktiviert werden sollen. Dabei handelt es sich um ein Objekt, bei dem jeder Schlüssel eine eindeutige ID ist und jeder Wert ein Objekt ist, das das Abonnement für diesen Connector definiert. Sie müssen ein Abonnement konfigurieren, wenn Sie eine V1-Connectorkomponente 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 Kerngerät.

> **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 dem Abonnement entspricht.
> 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-Connector-Komponente bereitzustellen, geben Sie den ARN der Lambda-Funktion der Connector-Komponente an.

`subject`

Das MQTT-Thema oder der Themenfilter, auf dem Quelle und Ziel Nachrichten veröffentlichen und empfangen können. Dieser Wert unterstützt die Platzhalter + und # Topic.

`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 Kerngerät veröffentlicht. Geben Sie eines der folgenden Elemente an:

- `cloud`. Die `source` Funktion veröffentlicht MQTT-Nachrichten an AWS IoT Core.
- Der Amazon-Ressourcenname (ARN) einer Lambda-Funktion auf dem Kerngerät.

> **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 dem Abonnement entspricht.
> 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 `hello/world` diesem Thema 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 spezifiziert, dass die `Greengrass_HelloWorld` Funktion MQTT-Nachrichten zu diesem Thema `Greengrass_MessageRelay` an die `hello/world` 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 den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
2.1.15	Die Version wurde für die Version 2.16.0 von Greengrass Nucleus aktualisiert.
2.1.14	Die Version wurde für die Version 2.15.0 von Greengrass Nucleus aktualisiert.
2.1.13	Die Version wurde für die Version 2.14.0 von Greengrass Nucleus aktualisiert.
2.1.12	Die Version wurde für die Version 2.13.0 von Greengrass Nucleus 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	Fehlerkorrekturen und Verbesserungen • Integriert die Unterstützung für die Angabe von Komponentennamen anstelle von ARNs für `source` und`target`. Wenn Sie einen Komponent ennamen für ein Abonnement angeben, müssen Sie das Abonnement

Version	Änderungen
	nicht jedes Mal neu konfigurieren, wenn sich die Version der Lambda-Funktion 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
- Typ
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Konfiguration
- Usage
- Lokale Protokolldatei
- Änderungsprotokoll

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 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 AWS IoT Greengrass Protokolle überwachen.

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.

2.4.6

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.4.6 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.10.0 <2.17.0	Hart
Greengrass CLI	>=2.10.0 <2.17.0	Hart

2.4.5

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.4.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.10.0 <2.16.0	Hart
Greengrass CLI	>=2.10.0 <2.16.0	Hart

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	>=2.10.0 <2.15.0	Hart
Greengrass CLI	>=2.10.0 <2.15.0	Hart

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	>=2.10.0 <2.14.0	Hart
Greengrass CLI	>=2.10.0 <2.14.0	Hart

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
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

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
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 Usage.

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"
}
```

Usage

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.

So öffnen Sie die lokale Debug-Konsole

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 aus Sicherheitsgründen ein Kennwort erforderlich, da Sie mit dieser Komponente wichtige Informationen anzeigen und Vorgänge auf dem Kerngerät ausführen können. 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.

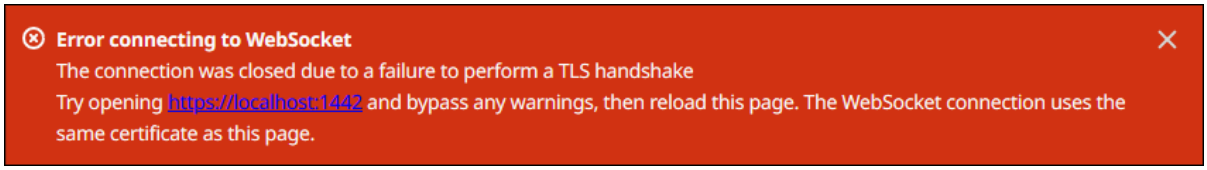
   > **Note**
   >
   > Wenn der Fingerabdruck des Zertifikats nicht übereinstimmt, gehen Sie Step 2 zu Eine neue Sitzung erstellen. Wenn der Fingerabdruck des Zertifikats immer noch nicht übereinstimmt, ist Ihre Verbindung möglicherweise unsicher.

   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 ändern1442. 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 aus Sicherheitsgründen ein Kennwort erforderlich, da Sie mit dieser Komponente wichtige Informationen anzeigen und Vorgänge auf dem Kerngerät ausführen können.

   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
2.4.6	Die Version wurde für die Version 2.16.0 von Greengrass Nucleus aktualisiert.
2.4.5	Die Version wurde für die Version 2.15.0 von Greengrass Nucleus aktualisiert.
2.4.4	Die Version wurde für die Version 2.14.0 von Greengrass Nucleus aktualisiert.
2.4.3	Die Version wurde für die Version 2.13.0 von Greengrass Nucleus aktualisiert. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, das fälschlicherweise `STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH` in Megabit pro Sekunde (Mbit/s) statt in Bytes pro Sekunde (Bps) angezeigt wurde.
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

> **Warning**
>
> Wir empfehlen ein Upgrade auf Log Manager v2.3.5 oder höher. Version 2.3.5 optimiert die Schreibvorgänge in der Log Manager-Konfiguration, reduziert I/O-Operationen und verbessert die Geschwindigkeit beim Hochladen von Protokollen, verbessert die Gesamtleistung des Geräts und verlängert möglicherweise die Lebensdauer des Geräts.

Die Log-Manager-Komponente (`aws.greengrass.LogManager`) lädt Protokolle von AWS IoT Greengrass Kerngeräten zu Amazon CloudWatch Logs hoch. Sie können Protokolle vom Greengrass-Nucleus, anderen Greengrass-Komponenten und anderen Anwendungen und Diensten hochladen, die keine Greengrass-Komponenten sind. Weitere Informationen zur Überwachung von

Protokollen in CloudWatch Logs und im lokalen Dateisystem finden Sie unter. AWS IoT Greengrass Protokolle überwachen

Die folgenden Überlegungen gelten, wenn Sie die Log-Manager-Komponente zum Schreiben in CloudWatch Logs verwenden:

- Verzögerungen protokollieren

  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 die Protokolldateien jede Stunde oder nachdem sie 1.024 KB groß sind. Daher lädt die Log-Manager-Komponente Protokolle erst hoch, nachdem die AWS IoT Greengrass Core-Software oder eine Greengrass-Komponente Protokolle im Wert von über 1.024 KB geschrieben hat. Sie können eine niedrigere Größenbeschränkung für Protokolldateien konfigurieren, damit Protokolldateien häufiger rotieren. Dies führt dazu, dass die Log-Manager-Komponente Logs häufiger in CloudWatch Logs hochlädt.

  Die Log Manager-Komponente Version 2.3.0 (und höher) verarbeitet und lädt alle Protokolle hoch. Wenn Sie ein neues Protokoll schreiben, verarbeitet Log Manager Version 2.3.0 (und höher) diese aktive Protokolldatei und lädt sie 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 Log-Manager-Komponente Logs häufiger in Logs hochlädt, CloudWatch indem Sie den konfigurieren. `periodicUploadIntervalSec` Weitere Informationen zur Konfiguration dieses periodischen 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 unterschiedliche Dateisysteme zum Schreiben von Protokollen verwenden, kehrt der Log Manager zum Verhalten der Protokollmanager-Komponentenversionen 2.2.8 und früher zurück. Informationen zum Zugriff auf Dateisystemprotokolle finden Sie unter Zugriff auf Dateisystemprotokolle.

- Zeitversatz

  Die Log-Manager-Komponente verwendet den standardmäßigen Signaturprozess von Signature Version 4, um API-Anfragen an CloudWatch Logs zu erstellen. Wenn die Systemzeit auf einem Kerngerät um mehr als 15 Minuten nicht mehr synchron ist, lehnt CloudWatch Logs die Anfragen ab. Weitere Informationen finden Sie unter Signaturprozess mit Signaturversion 4 in derAllgemeine AWS-Referenz .

Hinweise zu den Protokollgruppen und Protokollströmen, in die diese Komponente Protokolle hochlädt, finden Sie unter. Usage

Themen

- Versionen
- Typ
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Konfiguration
- Usage
- 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 AWS IoT Greengrass Protokolle überwachen.

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 `logs:DescribeLogStreams` Aktionen `logs:CreateLogGrouplogs:CreateLogStream`,`logs:PutLogEvents`, und zulassen, wie in der folgenden IAM-Beispielrichtlinie 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 verfügen`logs.region.amazonaws.com`, zu der der VPC-Endpunkt ist. `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 Endpunkten und Ports, die für den Basisbetrieb erforderlich sind. Weitere Informationen finden Sie unter Geräteverkehr über einen Proxy oder eine Firewall zulassen.

Endpoint	Port	Erforderlich	Description
`logs.`*`region`*`.amazonaws.com`	443	Nein	Erforderlich, wenn Sie Protokolle in Logs schreiben. CloudWatch

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.

2.3.11

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.3.11 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.1.0 <2.17.0	Weich

2.3.10

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.3.10 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.1.0 <2.16.0	Weich

2.3.9

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.3.9 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.1.0 <2.15.0	Weich

2.3.8

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.3.8 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.1.0 <2.14.0	Weich

2.3.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.3.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=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	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	>=2.0.3 <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.

v2.3.10

`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 Greengrass Core-Softwaresystemprotokolle, die Protokolle der Greengrass-Nukleus - 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 Logs hochladen.

Standard: `false`

`minimumLogLevel`

(Optional) Die Mindestanzahl an Protokollnachrichten, die hochgeladen werden sollen. Diese Mindeststufe gilt nur, wenn Sie die Greengrass Nucleus-Komponente für die Ausgabe von Protokollen im JSON-Format konfigurieren. Um Logs im JSON-Format zu aktivieren, geben Sie `JSON` für den Logging-Formatparameter (`logging.format`) an.

Wählen Sie aus den folgenden Protokollebenen, die hier in der Reihenfolge der Stufen aufgeführt sind:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Standard: `INFO`

`diskSpaceLimit`

(Optional) Die maximale Gesamtgröße der Greengrass-Systemprotokolldateien in der von Ihnen angegebenen Einheit. `diskSpaceLimitUnit` Wenn 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 Log-Manager-Komponente die Protokolle in Logs hochgeladen hat. CloudWatch

Standard: `false`

`componentLogsConfigurationMap`

(Optional) Eine Übersicht der Protokollkonfigurationen für Komponenten auf dem Kerngerät. Jedes `componentName` Objekt in dieser Zuordnung definiert die Protokollkonfiguration für die Komponente oder Anwendung. Die Log-Manager-Komponente lädt diese Komponentenprotokolle in Logs hoch. CloudWatch

> **Important**
>
> Wir empfehlen dringend, einen einzigen 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`. Die Nichtbeachtung dieser Empfehlung kann dazu führen, dass doppelte Protokolle hochgeladen werden CloudWatch. Wenn Sie auf mehrere aktive Protokolldateien mit einem einzigen regulären Ausdruck abzielen, empfehlen wir Ihnen, ein Upgrade auf Log Manager v2.3.1 oder höher durchzuführen und eine Änderung Ihrer Konfiguration anhand der Beispielkonfiguration in Betracht zu ziehen.

> **Note**
>
> Wenn Sie ein Upgrade von einer früheren Version von Log Manager als Version 2.2.0 durchführen, können Sie stattdessen weiterhin die Liste verwenden. `componentLogsConfiguration` `componentLogsConfigurationMap` Es wird jedoch dringend empfohlen, das Zuordnungsformat zu verwenden, sodass Sie Aktualisierungen zum Zusammenführen und Zurücksetzen verwenden können, um Konfigurationen für bestimmte Komponenten zu ändern. Informationen zu diesem `componentLogsConfiguration` Parameter finden Sie in den Konfigurationsparametern für Version 2.1.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 Mindestanzahl der hochzuladenden Protokollnachrichten. Diese Mindeststufe gilt nur, wenn die Protokolle dieser Komponente ein bestimmtes JSON-Format verwenden, das Sie im Repository des AWS IoT Greengrass Protokollierungsmoduls finden GitHub.

Wählen Sie aus den folgenden Protokollebenen, die hier in der Reihenfolge der Stufen aufgeführt sind:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Standard: `INFO`

`diskSpaceLimit`

(Optional) Die maximale Gesamtgröße aller Protokolldateien für diese Komponente in der von Ihnen angegebenen Einheit`diskSpaceLimitUnit`. Wenn 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 für die Loggrößenbeschränkung (`totalLogsSizeKB`) der Greengrass-Kernkomponente. Die AWS IoT Greengrass Core-Software verwendet den kleinsten 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 auf Standardausgabe (stdout) und Standardfehler (stderr) drucken.

Standard: *`/greengrass/v2`*`/logs`.

`logFileRegex`

(Optional) Ein regulärer Ausdruck, der das Namensformat der Protokolldatei 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 auf Standardausgabe (stdout) und Standardfehler (stderr) drucken.

Wenn Ihre Komponente oder Anwendung Protokolldateien rotiert, geben Sie einen regulären Ausdruck an, der den Namen der rotierten Protokolldateien entspricht. Sie könnten beispielsweise angeben**`hello_world\\\\w*.log`**, dass Protokolle für eine Hello World-Anwendung hochgeladen werden sollen. Das `\\\\w*` Muster entspricht null oder mehr Wortzeichen, einschließlich alphanumerischer Zeichen und Unterstriche. Dieser reguläre Ausdruck entspricht Protokolldateien mit und ohne Zeitstempel im Namen. 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 Log-Manager-Komponente die Protokolle in Logs hochgeladen hat. CloudWatch

Standard: `false`

`multiLineStartPattern`

(Optional) Ein regulärer Ausdruck, der angibt, wann es sich bei einer Protokollnachricht in einer neuen Zeile um eine neue Protokollnachricht handelt. Wenn der reguläre Ausdruck nicht mit der neuen Zeile übereinstimmt, hängt die Log-Manager-Komponente 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. einem Tabulator 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 Log-Manager-Komponente Nachrichten, die sich über mehrere Zeilen erstrecken, nicht aufteilt, wie z. B. Stack-Traces.

`periodicUploadIntervalSec`

(Optional) Der Zeitraum in Sekunden, in dem die Log Manager-Komponente nach neuen Protokolldateien sucht, die hochgeladen werden können.

Standard: `300` (5 Minuten)

Minimum: `0.000001` (1 Mikrosekunde)

`updateToTlogIntervalSec`

(Optional) Der Zeitraum in Sekunden, in dem der Nucleus Details zu Amazon CloudWatch Events Log-Upload-Ereignissen in das lokale Transaktionslog () schreibt. `config.tlog` Standardmäßig wird der in angegebene Wert verwendet. `periodicUploadIntervalSec` Sie können diesen Parameter ändern, um das Schreibintervall zu verlängern.

Standard: `periodicUploadIntervalSec`

Minimum: `periodicUploadIntervalSec`

`deprecatedVersionSupport`

Gibt an, ob der Protokollmanager die in Log Manager v2.3.5 eingeführten Verbesserungen der Protokollierungsgeschwindigkeit verwenden sollte. Stellen Sie den Wert auf ein, `false` um die Verbesserungen zu nutzen.

Wenn Sie diesen Wert `false` bei der Aktualisierung von Log Manager v2.3.1 oder früher auf festlegen, werden möglicherweise doppelte Protokolleinträge hochgeladen.

Der Standardwert ist `true`.

Example Beispiel: Update zur Zusammenführung von Konfigurationen

Die folgende Beispielkonfiguration legt fest, dass System- und `com.example.HelloWorld` Komponentenprotokolle in Logs hochgeladen werden CloudWatch sollen.

```
{
  "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 als Ziel verwenden möchten. Diese Beispielkonfiguration gibt an, in welche aktiven Protokolldateien Sie hochladen möchten CloudWatch. Bei Verwendung dieser Beispielkonfiguration werden auch alle rotierten Dateien hochgeladen, die dem entsprechen`logFileRegex`. Diese Beispielkonfiguration wird von 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"
}
```

v2.3.6 – v2.3.9

`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 Greengrass Core-Softwaresystemprotokolle, die Protokolle der Greengrass-Nukleus - 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 Logs hochladen.

Standard: `false`

`minimumLogLevel`

(Optional) Die Mindestanzahl an Protokollnachrichten, die hochgeladen werden sollen. Diese Mindeststufe gilt nur, wenn Sie die Greengrass Nucleus-Komponente für die Ausgabe von Protokollen im JSON-Format konfigurieren. Um Logs im JSON-Format zu aktivieren, geben Sie `JSON` für den Logging-Formatparameter (`logging.format`) an.

Wählen Sie aus den folgenden Protokollebenen, die hier in der Reihenfolge der Stufen aufgeführt sind:

- `DEBUG`

- `INFO`
- `WARN`
- `ERROR`

Standard: `INFO`

`diskSpaceLimit`

(Optional) Die maximale Gesamtgröße der Greengrass-Systemprotokolldateien in der von Ihnen angegebenen Einheit. `diskSpaceLimitUnit` Wenn 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 Log-Manager-Komponente die Protokolle in Logs hochgeladen hat. CloudWatch

Standard: `false`

`componentLogsConfigurationMap`

(Optional) Eine Übersicht der Protokollkonfigurationen für Komponenten auf dem Kerngerät. Jedes `componentName` Objekt in dieser Zuordnung definiert die Protokollkonfiguration für die Komponente oder Anwendung. Die Log-Manager-Komponente lädt diese Komponentenprotokolle in Logs hoch. CloudWatch

> **Important**
>
> Wir empfehlen dringend, einen einzigen 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`. Die Nichtbeachtung dieser Empfehlung kann dazu führen, dass doppelte Protokolle hochgeladen werden CloudWatch. Wenn Sie auf mehrere aktive Protokolldateien mit einem einzigen regulären Ausdruck abzielen, empfehlen wir Ihnen, ein Upgrade auf Log Manager v2.3.1 oder höher durchzuführen und eine Änderung Ihrer Konfiguration anhand der Beispielkonfiguration in Betracht zu ziehen.

> **Note**
>
> Wenn Sie ein Upgrade von einer früheren Version von Log Manager als Version 2.2.0 durchführen, können Sie stattdessen weiterhin die Liste verwenden. `componentLogsConfiguration` `componentLogsConfigurationMap` Es wird jedoch dringend empfohlen, das Zuordnungsformat zu verwenden, sodass Sie Aktualisierungen zum Zusammenführen und Zurücksetzen verwenden können, um Konfigurationen für bestimmte Komponenten zu ändern. Informationen zu diesem `componentLogsConfiguration` Parameter finden Sie in den Konfigurationsparametern für Version 2.1.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 Mindestanzahl der hochzuladenden Protokollnachrichten. Diese Mindeststufe gilt nur, wenn die Protokolle dieser Komponente ein bestimmtes JSON-Format verwenden, das Sie im Repository des AWS IoT Greengrass Protokollierungsmoduls finden GitHub.

Wählen Sie aus den folgenden Protokollebenen, die hier in der Reihenfolge der Stufen aufgeführt sind:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Standard: `INFO`

`diskSpaceLimit`

(Optional) Die maximale Gesamtgröße aller Protokolldateien für diese Komponente in der von Ihnen angegebenen Einheit`diskSpaceLimitUnit`. Wenn 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 für die Loggrößenbeschränkung (`totalLogsSizeKB`) der Greengrass-Kernkomponente. Die AWS IoT Greengrass Core-Software verwendet den kleinsten 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 auf Standardausgabe (stdout) und Standardfehler (stderr) drucken.

Standard: */greengrass/v2*/logs.

`logFileRegex`

(Optional) Ein regulärer Ausdruck, der das Namensformat der Protokolldatei 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 auf Standardausgabe (stdout) und Standardfehler (stderr) drucken.

Wenn Ihre Komponente oder Anwendung Protokolldateien rotiert, geben Sie einen regulären Ausdruck an, der den Namen der rotierten Protokolldateien entspricht. Sie könnten beispielsweise angeben**`hello_world\\\\w*.log`**, dass Protokolle für eine Hello World-Anwendung hochgeladen werden sollen. Das `\\\\w*` Muster entspricht null oder mehr Wortzeichen, einschließlich alphanumerischer Zeichen und Unterstriche. Dieser reguläre Ausdruck entspricht Protokolldateien mit und ohne Zeitstempel im Namen. 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 Log-Manager-Komponente die Protokolle in Logs hochgeladen hat. CloudWatch

Standard: `false`

`multiLineStartPattern`

(Optional) Ein regulärer Ausdruck, der angibt, wann es sich bei einer Protokollnachricht in einer neuen Zeile um eine neue Protokollnachricht handelt. Wenn der reguläre Ausdruck nicht mit der neuen Zeile übereinstimmt, hängt die Log-Manager-Komponente 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. einem Tabulator 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 Log-Manager-Komponente Nachrichten, die sich über mehrere Zeilen erstrecken, nicht aufteilt, wie z. B. Stack-Traces.

`periodicUploadIntervalSec`

(Optional) Der Zeitraum in Sekunden, in dem die Log Manager-Komponente nach neuen Protokolldateien sucht, die hochgeladen werden können.

Standard: `300` (5 Minuten)

Minimum: `0.000001` (1 Mikrosekunde)

`deprecatedVersionSupport`

Gibt an, ob der Protokollmanager die in Log Manager v2.3.5 eingeführten Verbesserungen der Protokollierungsgeschwindigkeit verwenden sollte. Stellen Sie den Wert auf ein, `false` um die Verbesserungen zu nutzen.

Wenn Sie diesen Wert `false` bei der Aktualisierung von Log Manager v2.3.1 oder früher auf festlegen, werden möglicherweise doppelte Protokolleinträge hochgeladen.

Der Standardwert ist `true`.

Example Beispiel: Update zur Zusammenführung von Konfigurationen

Die folgende Beispielkonfiguration legt fest, dass System- und `com.example.HelloWorld` Komponentenprotokolle in Logs hochgeladen werden CloudWatch sollen.

```
{
  "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 als Ziel verwenden möchten. Diese Beispielkonfiguration gibt an, in welche aktiven Protokolldateien Sie hochladen möchten CloudWatch. Bei Verwendung dieser Beispielkonfiguration werden auch alle rotierten Dateien hochgeladen, die dem entsprechen`logFileRegex`. Diese Beispielkonfiguration wird von 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"
}
```

v2.3.0 – 2.3.5

`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 Greengrass Core-Softwaresystemprotokolle, die Protokolle der Greengrass-Nukleus - 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 Logs hochladen.

Standard: `false`

`minimumLogLevel`

(Optional) Die Mindestanzahl an Protokollnachrichten, die hochgeladen werden sollen. Diese Mindeststufe gilt nur, wenn Sie die Greengrass Nucleus-Komponente für die Ausgabe von Protokollen im JSON-Format konfigurieren. Um Logs im JSON-Format zu aktivieren, geben Sie `JSON` für den Logging-Formatparameter (`logging.format`) an.

Wählen Sie aus den folgenden Protokollebenen, die hier in der Reihenfolge der Stufen aufgeführt sind:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Standard: `INFO`

`diskSpaceLimit`

(Optional) Die maximale Gesamtgröße der Greengrass-Systemprotokolldateien in der von Ihnen angegebenen Einheit. `diskSpaceLimitUnit` Wenn 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 Log-Manager-Komponente die Protokolle in Logs hochgeladen hat. CloudWatch

Standard: `false`

`componentLogsConfigurationMap`

(Optional) Eine Übersicht der Protokollkonfigurationen für Komponenten auf dem Kerngerät. Jedes `componentName` Objekt in dieser Zuordnung definiert die Protokollkonfiguration für die Komponente oder Anwendung. Die Log-Manager-Komponente lädt diese Komponentenprotokolle in Logs hoch. CloudWatch

> **Important**
>
> Wir empfehlen dringend, einen einzigen 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`. Die Nichtbeachtung dieser Empfehlung kann dazu führen, dass doppelte Protokolle hochgeladen werden CloudWatch. Wenn Sie auf mehrere aktive Protokolldateien mit einem einzigen regulären Ausdruck abzielen, empfehlen wir Ihnen, auf Log Manager v2.3.1 zu aktualisieren und eine Änderung Ihrer Konfiguration anhand der Beispielkonfiguration in Betracht zu ziehen.

> **Note**
>
> Wenn Sie ein Upgrade von einer früheren Version von Log Manager als Version 2.2.0 durchführen, können Sie stattdessen weiterhin die Liste verwenden. `componentLogsConfiguration` `componentLogsConfigurationMap` Es wird jedoch dringend empfohlen, das Zuordnungsformat zu verwenden, sodass Sie Aktualisierungen zum Zusammenführen und Zurücksetzen verwenden können, um Konfigurationen für bestimmte Komponenten zu ändern. Informationen zu diesem `componentLogsConfiguration` Parameter finden Sie in den Konfigurationsparametern für Version 2.1.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 Mindestanzahl der hochzuladenden Protokollnachrichten. Diese Mindeststufe gilt nur, wenn die Protokolle dieser Komponente ein bestimmtes JSON-Format verwenden, das Sie im Repository des AWS IoT Greengrass Protokollierungsmoduls finden GitHub.

Wählen Sie aus den folgenden Protokollebenen, die hier in der Reihenfolge der Stufen aufgeführt sind:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Standard: `INFO`

`diskSpaceLimit`

(Optional) Die maximale Gesamtgröße aller Protokolldateien für diese Komponente in der von Ihnen angegebenen Einheit`diskSpaceLimitUnit`. Wenn 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 für die Loggrößenbeschränkung (`totalLogsSizeKB`) der Greengrass-Kernkomponente. Die AWS IoT Greengrass Core-Software verwendet den kleinsten 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 auf Standardausgabe (stdout) und Standardfehler (stderr) drucken.

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 auf Standardausgabe (stdout) und Standardfehler (stderr) drucken.

Wenn Ihre Komponente oder Anwendung Protokolldateien rotiert, geben Sie einen regulären Ausdruck an, der den Namen der rotierten Protokolldateien entspricht. Sie könnten beispielsweise angeben**`hello_world\\\\w*.log`**, dass Protokolle

für eine Hello World-Anwendung hochgeladen werden sollen. Das `\\\\w*` Muster entspricht null oder mehr Wortzeichen, einschließlich alphanumerischer Zeichen und Unterstriche. Dieser reguläre Ausdruck entspricht Protokolldateien mit und ohne Zeitstempel im Namen. 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 Log-Manager-Komponente die Protokolle in Logs hochgeladen hat. CloudWatch

Standard: `false`

`multiLineStartPattern`

(Optional) Ein regulärer Ausdruck, der angibt, wann es sich bei einer Protokollnachricht in einer neuen Zeile um eine neue Protokollnachricht handelt. Wenn der reguläre Ausdruck nicht mit der neuen Zeile übereinstimmt, hängt die Log-Manager-Komponente 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. einem Tabulator 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 Log-Manager-Komponente Nachrichten, die sich über mehrere Zeilen erstrecken, nicht aufteilt, wie z. B. Stack-Traces.

`periodicUploadIntervalSec`

(Optional) Der Zeitraum in Sekunden, in dem die Log Manager-Komponente nach neuen Protokolldateien sucht, die hochgeladen werden können.

Standard: `300` (5 Minuten)

Minimum: `0.000001` (1 Mikrosekunde)

Example Beispiel: Aktualisierung der Zusammenführung von Konfigurationen

Die folgende Beispielkonfiguration legt fest, dass System- und `com.example.HelloWorld` Komponentenprotokolle in Logs hochgeladen werden CloudWatch sollen.

```
{
  "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 als Ziel verwenden möchten. Diese Beispielkonfiguration gibt an, in welche aktiven Protokolldateien Sie hochladen möchten CloudWatch. Bei Verwendung dieser Beispielkonfiguration werden auch alle rotierten Dateien hochgeladen, die dem entsprechen`logFileRegex`. Diese Beispielkonfiguration wird von 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 Greengrass Core-Softwaresystemprotokolle, die Protokolle der Greengrass-Nukleus - 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 Logs hochladen.

Standard: `false`

`minimumLogLevel`

(Optional) Die Mindestanzahl an Protokollnachrichten, die hochgeladen werden sollen. Diese Mindeststufe gilt nur, wenn Sie die Greengrass Nucleus-Komponente für die Ausgabe von Protokollen im JSON-Format konfigurieren. Um Logs im JSON-Format zu aktivieren, geben Sie `JSON` für den Logging-Formatparameter (`logging.format`) an.

Wählen Sie aus den folgenden Protokollebenen, die hier in der Reihenfolge der Stufen aufgeführt sind:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Standard: `INFO`

`diskSpaceLimit`

(Optional) Die maximale Gesamtgröße der Greengrass-Systemprotokolldateien in der von Ihnen angegebenen Einheit. `diskSpaceLimitUnit` Wenn 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 Log-Manager-Komponente die Protokolle in Logs hochgeladen hat. CloudWatch

Standard: `false`

`componentLogsConfigurationMap`

(Optional) Eine Übersicht der Protokollkonfigurationen für Komponenten auf dem Kerngerät. Jedes `componentName` Objekt in dieser Zuordnung definiert die Protokollkonfiguration für die Komponente oder Anwendung. Die Log-Manager-Komponente lädt diese Komponentenprotokolle in Logs hoch. CloudWatch

> **Note**
>
> Wenn Sie ein Upgrade von einer früheren Version von Log Manager als Version 2.2.0 durchführen, können Sie stattdessen

weiterhin die `componentLogsConfiguration` Liste verwenden. `componentLogsConfigurationMap` Es wird jedoch dringend empfohlen, das Zuordnungsformat zu verwenden, sodass Sie Aktualisierungen zum Zusammenführen und Zurücksetzen verwenden können, um Konfigurationen für bestimmte Komponenten zu ändern. Informationen zu diesem `componentLogsConfiguration` Parameter finden Sie in den Konfigurationsparametern für Version 2.1.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 Mindestanzahl der hochzuladenden Protokollnachrichten. Diese Mindeststufe gilt nur, wenn die Protokolle dieser Komponente ein bestimmtes JSON-Format verwenden, das Sie im Repository des AWS IoT Greengrass Protokollierungsmoduls finden GitHub.

Wählen Sie aus den folgenden Protokollebenen, die hier in der Reihenfolge der Stufen aufgeführt sind:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Standard: `INFO`

`diskSpaceLimit`

(Optional) Die maximale Gesamtgröße aller Protokolldateien für diese Komponente in der von Ihnen angegebenen Einheit`diskSpaceLimitUnit`. Wenn 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 für die Loggrößenbeschränkung (`totalLogsSizeKB`) der Greengrass-Kernkomponente. Die AWS IoT Greengrass Core-Software verwendet den kleinsten 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 auf Standardausgabe (stdout) und Standardfehler (stderr) drucken.

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 auf Standardausgabe (stdout) und Standardfehler (stderr) drucken.

Wenn Ihre Komponente oder Anwendung Protokolldateien rotiert, geben Sie einen regulären Ausdruck an, der den Namen der rotierten Protokolldateien entspricht. Sie könnten beispielsweise angeben**`hello_world\\\\w*.log`**, dass Protokolle für eine Hello World-Anwendung hochgeladen werden sollen. Das `\\\\w*` Muster

entspricht null oder mehr Wortzeichen, einschließlich alphanumerischer Zeichen und Unterstriche. Dieser reguläre Ausdruck entspricht Protokolldateien mit und ohne Zeitstempel im Namen. 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 Log-Manager-Komponente die Protokolle in Logs hochgeladen hat. CloudWatch

Standard: `false`

`multiLineStartPattern`

(Optional) Ein regulärer Ausdruck, der angibt, wann es sich bei einer Protokollnachricht in einer neuen Zeile um eine neue Protokollnachricht handelt. Wenn der reguläre Ausdruck nicht mit der neuen Zeile übereinstimmt, hängt die Log-Manager-Komponente 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. einem Tabulator 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 Log-Manager-Komponente Nachrichten, die sich über mehrere Zeilen erstrecken, nicht aufteilt, wie z. B. Stack-Traces.

`periodicUploadIntervalSec`

(Optional) Der Zeitraum in Sekunden, in dem die Log Manager-Komponente nach neuen Protokolldateien sucht, die hochgeladen werden können.

Standard: `300` (5 Minuten)

Minimum: `0.000001` (1 Mikrosekunde)

Example Beispiel: Aktualisierung der Zusammenführung von Konfigurationen

Die folgende Beispielkonfiguration legt fest, dass System- und `com.example.HelloWorld` Komponentenprotokolle in Logs hochgeladen werden CloudWatch sollen.

```
{
  "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 Greengrass Core-Softwaresystemprotokolle, die Protokolle der Greengrass-Nukleus - 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 Logs hochladen.

Standard: `false`

`minimumLogLevel`

(Optional) Die Mindestanzahl an Protokollnachrichten, die hochgeladen werden sollen. Diese Mindeststufe gilt nur, wenn Sie die Greengrass Nucleus-Komponente für die Ausgabe von Protokollen im JSON-Format konfigurieren. Um Logs im JSON-Format zu aktivieren, geben Sie `JSON` für den Logging-Formatparameter (`logging.format`) an.

Wählen Sie aus den folgenden Protokollebenen, die hier in der Reihenfolge der Stufen aufgeführt sind:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Standard: `INFO`

`diskSpaceLimit`

(Optional) Die maximale Gesamtgröße der Greengrass-Systemprotokolldateien in der von Ihnen angegebenen Einheit. `diskSpaceLimitUnit` Wenn 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 Log-Manager-Komponente die Protokolle in Logs hochgeladen hat. CloudWatch

Standard: `false`

`componentLogsConfiguration`

(Optional) Eine Liste der Protokollkonfigurationen für Komponenten auf dem Kerngerät. Jede Konfiguration in dieser Liste definiert die Protokollkonfiguration für eine Komponente oder Anwendung. Die Log-Manager-Komponente lädt diese Komponentenprotokolle in Logs hoch CloudWatch

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 Mindestanzahl der hochzuladenden Protokollnachrichten. Diese Mindeststufe gilt nur, wenn die Protokolle dieser Komponente ein bestimmtes JSON-Format verwenden, das Sie im Repository des AWS IoT Greengrass Protokollierungsmoduls finden GitHub.

Wählen Sie aus den folgenden Protokollebenen, die hier in der Reihenfolge der Stufen aufgeführt sind:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Standard: `INFO`

`diskSpaceLimit`

(Optional) Die maximale Gesamtgröße aller Protokolldateien für diese Komponente in der von Ihnen angegebenen Einheit`diskSpaceLimitUnit`. Wenn 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 für die Loggrößenbeschränkung (`totalLogsSizeKB`) der Greengrass-Kernkomponente. Die AWS IoT Greengrass Core-Software verwendet den kleinsten 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 auf Standardausgabe (stdout) und Standardfehler (stderr) drucken.

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 auf Standardausgabe (stdout) und Standardfehler (stderr) drucken.

Wenn Ihre Komponente oder Anwendung Protokolldateien rotiert, geben Sie einen regulären Ausdruck an, der den Namen der rotierten Protokolldateien entspricht. Sie könnten beispielsweise angeben**`hello_world\\\\w*.log`**, dass Protokolle für eine

Hello World-Anwendung hochgeladen werden sollen. Das `\\\\w*` Muster entspricht null oder mehr Wortzeichen, einschließlich alphanumerischer Zeichen und Unterstriche. Dieser reguläre Ausdruck entspricht Protokolldateien mit und ohne Zeitstempel im Namen. 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 Log-Manager-Komponente die Protokolle in Logs hochgeladen hat. CloudWatch

Standard: `false`

`multiLineStartPattern`

(Optional) Ein regulärer Ausdruck, der angibt, wann es sich bei einer Protokollnachricht in einer neuen Zeile um eine neue Protokollnachricht handelt. Wenn der reguläre Ausdruck nicht mit der neuen Zeile übereinstimmt, hängt die Log-Manager-Komponente 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. einem Tabulator 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 Log-Manager-Komponente Nachrichten, die sich über mehrere Zeilen erstrecken, nicht aufteilt, wie z. B. Stack-Traces.

`periodicUploadIntervalSec`

(Optional) Der Zeitraum in Sekunden, in dem die Log Manager-Komponente nach neuen Protokolldateien sucht, die hochgeladen werden können.

Standard: `300` (5 Minuten)

Minimum: `0.000001` (1 Mikrosekunde)

Example Beispiel: Aktualisierung der Zusammenführung von Konfigurationen

Die folgende Beispielkonfiguration legt fest, dass System- und `com.example.HelloWorld` Komponentenprotokolle in Logs hochgeladen werden CloudWatch sollen.

```
{
  "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 Greengrass Core-Software-Systemprotokolle. 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 Logs hochladen.

Standard: `false`

`minimumLogLevel`

(Optional) Die Mindestanzahl an Protokollnachrichten, die hochgeladen werden sollen. Diese Mindeststufe gilt nur, wenn Sie die Greengrass Nucleus-Komponente für die Ausgabe von Protokollen im JSON-Format konfigurieren. Um Logs im JSON-Format zu aktivieren, geben Sie `JSON` für den Logging-Formatparameter (`logging.format`) an.

Wählen Sie aus den folgenden Protokollebenen, die hier in der Reihenfolge der Stufen aufgeführt sind:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Standard: `INFO`

`diskSpaceLimit`

(Optional) Die maximale Gesamtgröße der Greengrass-Systemprotokolldateien in der von Ihnen angegebenen Einheit. `diskSpaceLimitUnit` Wenn 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 Log-Manager-Komponente die Protokolle in Logs hochgeladen hat. CloudWatch

Standard: `false`

`componentLogsConfiguration`

(Optional) Eine Liste der Protokollkonfigurationen für Komponenten auf dem Kerngerät. Jede Konfiguration in dieser Liste definiert die Protokollkonfiguration für eine Komponente oder Anwendung. Die Log-Manager-Komponente lädt diese Komponentenprotokolle in Logs hoch CloudWatch

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 Mindestanzahl der hochzuladenden Protokollnachrichten. Diese Mindeststufe gilt nur, wenn die Protokolle dieser Komponente ein bestimmtes JSON-Format verwenden, das Sie im Repository des AWS IoT Greengrass Protokollierungsmoduls finden GitHub.

Wählen Sie aus den folgenden Protokollebenen, die hier in der Reihenfolge der Stufen aufgeführt sind:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Standard: `INFO`

`diskSpaceLimit`

(Optional) Die maximale Gesamtgröße aller Protokolldateien für diese Komponente in der von Ihnen angegebenen Einheit`diskSpaceLimitUnit`. Wenn 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 für die Loggrößenbeschränkung (`totalLogsSizeKB`) der Greengrass-Kernkomponente. Die AWS IoT Greengrass Core-Software verwendet den kleinsten 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**`/greengrass/v2/logs`**, geben Sie Ihren Greengrass-Stammordner an und `/greengrass/v2` ersetzen Sie ihn durch diesen.

`logFileRegex`

Ein regulärer Ausdruck, der das Namensformat der Protokolldatei 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 Logs einer Greengrass-Komponente hochzuladen, geben Sie eine Regex an, die den rotierten Log-Dateinamen entspricht. Sie könnten beispielsweise angeben**`com.example.HelloWorld\\w*.log`**, dass Protokolle für eine Hello World-Komponente hochgeladen werden sollen. Das `\\w*` Muster entspricht null oder mehr Wortzeichen, einschließlich alphanumerischer Zeichen und Unterstriche. Dieser reguläre Ausdruck entspricht Protokolldateien mit und ohne Zeitstempel im Namen. 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-Kern fügt den Protokolldateien einen rotierenden Zeitstempel hinzu.

`deleteLogFileAfterCloudUpload`

(Optional) Sie können eine Protokolldatei löschen, nachdem die Log-Manager-Komponente die Protokolle in Logs hochgeladen hat. CloudWatch

Standard: `false`

`multiLineStartPattern`

(Optional) Ein regulärer Ausdruck, der angibt, wann es sich bei einer Protokollnachricht in einer neuen Zeile um eine neue Protokollnachricht handelt. Wenn der reguläre Ausdruck nicht mit der neuen Zeile übereinstimmt, hängt die Log-Manager-Komponente 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. einem Tabulator 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 Log-Manager-Komponente Nachrichten, die sich über mehrere Zeilen erstrecken, nicht aufteilt, wie z. B. Stack-Traces.

`periodicUploadIntervalSec`

(Optional) Der Zeitraum in Sekunden, in dem die Log Manager-Komponente nach neuen Protokolldateien sucht, die hochgeladen werden können.

Standard: `300` (5 Minuten)

Minimum: `0.000001` (1 Mikrosekunde)

Example Beispiel: Aktualisierung der Zusammenführung von Konfigurationen

Die folgende Beispielkonfiguration legt fest, dass System- und `com.example.HelloWorld` Komponentenprotokolle in Logs hochgeladen werden CloudWatch sollen.

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfiguration": [
      {
        "componentName": "com.example.HelloWorld",
        "minimumLogLevel": "INFO",
        "logFileDirectoryPath": "/greengrass/v2/logs",
        "logFileRegex": "com.example.HelloWorld\\w*.log",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    ]
  },
  "periodicUploadIntervalSec": "300"
}
```

Usage

Die Log-Manager-Komponente lädt Dateien in die folgenden Protokollgruppen und Protokolldatenströme hoch.

2.1.0 and later

Name der Protokollgruppe

```
/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 umfasst Protokolle für die Nucleus- und Plugin-Komponenten, die in derselben JVM wie der Greengrass-Nucleus ausgeführt werden. Die Komponente ist Teil des Greengrass-Kerns.

- `UserComponent`— Diese Protokollgruppe umfasst 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 Kerngerät verwendet.
- `componentName`— Der Name der Komponente. Für Systemprotokolle ist dieser Wert`System`.

Name des Protokollstreams

```
/date/thing/thingName
```

Der Name des Protokolldatenstroms verwendet die folgenden Variablen:

- `date`— Das Datum des Protokolls, z. `2020/12/15` B. Die Log-Manager-Komponente verwendet das `yyyy/MM/dd` Format.
- `thingName`— Der Name des Kerngeräts.

> **Note**
>
> Wenn ein Dingname einen Doppelpunkt (`:`) enthält, ersetzt der Logmanager den Doppelpunkt durch ein Plus (+).

2.0.x

Name der Protokollgruppe

```
/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 Logmanager verwendet diesen Typ für Greengrass-Komponenten und andere Anwendungen auf dem Gerät.
- `region`— Die AWS Region, die das Kerngerät verwendet.

- `componentName`— Der Name der Komponente. Für Systemprotokolle ist dieser Wert`System`.

Name des Protokollstreams

```
/date/deploymentTargets/thingName
```

Der Name des Protokolldatenstroms verwendet die folgenden Variablen:

- `date`— Das Datum des Protokolls, z. `2020/12/15` B. Die Log-Manager-Komponente verwendet das `yyyy/MM/dd` Format.
- `deploymentTargets`— Die Dinge, zu deren Deployments die Komponente gehört. Die Log-Manager-Komponente trennt jedes Ziel durch einen Schrägstrich. Wenn die Komponente aufgrund einer lokalen Bereitstellung auf dem Kerngerät ausgeführt wird, ist `LOCAL_DEPLOYMENT` dieser Wert.

  Stellen Sie sich ein Beispiel vor`MyGreengrassCore`, bei dem Sie ein Kerngerät mit dem Namen haben und das Kerngerät über zwei Bereitstellungen verfügt:

  - Eine Bereitstellung, die auf das Kerngerät abzielt,`MyGreengrassCore`.
  - Eine Bereitstellung, die auf eine Dinggruppe mit dem Namen abzielt`MyGreengrassCoreGroup`, die das Kerngerät enthält.

  Die `deploymentTargets` für dieses Kerngerät sind`thing/MyGreengrassCore/thinggroup/MyGreengrassCoreGroup`.
- `thingName`— Der Name des Kerngeräts.

Formate für Protokolleinträge.

Der Greengrass-Kern schreibt Protokolldateien entweder im String- oder JSON-Format. Bei Systemprotokollen steuern Sie das Format, indem Sie das `format` Feld des `logging` Eintrags festlegen. Sie finden den `logging` Eintrag in der Konfigurationsdatei der Greengrass Nucleus-Komponente. Weitere Informationen finden Sie unter Greengrass Nucleus-Konfiguration.

Das Textformat ist formfrei und akzeptiert jede Zeichenfolge. Die folgende Servicemeldung zum Flottenstatus ist ein Beispiel für die Protokollierung im Zeichenkettenformat:

```
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-Befehl logs anzeigen oder programmgesteuert mit Protokollen interagieren möchten. Das folgende Beispiel skizziert die JSON-Form:

```
{
   "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 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
2.3.11	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem die Laufzeitkonfiguration von Log Manager mit veralteten Informationen zu hochgeladenen Protokolldateien unbegrenzt wuchs.
2.3.10	Neue Features • Fügt einen neuen Konfigurationsschlüssel (`updateToTlogIntervalSec`) hinzu, um die Häufigkeit zu steuern, mit der Details von Log-Upload-Ereignissen im lokalen Transaktionslog () gespeichert werden. `config.tlog` Fehlerkorrekturen und Verbesserungen • Verbessert den Protokollmanager, um den Cloudwatch-Client bei Socket-Verbindungsfehlern zu aktualisieren.
2.3.9	Die Version wurde für die Version 2.14.0 von Greengrass Nucleus aktualisiert.
2.3.8	Die Version wurde für die Version 2.13.0 von Greengrass Nucleus aktualisiert.
2.3.7	Die Version wurde für die Version 2.12.0 von Greengrass Nucleus aktualisiert.

Version	Änderungen
2.3.6	Fehlerkorrekturen und Verbesserungen • Passt die Protokollebenen für bestimmte Fehler an.
2.3.5	Verbesserungen Verbessert die Geschwindigkeit beim Hochladen von Protokollen. Die Version wurde für die Version 2.11.0 von Greengrass Nucleus aktualisiert.
2.3.4	Fehlerkorrekturen und Verbesserungen • Fügt Unterstützung für das Setzen des `periodicUploadIntervalSec` Parameters auf Bruchwerte hinzu. Das Minimum ist 1 Mikrosekunde. • Behebt ein Problem, bei dem der Protokollmanager die CloudWatch `putLogEvents` Grenzwerte nicht einhält.
2.3.3	Die Version wurde für die Version 2.10.0 von Greengrass Nucleus aktualisiert.
2.3.2	Fehlerkorrekturen und Verbesserungen • Verbessert die Speicherverwaltung, sodass Protokolldateien nicht gelöscht werden, bevor sie hochgeladen werden. • Behebt Probleme mit der Cache-Verwaltung. • Zusätzliche kleinere Bugfixes und Verbesserungen.
2.3.1	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem Zieldateigruppen mit mehreren aktiven Protokolldateien doppelte Einträge hochladen. CloudWatch • Zusätzliche kleinere Bugfixes und Verbesserungen.

Version	Änderungen
2.3.0	**Note** Wir empfehlen, dass Sie auf Greengrass Nucleus 2.9.1 aktualisieren, wenn Sie auf Log Manager 2.3.0 aktualisieren. Neue Features Reduziert Protokollverzögerungen, indem aktive Protokolldateien verarbeitet und direkt hochgeladen werden, anstatt darauf zu warten, dass neue Dateien rotiert werden. Fehlerkorrekturen und Verbesserungen • Verbessert die Unterstützung der Protokollrotation beim Rotieren von Dateien mit einem eindeutigen Namen. • Zusätzliche kleinere Bugfixes und Verbesserungen.
2.2.8	Die Version wurde für die Version 2.9.0 von Greengrass Nucleus aktualisiert.
2.2.7	Die Version wurde für die Version 2.8.0 von Greengrass Nucleus aktualisiert.
2.2.6	Die Version wurde für die Version 2.7.0 von Greengrass Nucleus aktualisiert.
2.2.5	Die Version wurde für die Version 2.6.0 von Greengrass Nucleus aktualisiert.
2.2.4	Fehlerkorrekturen und Verbesserungen • Verbessert die Stabilität beim Umgang mit ungültigen Konfigurationen. • Zusätzliche kleinere Korrekturen und Verbesserungen.

Version	Änderungen
2.2.3	Fehlerkorrekturen und Verbesserungen • Verbessert die Stabilität in bestimmten Szenarien, in denen die Komponente neu gestartet wird oder Fehler auftreten. • Behebt Probleme, bei denen große Protokollnachrichten und große Protokolldateien in bestimmten Szenarien nicht hochgeladen werden können. • Behebt Probleme mit der Art und Weise, wie diese Komponente Updates zum Zurücksetzen der Konfiguration verarbeitet. • Behebt ein Problem, bei dem ein `null` `diskSpaceLimit` Konfigura tionswert die Bereitstellung der Komponente verhinderte.
2.2.2	Fehlerkorrekturen und Verbesserungen • Integriert die Unterstützung für Protokollnachrichten, die größer als 256 Kilobyte sind. Die Log-Manager-Komponente teilt diese großen Protokoll nachrichten in mehrere Meldungen mit demselben Protokollereignis-Zeitstempel auf.
2.2.1	Die Version wurde für die Version 2.5.0 von Greengrass Nucleus aktualisiert.
2.2.0	Neues Feature • Fügt den `componentLogsConfigurationMap` Konfigurationspara meter hinzu, um ein Zuordnungsformat für Komponentenprotokollkonfigu rationen zu unterstützen. Jedes `componentName` Objekt in der Zuordnung definiert die Protokollkonfiguration für eine Komponente oder Anwendung.
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, bei dem die Systemprotokollkonfiguration in bestimmten Fällen nicht aktualisiert wurde.

Version	Änderungen
2.1.0	Fehlerkorrekturen und Verbesserungen • Verwenden Sie Standardwerte für `logFileDirectoryPath` Greengrass-Komponenten`logFileRegex` , die auf Standardausgabe (stdout) und Standardfehler (stderr) drucken, und die für diese funktioni eren. • Leiten Sie den Datenverkehr korrekt über einen konfigurierten Netzwerk-Proxy weiter, wenn Sie Protokolle in Logs hochladen. CloudWatch • Behandelt Doppelpunkte (`:`) in Protokolldatenstromnamen korrekt. CloudWatch Log-Log-Stream-Namen unterstützen keine Doppelpunkte. • Vereinfachen Sie die Namen von Protokolldatenströmen, indem Sie die Namen der Dinggruppen aus dem Protokollstream entfernen. • Entfernen Sie eine Fehlerprotokollmeldung, die bei normalem Verhalten ausgegeben wird.
2.0.x	Erste Version

Komponenten für maschinelles Lernen

AWS IoT Greengrass bietet die folgenden Komponenten für maschinelles Lernen, die Sie auf unterstützten Geräten bereitstellen können, um Inferenzen für maschinelles Lernen mithilfe von Modellen durchzuführen, die in Amazon SageMaker AI trainiert wurden, oder mit Ihren eigenen vortrainierten Modellen, die in Amazon S3 gespeichert sind.

AWS bietet die folgenden Kategorien von Komponenten für maschinelles Lernen:

- Modellkomponente — Enthält Modelle für maschinelles Lernen als Greengrass-Artefakte.
- Runtime-Komponente — Enthält das Skript, das das Machine Learning-Framework und seine Abhängigkeiten auf dem Greengrass-Core-Gerät installiert.
- Inferenzkomponente — Enthält den Inferenzcode und enthält Komponentenabhängigkeiten zur Installation des Frameworks für maschinelles Lernen und zum Herunterladen vorab trainierter Modelle für maschinelles Lernen.

Sie können den Beispiel-Inferenzcode und die vortrainierten Modelle in den AWS bereitgestellten Komponenten für maschinelles Lernen verwenden, um die Bildklassifizierung und Objekterkennung mit DLR und Lite durchzuführen. TensorFlow Um benutzerdefinierte Inferenzen für maschinelles Lernen mit Ihren eigenen Modellen durchzuführen, die in Amazon S3 gespeichert sind, oder um ein anderes Framework für maschinelles Lernen zu verwenden, können Sie die Rezepte dieser öffentlichen Komponenten als Vorlagen verwenden, um benutzerdefinierte maschinelle Lernkomponenten zu erstellen. Weitere Informationen finden Sie unter Passen Sie Ihre Komponenten für maschinelles Lernen an.

AWS IoT Greengrass enthält auch eine AWS bereitgestellte Komponente zur Verwaltung der Installation und des Lebenszyklus des SageMaker AI Edge Manager-Agenten auf Greengrass-Kerngeräten. Mit SageMaker AI Edge Manager können Sie mit Amazon SageMaker AI Neo kompilierte Modelle direkt auf Ihrem Kerngerät verwenden. Weitere Informationen finden Sie unter Verwenden Sie Amazon SageMaker AI Edge Manager auf Greengrass-Kerngeräten.

In der folgenden Tabelle sind die Komponenten für maschinelles Lernen aufgeführt, die in AWS IoT Greengrass verfügbar sind.

> **Note**
>
> Einige der AWS bereitgestellten Komponenten hängen von bestimmten Nebenversionen des Greengrass-Kerns ab. Aufgrund dieser Abhängigkeit müssen Sie diese Komponenten aktualisieren, wenn Sie den Greengrass Nucleus auf eine neue Nebenversion aktualisieren. Informationen zu den spezifischen Versionen von Nucleus, von denen jede Komponente abhängt, finden Sie im entsprechenden Komponententhema. Weitere Informationen zur Aktualisierung des Nucleus finden Sie unterAktualisieren Sie die AWS IoT Greengrass Core-Software (OTA).

Wenn eine Komponente einen Komponententyp sowohl generisch als auch Lambda hat, ist die aktuelle Version der Komponente der generische Typ und eine frühere Version der Komponente ist der Lambda-Typ.

Komponente	Description	Komponententyp	Unterstützes Betriebssystem	Open-Source
SageMaker KI-Edge-Manager	Stellt den Amazon SageMaker AI Edge Manager-Agenten auf dem Greengrass-Core-Gerät bereit.	Generisch	Linux, Windows	Nein
DLR-Bildklassifizierung	Inferenzkomponente, die den DLR-Bildklassifizierungsmodellspeicher und die DLR-Laufzeitkomponente als Abhängigkeiten verwendet, um DLR zu installieren, Beispielmodelle für die Bildklassifizierung herunterzuladen und Inferenzen	Generisch	Linux, Windows	Nein

Komponente	Description	Komponententyp	Unterstützes Betriebssystem	Open-Source
	zur Bildklassifizierung auf unterstützten Geräten durchzuführen.			
DLR-Objekterkennung	Inferenzkomponente, die den DLR-Objekterkennungsmodellspeicher und die DLR-Laufzeitkomponente als Abhängigkeiten verwendet, um DLR zu installieren, Beispielmodelle zur Objekterkennung herunterzuladen und Objekterkennungsinferenzen auf unterstützten Geräten durchzuführen.	Generisch	Linux, Windows	Nein

Komponente	Description	Komponententyp	Unterstützes Betriebssystem	Open-Source
Modellspeicher für die DLR-Bildklassifizierung	Modellkomponente, die Stichprobe ResNet -50 Bildklassifizierungsmodelle als Greengrass-Artefakte enthält.	Generisch	Linux, Windows	Nein
Modellspeicher für DLR-Objekterkennung	Modellkomponente, die Beispielmodelle zur YOLOv3 Objekterkennung als Greengrass-Artefakte enthält.	Generisch	Linux, Windows	Nein

Komponente	Description	Komponententyp	Unterstützes Betriebssystem	Open-Source
DLR-Laufzeit	Runtime-Komponente, die ein Installationsskript enthält, das zur Installation von DLR und seinen Abhängigkeiten auf dem Greengrass-Core-Gerät verwendet wird.	Generisch	Linux, Windows	Nein

Komponente	Description	Komponententyp	Unterstützes Betriebssystem	Open-Source
TensorFlow Lite-Bildklassifizierung	Inferenzkomponente, die den TensorFlow Lite-Bildklassifizierungsmodellspeicher und die TensorFlow Lite-Runtime-Komponente als Abhängigkeiten verwendet, um TensorFlow Lite zu installieren, Beispielmodelle für die Bildklassifizierung herunterzuladen und die Inferenz für die Bildklassifizierung auf unterstützten Geräten durchzuführen.	Generisch	Linux, Windows	Nein

Komponente	Description	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, Beispielmodelle für die Objekterkennung herunterzuladen und Objekterkennungsinferenzen auf unterstützten Geräten durchzuführen.	Generisch	Linux, Windows	Nein

Komponente	Description	Komponententyp	Unterstützes Betriebssystem	Open-Source
TensorFlow Modellspeicher für Lite-Bildklassifizierung	Modellkomponente, die ein Beispiel für ein MobileNet v1-Modell als Greengrass-Artefakt enthält.	Generisch	Linux, Windows	Nein
TensorFlow Modellspeicher für Lite-Objekterkennung	Modellkomponente, die ein SSD-Beispielmodell (Single Shot Detection) als MobileNet Greengrass-Artefakt enthält.	Generisch	Linux, Windows	Nein

Komponente	Description	Komponententyp	Unterstützes Betriebssystem	Open-Source
TensorFlow Lite-Laufzeit	Runtime-Komponente, die ein Installationsskript enthält, das zur Installation von TensorFlow Lite und seinen Abhängigkeiten auf dem Greengrass-Core-Gerät verwendet wird.	Generisch	Linux, Windows	Nein

SageMaker KI-Edge-Manager

> **Important**
>
> SageMaker AI Edge Manager wurde am 26. April 2024 eingestellt. Weitere Informationen zur weiteren Bereitstellung Ihrer Modelle auf Edge-Geräten finden Sie unter Ende der Nutzungsdauer von SageMaker AI Edge Manager.

Die Amazon SageMaker AI Edge Manager-Komponente (`aws.greengrass.SageMakerEdgeManager`) installiert die SageMaker AI Edge Manager-Agent-Binärdatei.

SageMaker AI 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 AI Edge Manager-Komponente installiert und verwaltet den

Lebenszyklus des SageMaker AI Edge Manager-Agenten auf Ihrem Kerngerät. Sie können SageMaker AI Edge Manager auch verwenden, um mit SageMaker KI Neo kompilierte Modelle als Modellkomponenten auf Greengrass-Kerngeräten zu verpacken und zu verwenden. Weitere Informationen zur Verwendung des SageMaker AI Edge Manager-Agenten auf Ihrem Kerngerät finden Sie unter. Verwenden Sie Amazon SageMaker AI Edge Manager auf Greengrass-Kerngeräten

SageMaker Die AI 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 AI 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
- Typ
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Konfiguration
- Lokale Protokolldatei
- Änderungsprotokoll

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 AI 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 KI-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 AI 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 Geräteverkehr über einen Proxy oder eine Firewall zulassen.

Endpoint	Port	Erforderlich	Description
`edge.sagemaker.` *`region`*`.amazonaws.com`	443	Ja	Überprüfen Sie den Status der Geräteregistrierung und senden Sie Messwerte an KI. SageMaker
`*.s3.amazonaws.com`	443	Ja	Laden Sie die Erfassungsdaten in den von Ihnen angegebenen S3-

Endpoint	Port	Erforderlich	Description
			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

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
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 AI Edge Manager-Konfiguration finden Sie unter Edge Manager Agent im Amazon SageMaker AI Developer Guide.

`DeviceFleetName`

Der Name der SageMaker AI 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**
>
> Capture Data ist eine SageMaker KI-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 AI Edge Manager finden Sie unter Modell verwalten im Amazon SageMaker AI Developer Guide.

`CaptureDataBatchSize`

(Optional) Die Größe eines Stapels von Erfassungsdatenanfragen, 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. Folgende Parameterwerte sind möglich:

- `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 AI 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 AI 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 AI 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 AI 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 AI 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 Von AI 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 AI Edge Manager-Komponente verfügbar.

(Optional) Sie können diese Komponente aktivieren, um automatisch mit SageMaker KI Neo kompilierte Modelle abzurufen, von denen Sie auf Amazon S3 hochladen. Nachdem Sie ein neues Modell auf Amazon S3 hochgeladen haben, verwenden Sie SageMaker AI Studio oder die SageMaker KI-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 AI Edge Manager-Komponente verfügbar.

(Optional) Der Zeitraum (in Minuten), zwischen dem diese Komponente nach neuen Modellen zum Herunterladen sucht. 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 AI 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 AI 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": "amzn-s3-demo-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 Options-Parameter das DLRBackend 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 bei, wie GPUs z.
1.2.0	Neue Features • Fügt Unterstützung für diese Komponente hinzu, um automatisch mit SageMaker KI Neo kompilierte Modelle abzurufen, die Sie auf Amazon S3 hochladen. Wenn Sie diese Funktion aktivieren, können Sie neue Modelle auf Kerngeräten bereitstellen, ohne eine AWS IoT Greengrass Bereitstellung erstellen zu müssen.

Version	Änderungen
	• 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.
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
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

-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 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

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
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 read/write Zugriff haben.

Standard: `/greengrass/v2/packages/artifacts-unarchived/component-name/image_classification/sample_images/`

> **Note**
>
> Wenn Sie den Wert `UseCamera` auf setzen`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, read/write Zugriff 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

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, Zugriff hat read/write .

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.
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 (AArch64) -Plattformen 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 Inferenze rgebnissen 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 Komponentenkonfigu ration 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
- 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 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 read/write Zugriff haben.

Standard: *`/greengrass/v2`*`/packages/artifacts-unarchived/`*`component-name`*`/object_detection/sample_images/`

> **Note**
>
> Wenn Sie den Wert `UseCamera` auf setzen`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, read/write Zugriff 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

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, Zugriff hat read/write .

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 (AArch64) -Plattformen 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 Inferenze rgebnissen 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 Komponentenkonfigu ration 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 AI 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
- Typ
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Konfiguration
- Lokale Protokolldatei
- Änderungsprotokoll

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 Bildklassifizierungsmodell ResNet -50 für Armv8 () -Plattformen hinzu. AArch64 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-Modellspeicher zur Objekterkennung 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 AI Neo Deep Learning Runtime SageMaker kompiliert.

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
- 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 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.

2.1.13 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 () AArch64 -Geräten 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 () AArch64 -Plattformen 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

- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Konfiguration
- Usage
- Lokale Protokolldatei
- Änderungsprotokoll

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 Repositorys, in die Sie Pakete herunterladen, und der Besitzer eines Pakets kann ändern, was URLs der Paketindex bereitstellt.

Weitere Informationen zu den Abhängigkeiten, die diese Komponente installiert, und zur Deaktivierung des Installationsskripts finden Sie unter dem UseInstallerKonfigurationsparameter.

Weitere Informationen zu Endpunkten und Ports, die für den Basisbetrieb erforderlich sind, finden Sie unterGeräteverkehr über einen Proxy oder eine Firewall 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 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, Zugriff hat read/write .

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, Zugriff hat read/write .

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`

Usage

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`	Keine
Ubuntu	`wget`	Keine

Wenn Sie Ihre Inferenzkomponente bereitstellen, überprüft diese Runtime-Komponente zunächst, ob DLR und seine Abhängigkeiten bereits auf Ihrem Gerät installiert sind, und falls nicht, installiert sie sie dann für Sie.

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 () AArch64 -Plattfor men hinzugefügt. Dies erweitert die Unterstützung für maschinelles Lernen für Greengrass-Kerngeräte, auf denen NVIDIA Jetson ausgeführ t 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
- 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

-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 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 read/write Zugriff haben.

Standard: */greengrass/v2*`/packages/artifacts-unarchived/`*component-name*`/image_classification/sample_images/`

> **Note**
>
> Wenn Sie den Wert `UseCamera` auf setzen`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: `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, read/write Zugriff 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 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.

2.1.11 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 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 read/write Zugriff haben.

Standard: */greengrass/v2*`/packages/artifacts-unarchived/`*component-name*`/object_detection/sample_images/`

> **Note**
>
> Wenn Sie den Wert `UseCamera` auf setzen`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, read/write Zugriff 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
- 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 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.

2.1.11 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
- 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

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
- Typ
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Konfiguration
- Usage
- Lokale Protokolldatei
- Änderungsprotokoll

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 Repositorys, in die Sie Pakete herunterladen, und der Besitzer eines Pakets kann ändern, was URLs der Paketindex bereitstellt.

Weitere Informationen zu den Abhängigkeiten, die diese Komponente installiert, und zur Deaktivierung des Installationsskripts finden Sie unter dem UseInstallerKonfigurationsparameter.

Weitere Informationen zu Endpunkten und Ports, die für den Basisbetrieb erforderlich sind, finden Sie unterGeräteverkehr über einen Proxy oder eine Firewall 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 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, Zugriff hat read/write .

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, Zugriff hat read/write .

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`.

Setzen Sie diesen Wert auf, `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`

Usage

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`	Keine
Ubuntu	`wget`	Keine

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

Modbus-RTU-Protokolladapter

Die Modbus-RTU-Protokolladapter-Komponente (`aws.greengrass.Modbus`) fragt Informationen von lokalen Modbus RTU-Geräten ab.

Um mit dieser Komponente Informationen von einem lokalen Modbus RTU-Gerät anzufordern, veröffentlichen Sie eine Nachricht zu dem Thema, das diese Komponente abonniert. Geben Sie in der Nachricht die Modbus RTU-Anforderung an, die an ein Gerät gesendet werden soll. Anschließend veröffentlicht diese Komponente eine Antwort, die das Ergebnis der Modbus-RTU-Anfrage enthält.

> **Note**
>
> Diese Komponente bietet ähnliche Funktionen wie der Modbus RTU-Protokolladapter-Anschluss in V1. AWS IoT Greengrass Weitere Informationen finden Sie unter Modbus RTU-Protokolladapter-Anschluss im V1-Entwicklerhandbuch.AWS IoT Greengrass

Themen

- Versionen
- Typ
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Konfiguration
- Eingabedaten
- Ausgabedaten
- Modbus RTU-Anforderungen und -Antworten
- Lokale Protokolldatei
- Lizenzen
- Änderungsprotokoll

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 mithilfe der Lambda-Launcher-Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann nur auf Linux-Kerngeräten installiert werden.

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.
- Python-Version 3.7 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.
- Eine physische Verbindung zwischen dem AWS IoT Greengrass Kerngerät und den Modbus-Geräten. Das Kerngerät muss über eine serielle Schnittstelle, z. B. einen USB-Anschluss, physisch mit dem Modbus RTU-Netzwerk verbunden sein.
- Um Ausgabedaten von dieser Komponente zu empfangen, müssen Sie bei der Bereitstellung dieser Komponente das folgende Konfigurationsupdate für die ältere Abonnement-Router-Komponente (`aws.greengrass.LegacySubscriptionRouter`) zusammenführen. Diese Konfiguration gibt das Thema an, zu 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"
    }
  }
}
```

- *region*Ersetzen Sie es durch AWS-Region das, was Sie verwenden.
- *version*Ersetzen Sie durch die Version der Lambda-Funktion, die diese Komponente ausführt. Um die Version der Lambda-Funktion zu finden, müssen Sie sich das Rezept für die Version dieser Komponente ansehen, 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 Komponentenversion verwenden, die Sie bereitstellen.

Weitere Informationen finden Sie unter Erstellen von Bereitstellungen.

- Der Modbus-RTU-Protokolladapter wird für die Ausführung in einer VPC unterstützt.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt er 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

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 <2.16.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^2.0.0	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 <2.15.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Token-Austauschdienst	^2.0.0	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 <2.14.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^2.0.0	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 <2.13.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.12.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.11.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.10.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.9.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.8.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.7.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.6.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.5.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.4.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.3.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.2.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.3 <2.1.0	Hart
Lambda-Launcher	>=1.0.0	Hart
Lambda-Laufzeiten	>=1,0.0	Weich
Token-Austauschdienst	>=1.0.0	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**
>
> Die Standardkonfiguration dieser Komponente umfasst Lambda-Funktionsparameter. Wir empfehlen, dass Sie nur die folgenden Parameter 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 zur physischen seriellen Modbus-Schnittstelle auf dem Kerngerät, 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 read/write Zugriff auf das Gerät haben.

`ModbusBaudRate`

(Optional) Ein Zeichenkettenwert, der die Baudrate für die serielle Kommunikation mit lokalen Modbus-TCP-Geräten angibt.

Standard: `9600`

`ModbusByteSize`

(Optional) Ein Zeichenkettenwert, der die Größe eines Bytes bei der seriellen Kommunikation mit lokalen Modbus-TCP-Geräten angibt. Wählen Sie`5`, `6`7, oder `8` Bits.

Standard: `8`

`ModbusParity`

(Optional) Der Paritätsmodus, der zur Überprüfung der Datenintegrität bei der seriellen Kommunikation mit lokalen Modbus-TCP-Geräten verwendet werden soll.

- `E`— Überprüfen Sie die Datenintegrität mit gleichmäßiger Parität.
- `0`— Überprüfen Sie die Datenintegrität mit ungerader Parität.
- `N`— Überprüfen Sie nicht die Datenintegrität.

Standard: `N`

`ModbusStopBits`

(Optional) Ein Zeichenkettenwert, der die Anzahl der Bits angibt, die das Ende eines Bytes bei der seriellen Kommunikation mit lokalen Modbus-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 Modbus-Gerät zu gewähren.
- `NoContainer`— Die Komponente läuft nicht in einer isolierten Laufzeitumgebung.

Standard: `GreengrassContainer`

`containerParams`

(Optional) Ein Objekt, das die Container-Parameter 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 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, das Sie konfigurieren, in der `ModbusLocalPort` Umgebungsvariablen angeben.

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 Kerngerät. Dieser Wert muss denselben Wert haben wie der Wert, für den Sie die Konfiguration konfigurieren`ModbusLocalPort`.

`permission`

(Optional) Die Berechtigung, vom Container aus auf das Systemgerät zuzugreifen. Dieser Wert muss sein`rw`, was angibt, dass die Komponente read/write Zugriff auf das Systemgerät hat.

Standard: `rw`

`addGroupOwner`

(Optional) 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, für die 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 Themen abonniert. publish/subscribe

Dieses Objekt enthält die folgenden Informationen:

`0`— Dies ist ein Array-Index als Zeichenfolge.

Ein Objekt, das die folgenden Informationen enthält:

`type`

(Optional) Der publish/subscribe Messaging-Typ, den diese Komponente verwendet, um Nachrichten zu abonnieren. 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 aus, 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 aus 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 Container-Modus)

```
{
  "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 zur physischen seriellen Modbus-Schnittstelle auf dem Kerngerät, 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 read/write Zugriff auf das Gerät haben.

`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 Modbus-Gerät zu gewähren.
- `NoContainer`— Die Komponente läuft nicht in einer isolierten Laufzeitumgebung.

Standard: `GreengrassContainer`

`containerParams`

(Optional) Ein Objekt, das die Container-Parameter 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 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, das Sie konfigurieren, in der `ModbusLocalPort` Umgebungsvariablen angeben.

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 Kerngerät. Dieser Wert muss denselben Wert haben wie der Wert, für den Sie die Konfiguration konfigurieren`ModbusLocalPort`.

`permission`

(Optional) Die Berechtigung, vom Container aus auf das Systemgerät zuzugreifen. Dieser Wert muss sein`rw`, was angibt, dass die Komponente read/write Zugriff auf das Systemgerät hat.

Standard: `rw`

`addGroupOwner`

(Optional) 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, für die 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 Themen abonniert. publish/subscribe

Dieses Objekt enthält die folgenden Informationen:

`0`— Dies ist ein Array-Index als Zeichenfolge.

Ein Objekt, das die folgenden Informationen enthält:

`type`

(Optional) Der publish/subscribe Messaging-Typ, den diese Komponente verwendet, um Nachrichten zu abonnieren. 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 aus, 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 aus 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 Container-Modus)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "NoContainer"
}
```

Eingabedaten

Diese Komponente akzeptiert Modbus-RTU-Anforderungsparameter zum folgenden Thema und sendet die Modbus-RTU-Anforderung an das Gerät. Standardmäßig abonniert diese Komponente lokale Nachrichten. publish/subscribe Weitere Informationen zum Veröffentlichen von Nachrichten aus Ihren benutzerdefinierten Komponenten in dieser Komponente finden Sie unterLokale 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 Modbus-RTU-Anforderung.

Die Form der Anforderungsnachricht hängt vom Typ der Modbus-RTU-Anforderung ab, die sie darstellt. Die folgenden Eigenschaften sind für alle Anfragen erforderlich.

Typ: `object` der die folgenden Informationen enthält:

`operation`

Der Name des auszuführenden Vorgangs. Geben Sie beispielsweise an, `ReadCoilsRequest` dass Spulen auf einem Modbus-RTU-Gerät gelesen werden sollen. Weitere Informationen zu unterstützten Vorgängen 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 führt die zyklische Redundanzprüfung (CRC) durch, um Datenanfragen für Sie zu verifizieren.

> **Note**
>
> Wenn Ihre Anfrage 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, setzt die Komponente die `id` Eigenschaft im Antwortobjekt auf diesen Wert.

Typ: `string`

Example Beispieleingabe: Lesen Spulenanforderungen

```
{
  "request": {
    "operation": "ReadCoilsRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "MyRequest"
}
```

Ausgabedaten

Diese Komponente veröffentlicht standardmäßig Antworten als Ausgabedaten zum folgenden MQTT-Thema. Sie müssen dieses Thema `subject` in der Konfiguration für die ältere Abonnement-Router-Komponente angeben. Weitere Informationen zum Abonnieren von Nachrichten zu diesem Thema in Ihren benutzerdefinierten Komponenten finden Sie unterMQTT-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 Modbus RTU-Geräts.

Typ: der `object` die folgenden Informationen enthält:

`status`

Der Status der Anforderung. Der Status kann einer der folgenden Werte sein:

- `Success`— Die Anfrage war gültig, die Komponente hat die Anfrage an das Modbus RTU-Netzwerk gesendet und das Modbus RTU-Netzwerk hat eine Antwort zurückgegeben.

- `Exception`— Die Anfrage war gültig, die Komponente hat die Anfrage an das Modbus RTU-Netzwerk gesendet und das Modbus RTU-Netzwerk hat eine Ausnahme zurückgegeben. Weitere Informationen finden Sie unter Antwortstatus: Ausnahme.
- `No Response`— Die Anfrage war ungültig und die Komponente hat den Fehler erkannt, bevor sie die Anfrage an das Modbus 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 Anfrage gesendet hat.

`payload`

Die Antwort des Modbus RTU-Geräts. Wenn ja `status`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 Anfrage, anhand derer Sie ermitteln können, welche Antwort welcher Anfrage entspricht.

> **Note**
>
> Eine Antwort für einen Schreibvorgang ist lediglich ein Echo der Anforderung. Schreibantworten enthalten zwar keine aussagekräftigen Informationen, es empfiehlt sich jedoch, den Status der Antwort zu überprüfen, um festzustellen, ob die Anfrage erfolgreich ist oder nicht.

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

Vorgangsname in Anforderung	Funktionscode in Antwort
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 die Spulen

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 Sie diskrete Eingänge

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 Sie die Halteregister

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"
}
```

Eingaberegister lesen

Anfragebeispiel:

```
{
  "request": {
    "operation": "ReadInputRegistersRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "TestRequest"
}
```

Schreiben Sie Single Coil

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 Sie ein einzelnes Register

Anfragebeispiel:

```
{
  "request": {
    "operation": "WriteSingleRegisterRequest",
    "device": 1,
    "address": 1,
    "value": 1
  },
  "id": "TestRequest"
}
```

Schreiben Sie mehrere Spulen

Anfragebeispiel:

```
{
  "request": {
    "operation": "WriteMultipleCoilsRequest",
    "device": 1,
    "address": 1,
    "values": [1,0,0,1]
  },
  "id": "TestRequest"
}
```

Antwortbeispiel:

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "WriteMultipleCoilsRequest",
    "payload": {
      "function_code": 15,
      "address": 1,
      "count": 4
    }
  },
  "id" : "TestRequest"
}
```

Schreiben Sie mehrere 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"
}
```

Maske schreiben, Register schreiben

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"
```

```
}
```

Mehrere Register lesen, schreiben

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 beinhaltet die Register, 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
```

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.greengrass.Modbus.log
```

Lizenzen

Diese Komponente umfasst die folgende Software/Lizenzierung von Drittanbietern:

- pymodbus /BSD-Lizenz
- pyserial /BSD-Lizenz

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
2.1.11	Die Version wurde für die Version 2.15.0 von Greengrass Nucleus aktualisiert.
2.1.10	Die Version wurde für die Version 2.14.0 von Greengrass Nucleus aktualisiert.
2.1.9	Die Version wurde für die Version 2.13.0 von Greengrass Nucleus aktualisiert.
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	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem mit dem `ReadDiscreteInput` Vorgang.
2.1.4	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.

Version	Änderungen
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 die `ModbusStopBits` OptionenModbusBaudRate ,ModbusByteSize , und hinzuModbusParity , die Sie angeben können, um die serielle Kommunikation mit Modbus RTU-Geräten zu konfigurieren.
2.0.8	Die Version wurde für die Version 2.5.0 von Greengrass Nucleus aktualisiert.
2.0.7	Die Version wurde für die Version 2.4.0 von Greengrass Nucleus aktualisiert.
2.0.6	Die Version wurde für die Version 2.3.0 von Greengrass Nucleus aktualisiert.
2.0.5	Die Version wurde für die Version 2.2.0 von Greengrass Nucleus aktualisiert.
2.0.4	Die Version wurde für die Version 2.1.0 von Greengrass Nucleus 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 Sie 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.
- Lokal publish/subscribe — 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
- 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 Core-Gerät ändern.

Diese Komponente verwendet dieselbe Protokolldatei wie der Greengrass-Kern. Weitere Informationen finden Sie unter AWS IoT Greengrass Protokolle überwachen.

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. *`topicMappingNameKey`*Ersetzen Sie ihn durch einen Namen, anhand dessen Sie diese Themenzuordnung leichter identifizieren können.

Dieses Objekt enthält die folgenden Informationen:

`topic`

Das Thema oder der Themenfilter als Brücke zwischen dem Quell- und dem Zielbroker.

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 publish/subscribe Greengrass-Nachrichtenbroker.
- `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-Message-Broker. Wählen Sie aus den folgenden Optionen aus:

- `LocalMqtt`— Der lokale MQTT-Broker, über den Client-Geräte kommunizieren.
- `Pubsub`— Der lokale publish/subscribe Greengrass-Nachrichtenbroker.

- `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:

`mqtt5RouteOptionsNameKey`

Der Name der Routenoptionen für ein Topic-Mapping. *`mqtt5RouteOptionsNameKey`*Ersetzen Sie ihn durch den im `mqttTopicMapping` Feld *`topicMappingNameKey`* definierten Treffer.

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"
        }
    }
}
```

`noLocal`wird nur für Routen unterstützt, bei 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 es durch den Port, an *`port`* 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.
- Leiten Sie Nachrichten zu Themen, die publish/subscribe dem Themenfilter entsprechen, von Client-Geräten an lokale Geräte weiter und fügen Sie dem Zielthema das `events/input/` Präfix hinzu. `clients/+/detections` 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. *`topicMappingNameKey`*Ersetzen Sie ihn durch einen Namen, anhand dessen Sie diese Themenzuordnung leichter identifizieren können.

Dieses Objekt enthält die folgenden Informationen:

`topic`

Das Thema oder der Themenfilter als Brücke zwischen dem Quell- und dem Zielbroker.

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 publish/subscribe Greengrass-Nachrichtenbroker.
- `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-Message-Broker. Wählen Sie aus den folgenden Optionen aus:

- `LocalMqtt`— Der lokale MQTT-Broker, über den Client-Geräte kommunizieren.
- `Pubsub`— Der lokale publish/subscribe Greengrass-Nachrichtenbroker.
- `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 es durch den Port, an *`port`* 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.
- Leiten Sie Nachrichten zu Themen, die publish/subscribe dem Themenfilter entsprechen, von Client-Geräten an lokale Geräte weiter und fügen Sie dem Zielthema das `events/input/` Präfix hinzu. `clients/+/detections` 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. *`topicMappingNameKey`*Ersetzen Sie ihn durch einen Namen, anhand dessen Sie diese Themenzuordnung leichter identifizieren können.

Dieses Objekt enthält die folgenden Informationen:

`topic`

Das Thema oder der Themenfilter als Brücke zwischen dem Quell- und dem Zielbroker.

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 publish/subscribe Greengrass-Nachrichtenbroker.
- `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-Message-Broker. Wählen Sie aus den folgenden Optionen aus:

- `LocalMqtt`— Der lokale MQTT-Broker, über den Client-Geräte kommunizieren.
- `Pubsub`— Der lokale publish/subscribe Greengrass-Nachrichtenbroker.

- `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 es durch den Port, an *`port`* 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.
- Leiten Sie Nachrichten zu Themen, die publish/subscribe dem Themenfilter entsprechen, von Client-Geräten an lokale Geräte weiter und fügen Sie dem Zielthema das `events/input/` Präfix hinzu. `clients/+/detections` 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. *`topicMappingNameKey`*Ersetzen Sie ihn durch einen Namen, anhand dessen Sie diese Themenzuordnung leichter identifizieren können.

Dieses Objekt enthält die folgenden Informationen:

`topic`

Das Thema oder der Themenfilter als Brücke zwischen dem Quell- und dem Zielbroker.

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 publish/subscribe Greengrass-Nachrichtenbroker.
- `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-Message-Broker. Wählen Sie aus den folgenden Optionen aus:

- `LocalMqtt`— Der lokale MQTT-Broker, über den Client-Geräte kommunizieren.
- `Pubsub`— Der lokale publish/subscribe Greengrass-Nachrichtenbroker.
- `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 es durch den Port, an *`port`* 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. *`topicMappingNameKey`*Ersetzen Sie ihn durch einen Namen, anhand dessen Sie diese Themenzuordnung leichter identifizieren können.

Dieses Objekt enthält die folgenden Informationen:

`topic`

Das Thema oder der Themenfilter als Brücke zwischen dem Quell- und dem Zielbroker.

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 publish/subscribe Greengrass-Nachrichtenbroker.
- `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-Message-Broker. Wählen Sie aus den folgenden Optionen aus:

- `LocalMqtt`— Der lokale MQTT-Broker, über den Client-Geräte kommunizieren.
- `Pubsub`— Der lokale publish/subscribe Greengrass-Nachrichtenbroker.
- `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 `startupTimeoutSeconds` Konfigurationsoption hinzu.
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 Quell-Nachrichtenbroker angeben.

Version	Änderungen
	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.
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. Wählen Sie einen MQTT-Broker

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 Sie mit lokalen IoT-Geräten.

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 Core-Gerät ändern.

Diese Komponente verwendet dieselbe Protokolldatei wie der Greengrass-Kern. Weitere Informationen finden Sie unter AWS IoT Greengrass Protokolle überwachen.

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 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.

2.3.7

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.

Version	Änderungen
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 Clients beim Senden oder Empfangen von Nachrichten zu ungültigen Sitzungsf ehlern kommen kann IDs. 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.
2.3.0	Fügt Unterstützung für Zertifikatsketten hinzu.
2.2.0	Die Version wurde für die Version 2.2.0 der Client-Geräteauthentifizierung aktualisiert.

Version	Änderungen
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. Wählen Sie einen MQTT-Broker

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 Sie mit lokalen IoT-Geräten.

Themen

- Versionen
- Typ
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Konfiguration
- Lokale Protokolldatei
- Lizenzen
- Änderungsprotokoll

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.

2.0.2 – 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
Authentifizierung auf dem Client-Gerät	>=2.2.0 <2.6.0	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
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 - 2.0.3

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 Standardeinstellungen, die für den MQTT 5 (EMQX) -Broker festgelegt wurden. 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",
                   "ssl_options": {
                        "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.

Im folgenden Beispiel wird die `etc/emqx.conf` Datei aktualisiert.

```
"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, um die angegebenen EMQX-Konfigurationsdateien zu ersetzen. 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 verwendenetc/`emqx.conf`.

Example Beispiel: Update zur Zusammenführung von Konfigurationen

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
2.0.3	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem EMQX unter Windows nicht gestartet wird, wenn der Pfad Leerzeichen enthält.
2.0.2	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem EMQX gestartet wird, bevor die Komponente für die Client-Geräteauthentifizierung bereit ist.
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

Nucleus-Telemetrie-Sender

Die Nucleus-Telemetrie-Emitter-Komponente (`aws.greengrass.telemetry.NucleusEmitter`) sammelt Telemetriedaten zur Systemintegrität und veröffentlicht sie kontinuierlich zu einem lokalen Thema und einem MQTT-Thema. AWS IoT Core Mit dieser Komponente können Sie Systemtelemetrie in Echtzeit auf Ihren Greengrass-Kerngeräten erfassen. Informationen zum

Greengrass-Telemetrie-Agenten, der Systemtelemetrie-Daten auf Amazon EventBridge veröffentlicht, finden Sie unter. Erfassung von Telemetriedaten zur Systemintegrität von AWS IoT Greengrass Kerngeräten

Standardmäßig veröffentlicht die Nucleus-Telemetrie-Emitter-Komponente alle 60 Sekunden Telemetriedaten zum folgenden lokalen Thema. publish/subscribe

```
$local/greengrass/telemetry
```

Die Nucleus-Telemetrie-Emitter-Komponente veröffentlicht standardmäßig nicht in 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 wird. Die Verwendung eines MQTT-Topics zur Veröffentlichung von Daten in der AWS Cloud ist kostenpflichtig AWS IoT Core .

AWS IoT Greengrass bietet mehrere Community-Komponenten, mit denen Sie Telemetriedaten mithilfe von InfluxDB und Grafana lokal auf Ihrem Kerngerät analysieren und visualisieren können. Diese Komponenten verwenden Telemetriedaten von der Nukleus-Emitter-Komponente. Weitere Informationen finden Sie in der README-Datei für die InfluxDB-Publisher-Komponente.

Themen

- Versionen
- Typ
- Betriebssystem
- Abhängigkeiten
- Konfiguration
- Ausgabedaten
- Usage
- Lokale Protokolldatei
- Änderungsprotokoll

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 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 AWS IoT Greengrass Protokolle überwachen.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann auf Kerngeräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

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.

1.0.12

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.0.12 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2,4,0 <2,17,0	Weich

1.0.11

In der folgenden Tabelle sind die Abhängigkeiten für Version 1.0.11 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2,4,0 <2,16,0	Hart

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	>=2,4,0 <2,15,0	Hart

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	>=2.4.0 <2.14.0	Hart

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	>=2,4,0 <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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=2,4,0 <2,5,0	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.

`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.

Setzen Sie diesen Wert auf das AWS IoT Core MQTT-Thema, in dem Sie Telemetriedaten veröffentlichen möchten. Wenn dieser Wert leer ist, veröffentlicht der Nucle-Emitter keine Telemetriedaten auf dem. AWS Cloud

> **Note**
>
> Die Verwendung eines MQTT-Themas zur Veröffentlichung von Daten auf der AWS Cloud ist kostenpflichtig.AWS IoT Core

Standard: `""`

`telemetryPublishIntervalMs`

(Optional) Der Zeitraum (in Millisekunden), zwischen dem 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 Kerngerät. Wir empfehlen, mit dem Standard-Veröffentlichungsintervall zu beginnen und es an die CPU-Auslastung Ihres Geräts anzupassen.

Minimum: `500`

Standard: `60000`

Example Beispiel: Aktualisierung zur Zusammenführung von Konfigurationen

Das folgende Beispiel zeigt ein Beispiel für ein Update zur Zusammenführung von Konfigurationen, das die Veröffentlichung von Telemetriedaten für das `$local/greengrass/telemetry` Thema und das `greengrass/myTelemetry` AWS IoT Core MQTT-Thema alle 5 Sekunden ermöglicht.

```
{
```

```
  "pubSubPublish": "true",
  "mqttTopic": "greengrass/myTelemetry",
  "telemetryPublishIntervalMs": 5000
}
```

Ausgabedaten

Diese Komponente veröffentlicht Telemetrie-Metriken als JSON-Array zum folgenden Thema.

Lokales Thema: `$local/greengrass/telemetry`

Sie können sich optional dafür entscheiden, auch Telemetrie-Metriken zu einem AWS IoT Core MQTT-Thema zu veröffentlichen. Weitere Informationen zu den Themen finden Sie unter MQTT-Themen im AWS IoT Core Developer Guide.

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 der durchschnittlichen CPU-Auslastung seit dem letzten Veröffentlichungsereignis entspricht.

Bei allen anderen Metriken aggregiert der Nucleus-Emitter den Metrikwert nicht, und diese Eigenschaft ist auf `Count` gesetzt.

`N`

Name der Metrik.

`NS`

Der Metrik-Namespace.

`TS`

Der Zeitstempel, zu dem die Daten gesammelt wurden.

`U`

Die Einheit des metrischen Werts.

`V`

Der -Metrikwert

Der Nukleus-Emitter veröffentlicht die folgenden Metriken:

Name	Description	
System (System)		
`SystemMemUsage`	Die Menge an Speicher, die derzeit von allen Anwendung en auf dem Greengrass-Core-Gerät verwendet wird, einschließlich des Betriebss ystems.	
`CpuUsage`	Die Menge an CPU, die derzeit von allen Anwendung en auf dem Greengrass-Core-Gerät verwendet wird, einschließlich des Betriebss ystems.	
`TotalNumberOfFDs`	Die Anzahl der Dateidesk riptoren, die vom Betriebss	

Name	Description
	ystem des Greengrass-Core-Geräts gespeichert werden. Ein Dateideskriptor identifiziert eindeutig eine geöffnete Datei.
Grüngraskern	
`NumberOfComponentsRunning`	Die Anzahl der Komponenten, die auf dem Greengrass-Core-Gerät ausgeführt werden.
`NumberOfComponentsErrored`	Die Anzahl der Komponenten, die sich auf dem Greengras s-Core-Gerät im Fehlerstatus befinden.
`NumberOfComponentsInstalled`	Die Anzahl der Komponenten, die auf dem Greengrass-Core-Gerät installiert sind.
`NumberOfComponentsStarting`	Die Anzahl der Komponenten, die auf dem Greengrass-Core-Gerät gestartet werden.
`NumberOfComponentsNew`	Die Anzahl der Komponenten, die auf dem Greengrass-Core-Gerät neu sind.
`NumberOfComponentsStopping`	Die Anzahl der Komponenten, die auf dem Greengrass-Core-Gerät gestoppt werden.
`NumberOfComponentsFinished`	Die Anzahl der Komponenten, die auf dem Greengrass-Core-Gerät fertiggestellt wurden.

Name	Description	
`NumberOfComponentsBroken`	Die Anzahl der Komponenten, die auf dem Greengrass-Core-Gerät defekt sind.	
`NumberOfComponentsStateless`	Die Anzahl der Komponenten, die auf dem Greengrass-Core-Gerät zustandslos sind.	

Usage

Um Telemetriedaten zur Systemintegrität zu verwenden, können Sie benutzerdefinierte Komponenten erstellen, die die Themen abonnieren, zu denen der Nucleus-Emitter die Telemetriedaten veröffentlicht, und bei Bedarf auf diese Daten reagieren. Da die Nucle-Emitter-Komponente die Möglichkeit bietet, Telemetriedaten zu einem lokalen Thema zu veröffentlichen, können Sie dieses Thema abonnieren und die veröffentlichten Daten verwenden, um lokal auf Ihrem Kerngerät zu agieren. Das Core-Gerät kann dann auf Telemetriedaten reagieren, selbst wenn es nur über eine begrenzte 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 streamen. AWS Cloud Weitere Hinweise zum Erstellen einer solchen Komponente finden Sie unter Lokale Nachrichten veröffentlichen/abonnieren und. Erstellen Sie benutzerdefinierte Komponenten, die den Stream-Manager verwenden

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
1.0.12	Die Version wurde für die Version 2.16.0 von Greengrass Nucleus aktualisiert.
1.0.11	Die Version wurde für die Version 2.15.0 von Greengrass Nucleus aktualisiert.
1.0.10	Die Version wurde für die Version 2.14.0 von Greengrass Nucleus aktualisiert.
1.0.9	Die Version wurde für die Version 2.13.0 von Greengrass Nucleus aktualisiert.
1.0.8	Die Version wurde für die Version 2.12.0 von Greengrass Nucleus aktualisiert.
1.0.7	Die Version wurde für die Version 2.11.0 von Greengrass Nucleus aktualisiert.
1.0.6	Die Version wurde für die Version 2.10.0 von Greengrass Nucleus aktualisiert.
1.0.5	Die Version wurde für die Version 2.9.0 von Greengrass Nucleus aktualisiert.
1.0.4	Die Version wurde für die Version 2.8.0 von Greengrass Nucleus aktualisiert.

Version	Änderungen
1.0.3	Die Version wurde für die Version 2.7.0 von Greengrass Nucleus aktualisiert.
1.0.2	Die Version wurde für die Version 2.6.0 von Greengrass Nucleus aktualisiert.
1.0.1	Die Version wurde für die Version 2.5.0 von Greengrass Nucleus aktualisiert.
1.0.0	Erste Version

PKCS #11 -Anbieter

Mit der PKCS #11 -Anbieterkomponente (`aws.greengrass.crypto.Pkcs11Provider`) können Sie die AWS IoT Greengrass Core-Software so konfigurieren, dass sie über die PKCS #11 -Schnittstelle ein Hardware-Sicherheitsmodul (HSM) verwendet. Diese Komponente ermöglicht es Ihnen, Zertifikate und private Schlüsseldateien sicher zu speichern, sodass sie nicht offengelegt oder in der Software 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 bei der Installation der Core-Software als Provisioning-Plugin angeben. AWS IoT Greengrass 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 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.

Themen

- Versionen
- Typ
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Konfiguration

- Lokale Protokolldatei
- Änderungsprotokoll

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 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 AWS IoT Greengrass Protokolle überwachen.

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-Schlüssel unterstützt.

> **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.

- Eine PKCS #11 -Anbieterbibliothek, die die AWS IoT Greengrass Core-Software zur Laufzeit laden kann (mit libdl), 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 Steckplatz speichern, und sie müssen dieselbe Objektbezeichnung und Objekt-ID verwenden, wenn das HSM Objekt unterstützt. IDs
- Das Zertifikat und der private Schlüssel müssen durch Objektbezeichnungen aufgelöst werden können.
- Der private Schlüssel muss über die folgenden Berechtigungen verfügen:
  - `sign`
  - `decrypt`
- (Optional) Um die Secret Manager-Komponente verwenden zu können, 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-Dienstdatei 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 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.

2.0.11

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.11 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2,5,3 <2,17,0	Weich

2.0.10

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.0.10 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2,5,3 <2,16,0	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	>=2.5.3 <2.15.0	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	>=2.5.3 <2.14.0	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	>=2.5.3 <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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=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	Art der Abhängigkeit
Grüngraskern	>=2,5,3 <2,6,0	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.

`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 Steckplatzes, der den privaten Schlüssel und das Gerätezertifikat enthält. Dieser Wert unterscheidet sich vom Steckplatzindex oder der Steckplatzbezeichnung.

`userPin`

Die Benutzer-PIN, die für den Zugriff auf den Steckplatz verwendet werden soll.

Example Beispiel: Aktualisierung der Zusammenführung von Konfigurationen

```
{
  "name": "softhsm_pkcs11",
  "library": "/usr/lib/softhsm/libsofthsm2.so",
  "slot": 1,
  "userPin": "1234"
}
```

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
2.0.11	Die Version wurde für die Version 2.16.0 von Greengrass Nucleus aktualisiert.
2.0.10	Die Version wurde für die Version 2.15.0 von Greengrass Nucleus aktualisiert.
2.0.9	Die Version wurde für die Version 2.14.0 von Greengrass Nucleus aktualisiert.
2.0.8	Die Version wurde für die Version 2.13.0 von Greengrass Nucleus aktualisiert.
2.0.7	Die Version wurde für die Version 2.12.0 von Greengrass Nucleus aktualisiert.
2.0.6	Die Version wurde für die Version 2.11.0 von Greengrass Nucleus aktualisiert.
2.0.5	Die Version wurde für die Version 2.10.0 von Greengrass Nucleus aktualisiert.

Version	Änderungen
2.0.4	Die Version wurde für die Version 2.9.0 von Greengrass Nucleus aktualisiert.
2.0.3	Die Version wurde für die Version 2.8.0 von Greengrass Nucleus aktualisiert.
2.0.2	Die Version wurde für die Version 2.7.0 von Greengrass Nucleus aktualisiert.
2.0.1	Die Version wurde für die Version 2.6.0 von Greengrass Nucleus 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 den GetSecretValueVorgang in. AWS IoT Device SDK Weitere Informationen erhalten Sie unter Verwenden Sie den AWS IoT Device SDK , um mit dem Greengrass-Kern und anderen Komponenten zu kommunizieren und AWS IoT Core und Geheime Werte abrufen.

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
- Typ
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Konfiguration

- Lokale Protokolldatei
- Änderungsprotokoll

Versionen

Diese Komponente hat die folgenden Versionen:

- 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 AWS IoT Greengrass Protokolle überwachen.

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.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Effect": "Allow",
      "Resource": [
      "arn:aws:secretsmanager:us-east-1:123456789012: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 Geräteverkehr über einen Proxy oder eine Firewall zulassen.

Endpoint	Port	Erforderlich	Description
`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.

2.2.7

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.2.7 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2,5,0 <2,17,0	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	>=2,5,0 <2,16,0	Weich

2.2.2 – 2.2.5

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.2.2 bis 2.2.5 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2,5,0 <2,15,0	Weich

2.2.0

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.2.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.13.0 <2.14.0	Weich

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.

`periodicRefreshIntervalMin` (optional)

Das Intervall in Minuten, in dem diese Komponente die konfigurierten Geheimnisse auf dem Kerngerät mit den neuesten Geheimwerten des AWS Secrets Manager Dienstes synchronisiert.

Wenn dieses Intervall nicht konfiguriert ist, aktualisiert Secret Manager die konfigurierten Geheimnisse nicht regelmäßig.

```
{
  "cloudSecrets": [
    {
      "arn": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyGreengrassSecret-abcdef"
    }
  ],
  "periodicRefreshIntervalMin" : 60
}
```

`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 mit Labels zur Identifizierung der Versionen des Secrets, die auf dem Kerngerät bereitgestellt werden sollen.

Jedes Label muss eine Zeichenfolge sein.

Example Beispiel: Aktualisierung der Konfigurationszusammenführung

```
{
  "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
2.2.7	Die Version wurde für die Version 2.16.0 von Greengrass Nucleus aktualisiert.
2.2.6	Fehlerkorrekturen und Verbesserungen Behebt ein Problem, bei dem der Secret Manager aufgrund eines langsamen oder nicht reagierenden Trusted Platform Module kein Geheimnis erhält. Die Version wurde für die Version 2.15.0 von Greengrass Nucleus aktualisiert.
2.2.5	Fehlerkorrekturen und Verbesserungen Behebt ein Problem, bei dem ein Geheimnis nicht aus dem abgerufen wird, AWS Cloud wenn es nicht im lokalen Cache vorhanden ist.
2.2.4	Fehlerkorrekturen und Verbesserungen Reduziert die Häufigkeit von Schreibvorgängen in den lokalen geheimen Speicher. Secret Manager schreibt jetzt nur noch in den lokalen Speicher, wenn Geheimnisse aktualisiert werden.
2.2.3	Fehlerkorrekturen und Verbesserungen Behebt ein Problem, bei dem der Secret Manager die lokal gespeicherten Geheimnisse löscht, wenn das Kerngerät offline ist und der Sicherheitsdienst des Geräts (z. B. ein HSM) nicht verfügbar ist.

Version	Änderungen
2.2.2	Fehlerkorrekturen und Verbesserungen Behebt ein Problem, bei dem der Secret Manager die mit teilweisen ARNS konfigurierten Geheimnisse nicht herunterlädt.
2.2.1	Fehlerkorrekturen und Verbesserungen Unterstützt Secret Manager auf den Nucleus-Versionen 2.5.0 und höher.
2.2.0	Neue Features Integriert die Unterstützung für die regelmäßige Aktualisierung der konfigurierten Secrets über einen neuen Komponentenkonfigurationssc hlüssel. Fügt Unterstützung für einen neuen Anforderungsparameter in der `GetSecretValue` IPC-Anfrage hinzu, um die Geheimnisse pro Anfrage zu aktualisieren
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 aktualisi ert.

Version	Änderungen
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. 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 China 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
- Typ
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Konfiguration
- Lokale Protokolldatei
- Lizenzen
- Usage
- Weitere Informationen finden Sie auch unter
- Änderungsprotokoll

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.

Architekturen:

- Armv71
- Arm 8 () 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 gilt nicht für 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 gilt nicht für 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 den 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 Geräteverkehr über einen Proxy oder eine Firewall zulassen.

Endpoint	Port	Erforderlich	Description
`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 – 1.1.3

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 1.0.19 bis 1.1.3 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

Usage

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
1.1.3	Fehlerkorrekturen und Verbesserungen • Führt ein Upgrade des zugrunde liegenden AWS IoT Geräteclients, der von der Komponente aufgerufen wird, von Version 1.10.0 auf Version 1.10.1 durch. • Behebt die Kompatibilitätsprobleme der GNU C-Bibliothek (Glibc) in Version 1.1.2 der Secure Tunneling-Komponente.
1.1.2	⚠ Warning Diese Version ist nicht mehr verfügbar. Die Verbesserungen in dieser Version sind in späteren Versionen dieser Komponente verfügbar. Fehlerkorrekturen und Verbesserungen • Führt ein Upgrade des zugrunde liegenden AWS IoT Geräteclients, der von der Komponente aufgerufen wird, von Version 1.9.0 auf Version 1.10.0 durch. • Behebt das Problem mit der Payload-Übertragung, das Benutzer daran hindert, große Dateien von Greengrass V2-Kerngeräten über den sicheren Tunnel an das Quellgerät weiterzuleiten.
1.1.1	Fehlerkorrekturen und Verbesserungen • Fügt eine Konfiguration zur Unterstützung von Greengrass Nucleus Lite hinzu.
1.1.0	Neue Features • Fügt Rezeptunterstützung für Greengrass Nucleus Lite hinzu.
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. • Erhöht das Limit für gleichzeitige Tunnel auf Komponentenebene auf 20 Tunnel.

Version	Änderungen
	• 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.
1.0.12	Fehlerkorrekturen und Verbesserungen • Fügt Unterstützung für x86_64 (AMD64) und ARMv8 (Aarch64) hinzu, wenn sie unter Raspberry Pi OS laufen.

Version	Änderungen
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 unterbrochen 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 unterInteragiere 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 Core-Gerät ändern.

Diese Komponente verwendet dieselbe Protokolldatei wie der Greengrass-Kern. Weitere Informationen finden Sie unter AWS IoT Greengrass Protokolle überwachen.

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 V2 Kerngeräte
- 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.

2.3.12

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.3.12 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2,5,0 <2,17,0	Weich

2.3.11

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.3.11 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2,5,0 <2,16,0	Weich

2.3.10

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.3.10 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2,5,0 <2,15,0	Weich

2.3.9

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.3.9 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2,5,0 <2,14,0	Weich

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, die diese Komponente verwendet, um Schatten zwischen AWS IoT Core und dem Kerngerät zu synchronisieren.

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 die *`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-Serviceanfragen 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 requests/second 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, die diese Komponente verwendet, um Schatten zwischen AWS IoT Core und dem Kerngerät zu synchronisieren.

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 die *`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-Serviceanfragen 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 requests/second 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, die diese Komponente verwendet, um Schatten zwischen AWS IoT Core und dem Kerngerät zu synchronisieren.

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 die *`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 requests/second 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 die *`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 requests/second 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
2.3.12	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem Bereitstellungen blockiert wurden, wenn mehr als 1024 Geräteschatten konfiguriert wurden.
2.3.11	Die Version wurde für die Version 2.15.0 von Greengrass Nucleus aktualisiert.
2.3.10	Die Version wurde für die Version 2.14.0 von Greengrass Nucleus aktualisiert.
2.3.9	Die Version wurde für die Version 2.13.0 von Greengrass Nucleus aktualisiert.
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 Schattenzuständen.
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.

Version	Änderungen
	• 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.
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 • Fügt Unterstützung für den lokalen Shadow-Service über die lokale publish/subscribe Schnittstelle hinzu. Sie können jetzt mit dem lokalen publish/subscribe Message Broker über Shadow-MQTT-Themen kommunizieren, um Shadows auf dem Kerngerät abzurufen, zu aktualisieren und zu löschen. Mit dieser Funktion können Sie Client-Geräte mit dem lokalen Shadow-Dienst 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 Schattenmanager 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 zusammengeführt 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 Developer Guide.

Um mit dieser Komponente in einem Amazon SNS SNS-Thema zu veröffentlichen, veröffentlichen Sie eine Nachricht zu dem Thema, das diese Komponente abonniert. Standardmäßig abonniert diese Komponente das `sns/message` lokale Thema „Veröffentlichen/Abonnieren“. Sie können andere Themen angeben, einschließlich AWS IoT Core MQTT-Themen, wenn Sie diese Komponente bereitstellen.

In Ihrer benutzerdefinierten Komponente möchten Sie möglicherweise eine 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 in einer einzigen Komponente zentralisieren.

> **Note**
>
> Diese Komponente bietet ähnliche Funktionen wie der Amazon SNS SNS-Connector in AWS IoT Greengrass Version 1. Weitere Informationen finden Sie unter Amazon SNS 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:

- 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 mithilfe der Lambda-Launcher-Komponente aus.

Weitere Informationen finden Sie unter Komponententypen.

Betriebssystem

Diese Komponente kann nur auf Linux-Kerngeräten installiert werden.

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.
- Python-Version 3.7 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH 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 im folgenden Beispiel für eine IAM-Richtlinie gezeigt.

  JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "sns:Publish"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:sns:us-east-1:123456789012: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 bei der Bereitstellung dieser Komponente das folgende Konfigurationsupdate für die ältere Abonnement-Router-Komponente (`aws.greengrass.LegacySubscriptionRouter`) zusammenführen. Diese Konfiguration gibt das Thema an, zu 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"
      }
    }
  }
  ```

  - *region*Ersetzen Sie es durch AWS-Region das, was Sie verwenden.
  - *version*Ersetzen Sie durch die Version der Lambda-Funktion, die diese Komponente ausführt. Um die Version der Lambda-Funktion zu finden, müssen Sie sich das Rezept für

die Version dieser Komponente ansehen, 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 Komponentenversion verwenden, die Sie bereitstellen.

Weitere Informationen finden Sie unter Erstellen von Bereitstellungen.

- Die Amazon SNS SNS-Komponente wird für die Ausführung in einer VPC unterstützt. Um diese Komponente in einer VPC bereitzustellen, ist Folgendes erforderlich.
  - Die Amazon SNS SNS-Komponente muss über Konnektivität verfügen`sns.region.amazonaws.com`, zu der der VPC-Endpunkt ist. `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 Endpunkten und Ports, die für den Basisbetrieb erforderlich sind. Weitere Informationen finden Sie unter Geräteverkehr über einen Proxy oder eine Firewall zulassen.

Endpoint	Port	Erforderlich	Description
`sns.region.amazonaws.com`	443	Ja	Veröffentlichen Sie Nachrichten in Amazon SNS.

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.

2.1.10

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 <2.16.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^2.0.0	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 <2.15.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^2.0.0	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 <2.14.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.13.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.12.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.11.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.10.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.9.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.8.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.7.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.6.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.5.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.4.0	Hart
Lambda-Launcher	^2.0.0	Hart

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.3.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.2.0	Hart
Lambda-Launcher	^2.0.0	Hart
Lambda-Laufzeiten	^2,0.0	Weich
Token-Austauschdienst	^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	Art der Abhängigkeit
Grüngraskern	>=2.0.3 <2.1.0	Hart
Lambda-Launcher	>=1.0.0	Hart
Lambda-Laufzeiten	>=1,0.0	Weich
Token-Austauschdienst	>=1.0.0	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**
>
> Die Standardkonfiguration dieser Komponente umfasst Lambda-Funktionsparameter. Wir empfehlen, dass Sie nur die folgenden Parameter 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 Amazon SNS SNS-Standardthemas, 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 läuft in einer isolierten Laufzeitumgebung innerhalb des AWS IoT Greengrass Containers.

Standard: `GreengrassContainer`

`containerParams`

(Optional) Ein Objekt, das die Container-Parameter 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 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 Themen abonniert. publish/subscribe

Dieses Objekt enthält die folgenden Informationen:

`0`— Dies ist ein Array-Index als Zeichenfolge.

Ein Objekt, das die folgenden Informationen enthält:

`type`

(Optional) Der publish/subscribe Messaging-Typ, den diese Komponente verwendet, um Nachrichten zu abonnieren. 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 aus, 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 aus 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 Container-Modus)

```
{
  "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 SNS-Zielthema. Standardmäßig abonniert diese Komponente lokale

publish/subscribe Nachrichten. Weitere Informationen zum Veröffentlichen von Nachrichten aus Ihren benutzerdefinierten Komponenten in dieser Komponente finden Sie unterLokale Nachrichten veröffentlichen/abonnieren.

Standardthema (lokales Veröffentlichen/Abonnieren): `sns/message`

Die Nachricht akzeptiert die folgenden Eigenschaften. Eingabenachrichten müssen im JSON-Format vorliegen.

`request`

Die Informationen zu der Nachricht, die an das Amazon SNS SNS-Thema gesendet werden soll.

Typ: der `object` 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 es `json` für die `message_structure` Eigenschaft an.

Typ: `string`

`subject`

(Optional) Der Betreff der Nachricht.

Typ: `string`

Der Betreff kann aus ASCII-Text und bis zu 100 Zeichen bestehen. Er muss mit einem Buchstaben, einer Zahl oder einem Satzzeichen beginnen. Es darf keine Zeilenumbrüche oder Steuerzeichen enthalten.

`sns_topic_arn`

(Optional) Der ARN des Amazon SNS SNS-Themas, in dem diese Komponente die Nachricht veröffentlicht. Geben Sie diese Eigenschaft an, um das standardmäßige Amazon SNS SNS-Thema zu überschreiben.

Typ: `string`

`message_structure`

(Optional) Die Struktur der Nachricht. Geben Sie `json` in der `content` Eigenschaft an, dass eine JSON-Nachricht gesendet werden soll, die Sie als Zeichenfolge 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, setzt die Komponente die `id` Eigenschaft im Antwortobjekt auf diesen Wert.

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 standardmäßig Antworten als Ausgabedaten zum folgenden MQTT-Thema. Sie müssen dieses Thema `subject` in der Konfiguration für die ältere Abonnement-Router-Komponente angeben. Weitere Informationen zum Abonnieren von Nachrichten zu diesem Thema in Ihren benutzerdefinierten Komponenten finden Sie unterMQTT-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
```

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.greengrass.SNS.log
  ```

Lizenzen

Diese Komponente umfasst die folgende Software/Lizenzierung von Drittanbietern:

- AWS SDK für Python (Boto3)/Apache-Lizenz 2.0
- botocore/Apache-Lizenz 2.0
- dateutil/PSF-Lizenz
- docutils/BSD-Lizenz, GNU General Public License (GPL), Python Software Foundation License, Public Domain
- jmespath/MIT-Lizenz
- s3transfer/Apache-Lizenz 2.0
- urllib3/MIT-Lizenz

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
2.1.10	Die Version wurde für die Version 2.15.0 von Greengrass Nucleus aktualisiert.
2.1.9	Die Version wurde für die Version 2.14.0 von Greengrass Nucleus aktualisiert.
2.1.8	Die Version wurde für die Version 2.13.0 von Greengrass Nucleus aktualisiert.

Version	Änderungen
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	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 HTTPS-Netzwerk-Proxykonfigurationen hinzu. Weitere Informationen erhalten Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy und Ermöglichen Sie dem Kerngerät, einem HTTPS-Proxy zu vertrauen.
2.0.8	Die Version wurde für die Version 2.5.0 von Greengrass Nucleus aktualisiert.
2.0.7	Die Version wurde für die Version 2.4.0 von Greengrass Nucleus aktualisiert.
2.0.6	Die Version wurde für die Version 2.3.0 von Greengrass Nucleus aktualisiert.
2.0.5	Die Version wurde für die Version 2.2.0 von Greengrass Nucleus aktualisiert.
2.0.4	Die Version wurde für die Version 2.1.0 von Greengrass Nucleus aktualisiert.
2.0.3	Erste Version

Stream-Manager

Die Stream-Manager-Komponente (`aws.greengrass.StreamManager`) ermöglicht es Ihnen, Datenströme zu verarbeiten, um sie AWS Cloud von Greengrass-Kerngeräten an die zu übertragen.

Weitere Informationen zur Konfiguration und Verwendung des Stream-Managers in benutzerdefinierten Komponenten finden Sie unterDatenströme auf Greengrass-Kerngeräten verwalten.

Themen

- Versionen
- Typ
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Konfiguration
- Lokale Protokolldatei
- Änderungsprotokoll

Versionen

Diese Komponente hat die folgenden Versionen:

- 2.2.x
- 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 Kerngeräten 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 Stream Manager verwenden. Weitere Informationen finden Sie unter:
  - the section called “AWS IoT Analytics Kanäle”
  - the section called “Amazon Kinesis Kinesis-Datenströme”
  - the section called “AWS IoT SiteWise Eigenschaften von Vermögenswerten”
  - the section called “Amazon-S3-Objekte”
- 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 mit dem AWS Service verbunden sein, für den 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 Endpunkt der Amazon S3 S3-VPC-Schnittstelle verfügbar. Weitere Informationen finden Sie unter Einschränkungen und Einschränkungen von AWS PrivateLink für Amazon S3. Um dieses Problem zu lösen, können Sie aus den folgenden Optionen wählen.
    - Konfigurieren Sie die Stream Manager-Komponente so, dass sie den regionalen S3-Endpunkt in der `us-east-1` Region verwendet, indem Sie sie `-Daws.s3UseUsEast1RegionalEndpoint=regional` in einrichten`JVM_ARGS`.
    - Erstellen Sie einen Amazon S3 S3-Gateway-VPC-Endpunkt anstelle eines VPC-Endpunkts mit Amazon S3 S3-Schnittstelle. S3-Gateway-Endpunkte unterstützen den Zugriff auf den

globalen S3-Endpunkt. Weitere Informationen finden Sie unter Einen Gateway-Endpunkt erstellen.

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 Geräteverkehr über einen Proxy oder eine Firewall zulassen.

Endpoint	Port	Erforderlich	Description
`iotanalytics.` *`region`*`.amazonaws.com`	443	Nein	Erforderlich, wenn Sie Daten veröffentlichen in. 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 veröffentlichen für AWS IoT SiteWise.
`*.s3.amazonaws.com`	443	Nein	Erforderlich, wenn Sie Daten

Endpoint	Port	Erforderlich	Description
			in S3-Buckets veröffentlichen. Sie können es * 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 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.

2.1.3 – 2.2.1

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.1.3 bis 2.2.1 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <3.0.0	Weich
Token-Austauschdienst	>=0.0.0	Hart

2.1.11 – 2.1.12

In der folgenden Tabelle sind die Abhängigkeiten für die Versionen 2.1.11 bis 2.1.10 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.13.0	Weich
Token-Austauschdienst	>=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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.12.0	Weich
Token-Austauschdienst	>=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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.11.0	Weich

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Token-Austauschdienst	>=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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.10.0	Weich
Token-Austauschdienst	>=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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.9.0	Weich
Token-Austauschdienst	>=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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.8.0	Weich
Token-Austauschdienst	>=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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.7.0	Weich
Token-Austauschdienst	>=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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.6.0	Weich
Token-Austauschdienst	>=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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.5.0	Weich
Token-Austauschdienst	>=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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.4.0	Weich
Token-Austauschdienst	>=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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.3.0	Weich
Token-Austauschdienst	>=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	Art der Abhängigkeit
Grüngraskern	>=2.0.0 <2.2.0	Weich
Token-Austauschdienst	>=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	Art der Abhängigkeit
Grüngraskern	>=2.0.3 <2.1.0	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.

`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 Stream-Manager-Stammordner angeben. Mit diesen Befehlen kann der Standardsystembenutzer,`ggc_user`, in 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`, dass ein zufälliger verfügbarer Port verwendet werden soll.

Standard: `8088`

`STREAM_MANAGER_AUTHENTICATE_CLIENT`

(Optional) Sie können festlegen, dass sich Clients authentifizieren müssen, 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`, erlaubt das Stream Manager SDK nur Greengrass-Komponenten als Clients.

Wenn Sie dies angeben`false`, lässt das Stream Manager SDK zu, dass alle Prozesse auf dem Kerngerät Clients sind.

Standard: `true`

`STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH`

(Optional) Die durchschnittliche maximale Bandbreite (in Kilobit pro Sekunde), die Stream Manager zum Exportieren von Daten verwenden kann.

Standard: keine Einschränkung

`STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE`

(Optional) Die maximale Anzahl aktiver Threads, die der 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 beschränkten 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 auf Amazon S3. Stream Manager verwendet diese Einstellung und die Größe der Eingabedatei, um zu bestimmen, wie Daten in einer mehrteiligen PUT-Anfrage gebündelt werden.

> **Note**
>
> Stream Manager verwendet die `sizeThresholdForMultipartUploadBytes` Streams-Eigenschaft, um zu bestimmen, ob der Export nach Amazon S3 als einteiliger

oder mehrteiliger Upload erfolgen 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, die hier in der Reihenfolge der Stufen 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.

`startupTimeoutSeconds`

(Optional) Die maximale Zeit in Sekunden für den Start der Komponente. Der Status der Komponente ändert sich auf, `ERRORED` wenn dieser Timeout überschritten wird.

Standard: `120`

Example Beispiel: Aktualisierung der Zusammenführung von Konfigurationen

In der folgenden Beispielkonfiguration wird angegeben, 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
```

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.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 den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
2.2.1	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem der Stream-Manager keine Nachrichten an Kinesis Data Streams Streams-Ziele exportiert. • Zusätzliche Verbesserungen der Leistung der Stream Manager-Exporte zu Kinesis Data Streams Streams-Zielen.
2.2.0	Neue Features • Fügt einen neuen Konfigurationsschlüssel für das Start-Timeout hinzu. Der Standardwert ist 120 Sekunden. • Fügt Rezeptunterstützung für Greengrass Nucleus Lite hinzu.
2.1.13	Fehlerkorrekturen und Verbesserungen Unterstützt FIPS-Endpunkte für AWS IoT SiteWise
2.1.12	Fehlerkorrekturen und Verbesserungen Aktualisiert die Reihenfolge, in der Anmeldeinformationen verwendet werden, sodass Greengrass-Anmeldeinformationen für AWS Servicean fragen bevorzugt werden.
2.1.11	Die Version wurde für die Version 2.12.0 von Greengrass Nucleus aktualisiert.
2.1.10	Fehlerkorrekturen und Verbesserungen Behebt ein Problem, bei dem die HTTPS-Proxykonfiguration der Zertifika tskette der Greengrass Certificate Authority (CA) nicht vertraut.
2.1.9	Die Version wurde für die Version 2.11.0 von Greengrass Nucleus aktualisiert.
2.1.8	Fehlerkorrekturen und Verbesserungen Behebt ein Problem, bei dem der Stream-Manager unendlich wiederholt, dass SiteWise Exporte fehlschlagen mit. `InvalidRequestException`

Version	Änderungen
2.1.7	Fehlerkorrekturen und Verbesserungen Behebt ein Problem, bei dem der Stream-Manager die Proxykonfiguration nicht korrekt lesen kann.
2.1.6	Fehlerkorrekturen und Verbesserungen Behebt ein Problem, das bei bestimmten ARMv8 Prozessoren, einschlie ßlich des Jetson Nano, zu einem Absturz beim Start führen konnte.
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 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. • Aktualisiert das Standard-Verbindungs-Timeout von 3 Sekunden auf 1 Minute.
2.1.3	Fehlerkorrekturen und Verbesserungen Behebt ein Startproblem unter Windows OS, wenn es als SYSTEM-Be nutzer ausgeführt wird.
2.1.2	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem unter Windows-Betriebssystemen, die eine andere Sprache als Englisch verwenden. • Die Version wurde für die Version 2.9.0 von Greengrass Nucleus aktualisiert.
2.1.1	Die Version wurde für die Version 2.8.0 von Greengrass Nucleus aktualisiert.

Version	Änderungen
2.1.0	Neue Features • Aktualisiert diese Komponente, sodass automatisch Telemetriedaten an Amazon EventBridge gesendet werden. Weitere Informationen finden Sie unter Erfassung von Telemetriedaten zur Systemintegrität von AWS IoT Greengrass Kerngeräten. Für diese Funktion ist Version 2.7.0 oder höher der Greengrass Nucleus-Komponente erforderlich. • Die Version wurde für die Version 2.7.0 von Greengrass Nucleus aktualisiert.
2.0.15	Die Version wurde für die Version 2.6.0 von Greengrass Nucleus aktualisiert.
2.0.14	Diese Version enthält Fehlerkorrekturen und Verbesserungen.
2.0.13	Die Version wurde für die Version 2.5.0 von Greengrass Nucleus aktualisiert.
2.0.12	Fehlerkorrekturen und Verbesserungen Behebt ein Problem, das ein Upgrade von Stream Manager v2.0.7 auf eine Version zwischen v2.0.8 und v2.0.11 verhinderte. Wenn Sie den Stream Manager verwenden, um Daten in die Cloud zu exportieren, können Sie jetzt ein Upgrade auf Version 2.0.12 durchführen.
2.0.11	Die Version wurde für die Version 2.4.0 von Greengrass Nucleus aktualisiert.
2.0.10	Die Version wurde für die Version 2.3.0 von Greengrass Nucleus aktualisiert.
2.0.9	Die Version wurde für die Version 2.2.0 von Greengrass Nucleus aktualisiert.
2.0.8	Die Version wurde für die Version 2.1.0 von Greengrass Nucleus aktualisiert.
2.0.7	Erste Version

Systemprotokollweiterleitung

Der System Log Forwarder (`aws.greengrass.SystemLogForwarder`) lädt aktive Systemprotokolle CloudWatch mithilfe der CloudWatch HTTPS-API direkt zu Amazon hoch.

> **Important**
>
> Diese Komponente leitet nur Systemd-Journald-Protokolle weiter, die während der Laufzeit generiert wurden. Weitere Informationen zu Systemd-Journald-Protokollen finden Sie unter systemd-journald und journalctl.

> **Note**
>
> Diese Komponente benötigt spezielle Berechtigungen, um Protokollgruppen und Streams zu erstellen und zu verwalten. CloudWatch

Themen

- Versionen
- Typ
- Betriebssystem
- Voraussetzungen
- Endpunkte und Ports
- Abhängigkeiten
- Konfiguration
- 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 muss auf systemd-basierten Linux-Systemen installiert werden.

Voraussetzungen

Für diese Komponente gelten die folgenden Anforderungen:

Die Komponente benötigt Zugriff zum Erstellen von Log- und Stream-Gruppen CloudWatch sowie die Erlaubnis, den PutLogEvents HTTP-Aufruf durchzuführen. Sie müssen dem Rollenalias Ihres Greengrass-Geräts mindestens die folgenden Richtlinienberechtigungen hinzufügen:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
    {
      "Effect": "Allow",
      "Action": ["logs:CreateLogGroup"],
      "Resource": "arn:aws:logs:us-east-1:111122223333:log-group:greengrass/
systemLogs:*"
    },
    {
      "Effect": "Allow",
      "Action": ["logs:CreateLogStream", "logs:PutLogEvents"],
      "Resource": "arn:aws:logs:us-east-1:111122223333:log-group:greengrass/
systemLogs:log-stream:${credentials-iot:ThingName}"
    }
  ]
}
```

> **Note**
>
> Weitere Informationen finden Sie auf der Github-Seite von System Log Forwarder.

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 Geräteverkehr über einen Proxy oder eine Firewall zulassen.

Endpoint	Port	Erforderlich	Description
`logs.region.amazonaws.com`	443	Nein	Erforderlich, wenn Sie Protokolle in Logs schreiben. CloudWatch

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.x

In der folgenden Tabelle sind die Abhängigkeiten für Version 2.1.x dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Token-Austauschdienst	>=2.0.0	Hart
Greengrass Nucleus Lite	>=2.3.0	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
Token-Austauschdienst	>=2.0.0	Hart

Konfiguration

Diese Komponente stellt die folgenden Konfigurationsparameter bereit, die Sie bei der Bereitstellung der Komponente anpassen können.

2.0.x-2.1.x

`maxUploadIntervalSec`

Der maximale Zeitraum, in dem der System-Log-Forwarder versucht, Protokolle hochzuladen. Da die Protokollweiterleitung Protokolle hochlädt, wenn der Speicher voll ist, kann es dennoch vorkommen, dass sie öfter als die konfigurierte maximale Schrittfrequenz hochlädt.

`maxRetriesCount`

Gibt an, wie oft der System-Log-Forwarder versucht, einen vorübergehenden HTTP-Fehler zu wiederholen.

`bufferCapacity`

Die Größe des Ringpuffers für die Speicherung von Protokollen im Arbeitsspeicher.

`logGroup`

Der Protokollpfad in CloudWatch.

`logStream`

Der CloudWatch LogStream.

`filters`

Eine Übersicht der Filterkonfigurationen für das Kerngerät.

`services`

Eine Liste von Dienstnamenfiltern, anhand derer System Log Forwarder bestimmt, welche Protokolle hochgeladen werden. Ein Protokoll wird nur hochgeladen, wenn der Dienst, von dem es stammt, mindestens einem der Filter in dieser Liste entspricht. Bei den Filtern in dieser Liste kann es sich entweder um eine Zeichenfolge handeln, der der Dienstname vollständig entsprechen muss, oder um eine Zeichenfolge, die mit * endet, was bedeutet, dass das Präfix übereinstimmen muss.

Standard: `[ggl.*]`

> **Important**
>
> Ein Protokoll wird nur hochgeladen, wenn der Dienst, von dem es stammt, mindestens einem der Filter in dieser Liste entspricht.

> **Note**
>
> Wenn Sie den Wert * verwenden, werden alle verfügbaren Dienste berücksichtigt.

Example Beispielkonfiguration:

Das folgende Beispiel filtert Logs nach allen Diensten, die in Greengrass Nucleus Lite enthalten sind.

```
{
    "maxUploadIntervalSec": 300,
    "maxRetriesCount": 3,
    "bufferCapacity": 1048576,
    "logGroup": "greengrass/systemLogs",
    "logStream": "deviceName",
```

```
    "filters": {
        "services": ["ggl.*"]
    }
}
```

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
2.1.0	Fehlerkorrekturen und Verbesserungen • Aktualisiert die Rezeptur der Komponenten, um den Greengrass-Kern ordnungsgemäß zu unterstützen. • Die Protokollausgabe für den Fall, dass keine Protokolle zum Hochladen vorhanden sind, wurde verbessert. • Allgemeine Fehlerbehebungen und Verbesserungen.
2.0.1	Fehlerkorrekturen und Verbesserungen Aktualisiert das Komponentenrezept, sodass es Aarch64-Systeme (arm64) ordnungsgemäß unterstützt.
2.0.0	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 EC2 Amazon-Instances, lokalen Servern und virtuellen Maschinen (VMs) sowie Edge-Geräten. Systems Manager ermöglicht es Ihnen, Betriebsdaten einzusehen, Betriebsaufgaben zu automatisieren und Sicherheit und Compliance aufrechtzuerhalten. 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
- Typ
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Konfiguration
- Lokale Protokolldatei
- Weitere Informationen finden Sie auch unter
- Änderungsprotokoll

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 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 Amazon SSMManaged InstanceCore verwaltete Richtlinie oder eine benutzerdefinierte Richtlinie enthalten, die entsprechende Berechtigungen definiert. Weitere Informationen finden Sie unter Erstellen einer IAM-Servicerolle für Edge-Geräte im AWS Systems Manager Benutzerhandbuch.

  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 Geräteverkehr über einen Proxy oder eine Firewall zulassen.

Endpoint	Port	Erforderlich	Description
`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 Sie mit Session Manager, einer Funktion

Endpoint	Port	Erforderlich	Description
			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 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 1.0.0 bis 1.3.0 dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Token-Austauschdienst	>=2.0.0 <3.0.0	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.

`SSMRegistrationRole`

Die IAM-Servicerolle, die Systems Manager übernehmen kann und die die von Amazon SSMManaged InstanceCore verwaltete Richtlinie oder eine benutzerdefinierte Richtlinie umfasst, die entsprechende Berechtigungen definiert. Weitere Informationen finden Sie im AWS Systems Manager Benutzerhandbuch unter Erstellen einer IAM-Servicerolle für Edge-Geräte.

`SSMOverrideExistingRegistration`

(Optional) Wenn auf dem Kerngerät bereits der Systems Manager Agent ausgeführt wird, der mit einer Hybrid-Aktivierung registriert ist, 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 EC2 Amazon-Instance läuft, auf der der Systems Manager Agent installiert und eine Instance-Profilrolle konfiguriert ist, beginnt die bestehende verwaltete Knoten-ID der EC2 Amazon-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 es 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 Sie Greengrass-Kerngeräte 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
1.3.0	Fehlerkorrekturen und Verbesserungen Aktualisiert diese Komponente auf die Agent-Version 3.3.1142.0.
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-Austauschdienst

Die Token-Exchange-Dienstkomponente (`aws.greengrass.TokenExchangeService`) stellt AWS Anmeldeinformationen bereit, mit denen Sie mit AWS Diensten in Ihren benutzerdefinierten Komponenten interagieren können.

Der Token-Austauschdienst wird als lokaler Server ausgeführt. Dieser lokale Server stellt über den AWS IoT Rollenalias, den Sie in der Greengrass Core Nucleus-Komponente konfigurieren, eine Verbindung zum Anbieter für AWS IoT Anmeldeinformationen her. Die Komponente stellt zwei Umgebungsvariablen bereit, `AWS_CONTAINER_CREDENTIALS_FULL_URI` und`AWS_CONTAINER_AUTHORIZATION_TOKEN`. `AWS_CONTAINER_CREDENTIALS_FULL_URI`definiert den URI zu diesem lokalen Server. Wenn eine Komponente einen AWS SDK-Client erstellt, erkennt der Client diese URI-Umgebungsvariable und verwendet das darin enthaltene Token, `AWS_CONTAINER_AUTHORIZATION_TOKEN` um eine Verbindung zum Token-Austauschdienst herzustellen und AWS Anmeldeinformationen abzurufen. Auf diese Weise können Greengrass-Kerngeräte den AWS Servicebetrieb anrufen. Weitere Informationen zur Verwendung dieser Komponente in benutzerdefinierten Komponenten finden Sie unterInteragieren Sie mit AWS Diensten.

> **Important**
>
> Die Support für den Erwerb von AWS Anmeldeinformationen auf diese Weise wurde 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 Developer Guide.

> **Note**
>
> Komponenten, die die AWS Standard-Credential-Kette verwenden, können andere Quellen von Anmeldeinformationen erkennen und verwenden, sofern sie existieren. Innerhalb des Amazon Amazon Elastic Container Service (Amazon ECS) haben Container-Anmeldeinformationen `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` Vorrang vor dem`AWS_CONTAINER_CREDENTIALS_FULL_URI`, was bedeutet, dass Anmeldeinformationen für den Token-Austausch-Service innerhalb von Amazon Elastic Container Service (Amazon ECS) möglicherweise nicht funktionieren. Weitere Informationen finden Sie unter Anbieter von standardisierten Anmeldeinformationen

Themen

- Versionen
- Typ
- Betriebssystem
- Abhängigkeiten
- Konfiguration
- Lokale Protokolldatei
- Änderungsprotokoll

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 Kerngeräten installiert werden, auf denen die folgenden Betriebssysteme ausgeführt werden:

- Linux
- Windows

Abhängigkeiten

Diese Komponente hat keine Abhängigkeiten.

Konfiguration

Diese Komponente stellt die folgenden Konfigurationsparameter bereit, die Sie bei der Bereitstellung der Komponente anpassen können.

`port`

Der Port, der für Verbindungen mit dem Token-Exchange-Dienst verwendet werden soll. Der Token-Austauschdienst wird neu gestartet, nachdem sich die Portkonfiguration geändert hat.

`credentialRetryInSec`

Gibt die Wiederholungsintervalle in Sekunden an, wenn der Token-Exchange-Dienst auf Fehler bei der Anforderung von Anmeldeinformationen stößt.

`clientError`

Das Wiederholungsintervall in Sekunden für Clientfehler (4xx-HTTP-Statuscodes).

Standard: `120`

Gültige Werte: `10` bis `42900`

`serverError`

Das Wiederholungsintervall in Sekunden für Serverfehler (5xx HTTP-Statuscodes).

Standard: `60`

Gültige Werte: `10` bis `42900`

`unknownError`

Das Wiederholungsintervall in Sekunden für unbekannte Fehler (Verbindungsfehler und HTTP-Statuscodes außerhalb der Bereiche 4xx und 5xx).

Standard: `300`

Gültige Werte: `10` bis `42900`

Example Beispiel: Aktualisierung der Zusammenführung von Konfigurationen

```
{
   "port": 2020,
   "credentialRetryInSec": {
    "clientError": 30,
    "serverError": 45,
    "unknownError": 60
   }
}
```

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
2.0.3	Erste Version

SiteWise IoT-OPC-UA-Kollektor

Die SiteWise IoT-OPC-UA-Sammlerkomponente (`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 Using AWS IoT SiteWise at the Edge im AWS IoT SiteWise Benutzerhandbuch.

Themen

- Versionen
- Typ
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Konfiguration
- Eingabedaten

- Ausgabedaten
- Lokale Protokolldatei
- Lizenzen
- Änderungsprotokoll
- Weitere Informationen finden Sie auch unter

Versionen

Diese Komponente hat die folgenden Versionen:

- 3.1.x
- 3.0.x
- 2.6.x
- 2.5.x
- 2.4.x
- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

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 höher

    Architektur: x86_64 () oder (Aarch64) AMD64 ARMv8
  - Betriebssystem: Red Hat Enterprise Linux (RHEL) 8

    Architektur: x86_64 (AMD64) oder (Aarch64) ARMv8
  - 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 später

    Architektur: x86_64 () AMD64
- Das Greengrass-Core-Gerät muss ausgehende Netzwerkkonnektivität zu OPC UA-Servern ermöglichen.

Abhängigkeiten

Wenn Sie eine Komponente bereitstellen, stellt es 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 alle Versionen 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
Geheimer Manager	>2,0,10 =2,0,8 <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 IoT SiteWise OPC UA-Collector-Komponente zu konfigurieren. Weitere Informationen finden Sie unter Schritt 4: Datenquellen hinzufügen — optional im AWS IoT SiteWise Benutzerhandbuch.

Eingabedaten

Diese Komponente akzeptiert nur Daten in den folgenden Formaten, alle anderen werden ignoriert und verworfen. In der folgenden Tabelle werden die OPC UA-Datentypen ihren entsprechenden Datentypen zugeordnet. SiteWise

SiteWise Datentyp	OPC UA-Datentyp	Beschreibung

`STRING`	`String`
	`Guid`
	`XmlElement`
		Eine Zeichenfolge mit einer maximalen Länge von 1024 Byte.

`INTEGER`	`SByte`
	`Byte`
	`Int16`
	`UInt16`
		Eine 32-Bit-Ganzzahl mit Vorzeichen und einem Bereich von `-2,147,483,648 to 2,147,483,647` .

SiteWise Datentyp	OPC UA-Datentyp	Beschreibung

	`Int32`
	`UInt32*`
	`Int64*`	

`DOUBLE`	`UInt32*`
	`Int64*`
	`Float`
	`Double`
		Eine Fließkommazahl mit einem Bereich von `-10^100 to 10^100` und `IEEE 754` doppelter Genauigkeit.

`BOOLEAN`	`Boolean`	`true` oder `false`.

* Bei OPC UA-Datentypen `UInt32` und ist der SiteWise Datentyp`Int64`, `INTEGER` wenn er in der Lage SiteWise ist, seinen Wert darzustellen, andernfalls ist es der Fall. `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
```

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.SiteWiseEdgeCollectorOpcua.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeCollectorOpcua.log -Tail
   10 -Wait
  ```

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
3.1.0	Neue Features • Fügt die Konfigurierbarkeit des Knotenerkennungsprozesses für jede Datenquelle hinzu. • Fügt die Konfigurierbarkeit des Standardauslösers für Datenänderungen für jede Datenquelle hinzu. Fehlerkorrekturen und Verbesserungen • Behebt Sicherheitslücken.

Version	Änderungen
3.0.3	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, das beim Synchronisieren großer Konfigurationen zu Dateikürzungen und Synchronisierungsfehlern führte. • Behebt ein Problem, das aufgrund der AWS IoT Core Abonnementlat enz zu Verzögerungen beim Startvorgang führte. • Behebt ein Problem, das bei kurzzeitigen Verbindungsabbrüchen von OPC UA-Servern die vollständige Erkennung auslöste.
3.0.2	Fehlerkorrekturen und Verbesserungen • Korrigiert die `OpcUaCollector.IncomingValuesCount` Metrik, sodass Snapshot-Punkte korrekt in Berechnungen berücksichtigt werden.
3.0.1	Fehlerkorrekturen und Verbesserungen • Behebt Sicherheitslücken. • Behebt ein Problem mit der Verarbeitung von OPC UA-Zeitstempeln. Fügt eine bevorzugte Zeitstempelauswahl hinzu und verbessert die Fallback-Logik, wenn die bevorzugten Zeitstempel nicht verfügbar sind. • Behebt ein Problem, bei dem OPC UA-Collector-Konfigurationen für klassische Streams und V2-Gateways standardmäßig nicht auf Stream Manager gesetzt werden, wenn kein Ziel angegeben ist.
3.0.0	Neue Features Unterstützung für MQTT-fähige V3-Gateways wurde zusätzlich zu Classic-Streams und V2-Gateways (früher bekannt als selbst gehostete Edge-Gateways) hinzugefügt. SiteWise Weitere Informationen finden Sie unter MQTT-fähige V3-Gateways für Edge. AWS IoT SiteWise • Ermöglicht die Veröffentlichung von OPC UA-Daten auf dem AWS IoT Greengrass MQTT 5-Broker (EMQX). Fehlerkorrekturen und Verbesserungen • Behebt Sicherheitslücken.

Version	Änderungen
2.6.0	Neue Features • Unterstützung für die Aufnahme von Null- und NaN-Werten hinzugefügt, wenn die Aufnahme in aktiviert ist. AWS IoT SiteWise Informationen zum Anzeigen oder Ändern der Null- und NaN-Konfiguration in AWS IoT SiteWise finden Sie unter DescribeStorageConfigurationund PutStorageConfiguration APIs.
2.5.1	Fehlerkorrekturen und Verbesserungen • Behebt einen Fehler, bei dem future Snapshot-Aufgaben abgebrochen werden, wenn während der Ausführung einer Snapshot-Aufgabe ein Fehler auftritt. • Behebt einen Fehler, bei dem Aktualisierungen der Datenquellenkonfiguration erst nach dem Neustart des OPC UA Collector bestehen bleiben, wenn die Verbindung zum OPC UA-Server der Datenquelle unterbrochen wurde.
2.5.0	Neue Features • Fügt eine Datenquellenoption hinzu, um einfache Arrays und Werte in Zeichenketten zu konvertieren. DateTime • Fügt eine Eigenschaftsgruppenoption hinzu, mit der beim Sammeln von Daten von einem OPC UA-Server entweder ein Quell- oder ein Serverzeitstempel ausgewählt werden kann. Fehlerkorrekturen und Verbesserungen • Setzt die Standard-Polling-Konfiguration zurück, sodass der Quell-Zeitstempel verwendet wird.
2.4.2	Fehlerkorrekturen und Verbesserungen • Behebt Probleme bei der Erkennung von OPC UA-Servern, bei denen ein Knoten mehrfach entdeckt werden kann. • Behebt die Snapshot-Funktion, um sicherzustellen, dass der Zeitstempel für jeden Snapshot-Datenpunkt neu ist.

Version	Änderungen
2.4.1	Fehlerkorrekturen und Verbesserungen • Behebt Probleme im Zusammenhang mit der Proxyunterstützung. • Behebt ein Problem, bei dem die Thread-Bereinigung fehlschlug und zu einer Datenblockierung führte.
2.4.0	Neue Features • Fügt ein Ereignisprotokoll hinzu, um Probleme leichter identifizieren und beheben zu können. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem mit dem OPC UA-Client, das zu Zertifikatsfehlern führte, wenn eine Verbindung zu einem OPC UA-Server hergestellt wurde, der Version 1.05 der OPC UA-Spezifikation verwendet.
2.3.0	Neue Features • Fügt Unterstützung für die Greengrass Nucleus-HTTP-Proxykonfiguration unter Linux hinzu. Fehlerkorrekturen und Verbesserungen • Behebt ein Sicherheitsproblem (CVE-2019-19135).

2.2.0	Neue Features
	• Fügt Unterstützung für die Installation von Data Collection Pack auf einer Linux-Architektur hinzu. ARMv8
	• Mindestanforderungen für Linux ARMv8:
	  • Arbeitsspeicher: 4 GB
	  • CPU: ARM Cortex-A72 oder gleichwertige Spezifikation
	Fehlerkorrekturen und Verbesserungen
	• Verbessert die Protokollierung von Metriken bei der Knotenerkennung.
	• Verbessert den Umgang mit nicht unterstützten Datentypen.
	• Verbessert die Protokollierung von Datenstromfehlern.

Version	Änderungen
2.1.3	Neue Features • Fügt Unterstützung für Windows Server 2019 oder höher hinzu. Fehlerkorrekturen und Verbesserungen • Verbessert die Fehlermeldungen, wenn Sie diese Komponente auf Geräten bereitstellen, die nicht unterstützt werden.

2.1.1	Neue Features
	• Integriert die Unterstützung für die Konfiguration der folgenden Abonnementeigenschaften:
	  • DataChangeTrigger- Sie können die Bedingung definieren, die eine Datenänderungswarnung auslöst.
	  • QueueSize- Die Tiefe der Warteschlange auf einem OPC-UA-Server für eine bestimmte Metrik, in der Benachrichtigungen für überwachte Objekte in die Warteschlange gestellt werden.
	  • PublishingIntervalMilliseconds- Das Intervall (in Millisekunden) eines Veröffentlichungszyklus, der bei der Erstellung eines Abonnements angegeben wird.
	  • SnapshotFrequencyMilliseconds - Sie können die Einstellung für das Timeout für die Snapshot-Frequenz konfigurieren, um sicherzustellen, dass AWS IoT SiteWise Edge einen stetigen Datenstrom aufnimmt.
	• Diese Version unterstützt die Aufnahme von `BAD` Qualitätsdaten und filtert Daten auf der Grundlage der folgenden Datenqualitäten:
	  • `UNCERTAIN` Qualitätsdaten
	  • `BAD`hochwertige Daten
	
	Fehlerkorrekturen und Verbesserungen
	• Verbesserungen der Kundenkennzahlen.
	• Behebt die Sicherheitskodierung, die manchmal zu Problemen bei der Verbindung zu Servern mit aktivierter Verschlüsselung führte.
	• Behebt ein Problem, bei dem die Eigenschaftengruppe nicht aktualisiert werden konnte.

Version	Änderungen
2.0.3	Fehlerkorrekturen und Verbesserungen.
2.0.2	Fehlerkorrekturen und Verbesserungen bei der prioritären Synchronisierung von Vermögenswerten 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 SiteWise IoT-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 SiteWise IoT-OPC-UA-Kollektorkomponente auf einem AWS IoT SiteWise Gateway gelesen wird. Anschließend können Sie anhand dieser AWS IoT SiteWise Beispieldaten Funktionen untersuchen. Weitere Informationen zu AWS IoT SiteWise Gateways finden Sie im AWS IoT SiteWise Benutzerhandbuch unter Using AWS IoT SiteWise at the Edge.

Themen

- Versionen
- Typ
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Konfiguration
- Lokale Protokolldatei
- Änderungsprotokoll
- Weitere Informationen finden Sie auch unter

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 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 in der Lage sein, Port 4840 auf dem lokalen Host zu verwenden. Der lokale OPC UA-Server dieser Komponente läuft an diesem Port.

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 alle Versionen dieser Komponente aufgeführt.

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
Grüngraskern	>=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.SiteWiseEdgeOpcuaDataSourceSimulator.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.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.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 den einzelnen Versionen der Komponente beschrieben.

Version	Änderungen
1.0.0	Erste Version 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
- Typ
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Konfiguration

- Eingabedaten
- Lokale Protokolldatei
- Problembehandlung und Debugging
- Lizenzen
- Änderungsprotokoll
- Weitere Informationen finden Sie auch unter

Versionen

Diese Komponente hat die folgenden Versionen:

- 4.1.x
- 4.0.x
- 3.2.x
- 3.1.x
- 3.0.x
- 2.4.x
- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

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 () oder (Aarch64) AMD64 ARMv8
  - Betriebssystem: Red Hat Enterprise Linux (RHEL) 8

    Architektur: x86_64 (AMD64) oder (Aarch64) ARMv8
  - 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 später

    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

  JSON

```
{
    "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 Geräteverkehr über einen Proxy oder eine Firewall zulassen.

Endpoint	Port	Erforderlich	Description
`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

-Abhängigkeit	Kompatible Versionen	Art der Abhängigkeit
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 unter PutAssetPropertyValueEntry in 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. Die Diagnoseinformationen umfassen Folgendes:

- 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
4.1.4	Fehlerkorrekturen und Verbesserungen • SiteWise Der IoT-Publisher meldet jetzt Systemintegritätsmetriken wie Speicher- und CPU-Auslastung des SiteWise IoT-Publishers an CloudWatch. • Die Fehlerberichterstattung wurde aktualisiert, AWS IoT Greengrass sodass der SiteWise IoT-Publisher neu gestartet wird, wenn der Start-Workflow teilweise ausfällt.
4.1.3	Fehlerkorrekturen und Verbesserungen • Behebt ein Speicherleck beim Datencheckpointing. • Behebt Sicherheitslücken.
4.1.2	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem beim Synchronisieren großer Konfigurationen. • Verzögerungen beim Abonnement wirken sich nicht AWS IoT Core mehr auf den Startvorgang aus. • Verzögerungen bei der Initialisierung von Checkpoints wirken sich nicht mehr auf den Startvorgang aus.
4.1.1	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, das zu einer hohen CPU- und Speicherauslastung auf Classic-Streams, V2-Gateways führt, wenn Daten nach dem Neustart des Publishers über ein AWS IoT SiteWise gepuffertes Amazon S3 S3-Ziel gesendet werden. • Fügt Unterstützung für das Überschreiben des standardmäßigen Protokollierungsverhaltens von Komponenten hinzu, indem eine optionale Datei erstellt wird, die sich unter: befindet. `/greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/log4j2.xml`
4.1.0	Neue Features • Fügt ein Befehlszeilentool zur Verwaltung integrierter EMQX-MQTT-Broker-Benutzer hinzu. Diese Schnittstelle ermöglicht die Benutzerk

Version	Änderungen
	ontenverwaltung und konfiguriert Zugriffskontrollrichtlinien für Veröffent lichungs- und Abonnementberechtigungen für MQTT-Themen. Fehlerkorrekturen und Verbesserungen • Behebt die Berechnung, die bestimmt, wie viele eindeutige Aliase in eine einzelne Buffered Ingestion Parquet-Datei passen können.
4.0.3	Fehlerkorrekturen und Verbesserungen • Aktualisiert die maximale Anzahl eindeutiger Aliase pro AWS IoT SiteWise gepufferter Aufnahmedatei, sodass sie dem Cloud-Limit von 10.000 entspricht. • Behebt ein AWS IoT Greengrass Protokollierungsproblem, das fälschlic herweise ungültige Dienststatus meldet. • Fügt eine Validierung hinzu, um doppelte Stream-Namen in Amazon S3 S3-Zielkonfigurationen in Classic-Streams und V2-Gateways zu verhindern. • Behebt ein Kompatibilitätsproblem mit AWS IoT SiteWise gepufferter Aufnahme in nicht standardmäßigen Partitionen. AWS • Behebt ein Problem, das zur doppelten Datenaufnahme führt.
4.0.2	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, das das Hochladen von Daten nach Ablauf der Batch-Wartezeit verhindert. • Fügt eine Validierung hinzu, um zu überprüfen, ob jedes Ziel in MQTT-fähigen V3-Gateway-Konfigurationen einzigartig ist.

Version	Änderungen
4.0.1	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, das das Hochladen von Daten verhindert, AWS IoT SiteWise wenn Sie die Konfiguration für die maximale Batch-Wartezeit deaktivieren. • Behebt ein Problem, bei dem die `IoTSiteWisePublisher.NumberOfSubscriptionsToMqttBroker` Metrik fälschlicherweise die Anzahl der einzelnen Themen angibt, die für den MQTT-Broker abonniert wurden.
4.0.0	Neue Features Integriert Unterstützung für MQTT-fähige V3-Gateways zusätzlich zu klassischen Streams und V2-Gateways (früher bekannt als selbst gehostete Edge-Gateways). SiteWise • Fügt die Möglichkeit hinzu, den MQTT 5-Broker (EMQX) zu abonniere n und Daten vom MQTT 5-Broker zu empfangen. AWS IoT Greengras s Weitere Informationen finden Sie unter MQTT-fähige V3-Gateways für Edge. AWS IoT SiteWise • Fügt Konfigurationsoptionen für Datenziele mithilfe von Pfadfiltern hinzu. Weitere Informationen finden Sie unter Grundlegendes zu Pfadfiltern für AWS IoT SiteWise Edge-Ziele. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem die `PublishSuccessCount` Metrik negative Werte meldet. • Behebt ein Problem, bei dem der Herausgeber nicht innerhalb von 100 Sekunden startet und den `BROKEN` Status wechselt. • Behebt Sicherheitslücken.

Version	Änderungen
3.2.0	Neue Features • Unterstützung für die Aufnahme von Null- und NaN-Werten hinzugefügt, wenn die Aufnahme in aktiviert ist. AWS IoT SiteWise Informationen zum Anzeigen oder Ändern der Null- und NaN-Konfiguration in AWS IoT SiteWise finden Sie unter DescribeStorageConfigurationund PutStorageConfiguration APIs. Fehlerkorrekturen und Verbesserungen • Behebt Probleme, die zu beschädigten Checkpoint-Datenbankdateien führen. • Behebt Probleme beim Generieren doppelter Metriken.
3.1.4	⚠ Warning Version 3.1.4 wurde am 20. Februar 2025 eingestellt. Die Verbesserungen in dieser Version sind in späteren Versionen dieser Komponente verfügbar. Fehlerkorrekturen und Verbesserungen • Behebt Probleme, die zu longer-than-expected Startzeiten führen konnten, nachdem Sie offline waren.

Version	Änderungen
3.1.3	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem die Ereignisprotokolldatei unter erstellt `/greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/logs/IoTSiteWisePublisherEvents.log` wurde, aber keine Ereignisse protokolliert wurden. • Fügt die folgenden CloudWatch Metriken für die Überwachung der Verbindung mit dem MQTT-Broker hinzu: • `IoTSiteWisePublisher.IsConnectedToMqttBroker` • `IoTSiteWisePublisher.NumberOfSubscriptionsToMqttBroker` • `IoTSiteWisePublisher.NumberOfUniqueMqttTopicsReceived` • `IoTSiteWisePublisher.MqttMessageReceivedSuccessCount` • `IoTSiteWisePublisher.MqttReceivedSuccessBytes` Weitere Informationen zu diesen Metriken finden Sie unter AWS IoT Greengrass Version 2 Gateway-Metriken. • Behebt ein Problem, bei dem die `BatchCreateJob` API auch dann noch aufgerufen wird, wenn das Hochladen einer Parquet-Datei auf S3 fehlschlägt.
3.1.2	Fehlerkorrekturen und Verbesserungen • Behebt das in Version 3.1.1 eingeführte Problem der hohen CPU-Auslastung.

Version	Änderungen
3.1.1	**Warning** Version 3.1.1 wurde am 12. März 2024 eingestellt. Die Verbesserungen in dieser Version sind in späteren Versionen dieser Komponente verfügbar. 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 StreamManager 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-Broker.

Version	Änderungen
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.
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 einer ARMv8 Linux-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.

Version	Änderungen
2.2.2	Fehlerkorrekturen und Verbesserungen • Führt die Unterstützung für das Hochladen von Daten AWS IoT SiteWise über einen HTTP-Proxyserver wieder ein.
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

Version	Änderungen

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.

2.1.4	Fehlerkorrekturen und Verbesserungen • Behebt die Kompatibilität mit Java-Version 8.

Version	Änderungen
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 Geräten bereitstellen, die nicht unterstützt werden. • 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 die Festplatte I/O und die entsprechende Latenz.
2.0.2	Fehlerkorrekturen und Verbesserungen.
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 AWS IoT SiteWise Classic Streams und V2-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.

> **Note**
>
> Die Funktion Data Processing Pack (DPP) steht Neukunden ab dem 7. November 2025 nicht mehr zur Verfügung. Wenn Sie DPP nutzen möchten, melden Sie sich vor diesem Datum an. Bestandskunden können den Service weiterhin wie gewohnt nutzen. Weitere Informationen finden Sie unter Änderung der Verfügbarkeit von Datenverarbeitungspaketen im AWS IoT SiteWise Benutzerhandbuch.

Themen

- Versionen
- Typ
- Betriebssystem
- Voraussetzungen
- Abhängigkeiten
- Konfiguration
- Lokale Protokolldatei
- Lizenzen
- Änderungsprotokoll
- Weitere Informationen finden Sie auch unter

Versionen

Diese Komponente hat die folgenden Versionen:

- 3.5.x
- 3.4.x
- 3.3.x
- 3.2.x
- 3.1.x
- 3.0.x
- 2.2.x
- 2.1.x
- 2.0.x

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 höher

Architektur: x86_64 () AMD64

- Betriebssystem: RedHat Enterprise Linux (RHEL) 8

  Architektur: x86_64 () AMD64

- Betriebssystem: Amazon Linux 2

  Architektur: x86_64 () AMD64

- Betriebssystem: Windows Server 2019 oder später

  Architektur: x86_64 () AMD64

- Betriebssystem: Debian 11 (Bullseye) oder später

  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 Geräteverkehr über einen Proxy oder eine Firewall zulassen.

Endpoint	Port	Erforderlich	Description
`model.iotsitewise. region.amazonaws.com`	443	Ja	Informieren Sie sich über Ihre AWS IoT SiteWise Anlagen und Anlagemodelle.
`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.

Endpoint	Port	Erforderlich	Description
`iot.region.amazonaws.com`	443	Ja	Holen Sie sich Geräteendpunkte für Ihre. AWS-Konto
`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 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 2.1.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- with-classpath-exception
- 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
3.5.1	Neue Features Unterstützung für die Aufnahme von Null- und NaN-Werten hinzugefü gt, wenn die Aufnahme in aktiviert ist. AWS IoT SiteWise Informationen zum Anzeigen oder Ändern der Null- und NaN-Konfiguration in AWS IoT SiteWise finden Sie unter DescribeStorageConfigurationund PutStorag eConfiguration APIs. Fehlerkorrekturen und Verbesserungen Aktualisierte Abhängigkeiten zur Behebung potenzieller Sicherheitslücken.
3.4.0	Neue Features • Optionale HTTP- und HTTPS-Proxyunterstützung hinzugefügt, um die Gateway-Kommunikation über Proxyserver zu ermöglichen, für die benutzerdefinierte Zertifikate erforderlich sind. Weitere Informationen zum Einrichten eines HTTP-Proxys für den AWS IoT Greengrass Core finden Sie unterVerbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy. Weitere Informationen zur SiteWise Edge-spezifischen Proxyunterstützung finden Sie unter Verwalten von Vertrauensspeicher n für die AWS IoT SiteWise Edge-Proxyunterstützung im AWS IoT SiteWise Benutzerhandbuch. • Es wurden konfigurierbare Einstellungen für das Sitzungs-Timeout hinzugefügt, um Inaktivitätszeiträume für AWS OpsHub Edge zu verwalten. SiteWise APIs Weitere Informationen finden Sie im Benutzerh andbuch unter Konfigurieren von Sitzungs-Timeouts für AWS IoT SiteWise Edge.AWS IoT SiteWise

Version	Änderungen
	Leistungsverbesserungen Die Zeit, in der eingehende Daten den Edge-Gerätespeicher erreichen, wurde von 5 Sekunden auf weniger als 1 Sekunde reduziert. Die Latenz für Daten-Uploads AWS IoT SiteWise bleibt unverändert.
3.3.1	Neues Feature • SiteWise Edge wurde um optionale CORS-Unterstützung erweitert APIs, wodurch die ursprungsübergreifende Nutzung von Ressource n verbessert wird. Diese Funktion verbessert die Flexibilität für Webanwendungen, die mit dem interagieren. APIs
3.3.0	Leistungsverbesserungen • Der Cache-Aktualisierungsmechanismus wurde optimiert, um die I/O Auslastung zwischen AWS IoT SiteWise Asset-Synchronisierungen zu reduzieren, indem nur Einträge für neue oder aktualisierte Assets aktualisiert werden. • Reduzierter Speicherbedarf für die Verwaltung eines Caches mit einer großen Anzahl synchronisierter Asset-Eigenschaften. Fehlerkorrekturen und Verbesserungen • Es wurden Protokolle für die Aufnahme einzelner Eigenschaftswerte unterdrückt, wenn keine Aufnahmefehler vorliegen. Dadurch wird das Protokollrauschen bei hohen Aufnahmeraten reduziert. • Die Lesbarkeit von Protokollen wurde verbessert, indem für bestimmte Protokolleinträge eine menschenlesbare Formatierung verwendet wurde. • Unterstützung für Java 17 und höher hinzugefügt.

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.

Version	Änderungen
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. 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 die AWS IoT SiteWise Datenebene APIs, z. B. `get-asset-property-value` , wird von `http://localhost:8081` zu geändert. `http://localhost:11080/data` Der lokale Endpunkt für die AWS IoT SiteWise Steuerungsebene 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.

Version	Änderungen
	• 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 extern aufgerufen werden konnte. APIs Nur HTTPS kann jetzt für Anrufe APIs außerhalb der lokalen Loopback-Adresse verwendet werden. • `ListAssets` Die API zeigt jetzt die Asset-Hierarchien für Assets, die am Edge gespeichert sind. • 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.

Version	Änderungen
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-Broker.
2.2.1	Fehlerkorrekturen und Verbesserungen • Passen Sie den Synchronisierungsprozess an, um die Datenspeicherung 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 durchzuführen, 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.

Version	Änderungen
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.
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 metric/transform Berechnungen an die Verwendung von Multithre ading unter Linux an. Windows führt aus Kompatibilitätsgründen weiterhin Single-Thread-Berechnungen aus. • 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.

Version	Änderungen
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.
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.

Version	Änderungen
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.
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 installiert sind, welche Version die einzelnen Komponenten haben und welchen Status die einzelnen Komponenten haben. Sie können diese Informationen 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.

Version	Änderungen
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.
2.0.2	Erste Version

Weitere Informationen finden Sie auch unter

- Was ist AWS IoT SiteWise? im AWS IoT SiteWise Benutzerhandbuch.

Von Publishern unterstützte Komponenten

Von Publisher unterstützte Komponenten befinden sich in einer Vorabversion für und können sich ändern. AWS IoT Greengrass Diese Komponenten werden von nicht unterstützt. AWS Bei Problemen mit den einzelnen Komponenten müssen Sie sich an den Herausgeber wenden.

Die von Greengrass Publisher unterstützten Komponenten werden von Drittanbietern entwickelt, angeboten und gewartet. Drittanbieter von Komponenten stammen entweder aus dem AWS Partner Gerätekatalog, AWS Heroes oder Community-Anbietern. Sie können die Komponenten in diesem Katalog erwerben, indem Sie sich direkt an den Drittanbieter der Komponenten wenden.

Zu den von Greengrass Publisher unterstützten Komponenten gehören:

Themen

- AIShield. Edge
- EdgeLabs KI-Sensor
- Greengrass S3 Ingestor

AIShield. Edge

Diese Komponente wurde von Bosch entwickelt und wird von AIShield Bosch unterstützt. Steigern Sie Ihre KI-Sicherheit mit AIShield .Edge. Diese Komponente wurde für die nahtlose Bereitstellung von bedrohungsinformierten, maßgeschneiderten Abwehrmaßnahmen für Edge-Geräte entwickelt, wodurch Ihre Geräte vor KI-Angriffen geschützt werden.

Diese Komponente bietet die folgenden Vorteile:

- Nahtloser Übergang von der Schwachstellenanalyse mit AIShield KI Security zu verstärkten Edge-Abwehrmaßnahmen innerhalb AWS
- Stellen Sie mühelos maßgeschneiderte Abwehrmaßnahmen auf mehreren Edge-Geräten bereit
- Umfassender Schutz, der auf verschiedene KI-Setups zugeschnitten ist und verschiedene Modelltypen und Frameworks unterstützt
- Bleiben Sie mit der nahtlosen Integration in Amazon SageMaker AI und Greengrass-Workflows auf dem Laufenden
- Verschaffen Sie sich sofortige Einblicke in potenzielle Bedrohungen, wobei die Daten direkt weitergeleitet werden AWS IoT Core
- Ein kohärenter KI-Sicherheitspfad für den Einsatz von Verteidigungsgütern an der Peripherie von AIShield KI Security auf dem Marketplace AWS

Diese Komponente muss auf der folgenden Plattform ausgeführt werden:

- Betriebssystem: Linux

Wenn Sie am Kauf dieser Komponente interessiert sind, wenden Sie sich an Bosch Software and Digital Solutions: `<AIShield.Contact@bosch.com>`.

EdgeLabs KI-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. Der KI-Sensor ist in eine Greengrass-Komponente integriert und wird zusammen mit anderen Greengrass-Komponenten als eigenständiger Container auf dem Kerngerät bereitgestellt.

Bei dieser aktuellen Komponente handelt es sich um einen containerbasierten Agenten, 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)
- Bedrohungsinformationen vor Ort auf dem Host mit minimalem Datentransfer nach außen
- Einfache Bereitstellung mit Docker und Greengrass

Diese Komponente muss auf einer der folgenden Plattformen laufen:

- Betriebssystem: Linux

Wenn Sie am Kauf dieser Komponente interessiert sind, wenden Sie sich an AI EdgeLabs: `<contact@edgelabs.ai>`.

Greengrass S3 Ingestor

Diese Komponente wurde von Nathan Glover entwickelt und wird unterstützt. Die Greengrass S3 Ingestor-Komponente wurde für die Verwendung mit der Stream-Manager-Komponente entwickelt. Diese Komponente nimmt einen durch Zeilen getrennten Stream von JSON-Nachrichten aus dem Stream-Manager auf und fasst sie in einer GZIP-Datei zusammen. Diese Komponente ermöglicht die effiziente Aufnahme von Daten in Amazon S3 zur weiteren Verarbeitung oder Speicherung. Diese Komponente unterstützt nicht das Senden von Daten an die AWS Cloud in Echtzeit.

Diese Komponente muss auf einer der folgenden Plattformen ausgeführt werden:

- Betriebssystem: Linux
- Betriebssystem: Windows

```
<Wenn Sie am Kauf dieser Komponente interessiert sind, wenden Sie sich an Nathan
```

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 verfügt über 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.
- GStreamerfür Amazon Lookout for Vision

Diese Komponente stellt ein GStreamer Plugin bereit, mit dem Sie die Erkennung von Lookout for Vision Vision-Anomalien 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.

- Influx-Dashboard DBGrafana

  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 AWS IoT Greengrass Komponenten erstellen.

- 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, ein LPWAN-Protokoll (Low-Power Wide Area Network). 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-Austauschdienst

- 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 hier: CONTRIBUTING.mdDatei im Repository jeder Komponente.

Einige der von AWS-bereitgestellten Komponenten sind ebenfalls Open Source. Weitere Informationen finden Sie unter AWS IoT Greengrass Open-Source-Kernsoftware.

AWS IoT Greengrass Entwicklungstools

Verwenden Sie AWS IoT Greengrass Entwicklungstools, um benutzerdefinierte Greengrass-Komponenten zu erstellen, zu testen, zu erstellen, zu veröffentlichen und bereitzustellen.

- Greengrass-Entwicklungskit CLI

  Verwenden Sie das 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 AWS IoT Greengrass AWS-Konto Dienst zu veröffentlichen.

- Greengrass-Befehlszeilenschnittstelle

  Verwenden Sie das Greengrass Command Line Interface (Greengrass CLI) auf Greengrass-Kerngeräten, um Greengrass-Komponenten bereitzustellen und zu debuggen. Die Greengrass-CLI ist eine Komponente, die Sie auf Ihren Kerngeräten 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 mithilfe einer lokalen Dashboard-Weboberfläche bereitzustellen und zu debuggen. Die lokale Debug-Konsole ist eine Komponente, die Sie auf Ihren Kerngeräten bereitstellen können, um lokale Bereitstellungen zu erstellen und Details zu den installierten Komponenten anzuzeigen.

AWS IoT Greengrass bietet außerdem Folgendes SDKs , das Sie in benutzerdefinierten Greengrass-Komponenten verwenden können:

- Das AWS IoT Device SDK, das 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 Datenstreams an das übertragen können. AWS Cloud Weitere Informationen finden Sie unter Datenströme auf Greengrass-Kerngeräten verwalten.

Themen

- AWS IoT Greengrass Befehlszeilenschnittstelle für das Entwicklungskit
- Greengrass-Befehlszeilenschnittstelle
- Verwenden Sie AWS IoT Greengrass Testing Framework

AWS IoT Greengrass Befehlszeilenschnittstelle für das Entwicklungskit

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-Softwarekatalog beginnen. Anschließend können Sie ein Build-System wählen, 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 sie mit der GDK-CLI im AWS IoT Greengrass Service veröffentlichen, sodass Sie die AWS IoT Greengrass Konsole oder API verwenden können, um die Komponente auf Ihren Greengrass-Kerngeräten bereitzustellen.

Wenn Sie Greengrass-Komponenten ohne die GDK-CLI entwickeln, müssen Sie die Version und das Artefakt URIs in der Komponentenrezeptdatei jedes Mal aktualisieren, wenn Sie eine neue Version der Komponente erstellen. Wenn Sie die GDK-CLI verwenden, kann sie die Version und das Artefakt jedes Mal, wenn Sie eine neue Version der Komponente veröffentlichen, automatisch URIs für Sie aktualisieren.

Die GDK-CLI ist Open Source und verfügbar unter GitHub. Sie können die GDK-CLI an Ihre Anforderungen an die Komponentenentwicklung anpassen und erweitern. Wir laden Sie ein, Issues und Pull-Requests im GitHub Repository zu öffnen. Die GDK-CLI-Quelle finden Sie unter dem folgenden Link: https://github.com/aws-greengrass/aws-greengrass-gdk-cli.

Voraussetzungen

Um das Greengrass Development Kit CLI zu installieren und zu verwenden, benötigen Sie Folgendes:

- Ein 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) wurde mit Anmeldeinformationen auf Ihrem Entwicklungscomputer installiert und konfiguriert. Weitere Informationen finden Sie unter Installation, Aktualisierung und Deinstallation von AWS CLI und Konfiguration von 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 Installation, Aktualisierung und Deinstallation von 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 mit der GDK-CLI eine Komponente zu erstellen, deren Artefakte in einem S3-Bucket und nicht im lokalen Dateisystem existieren, benötigen Sie die folgenden Berechtigungen:
  - `s3:ListBucket`

  Diese Funktion ist für GDK CLI v1.1.0 und höher verfügbar.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen der GDK-CLI beschrieben. Weitere Informationen finden Sie auf der Seite GDK CLI Releases unter GitHub.

Version	Änderungen
1.6.2	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem Windows gradlew.bat aufgrund des relativen Pfads nicht funktioniert. • Kleinere Verbesserungen bei der Protokollierung, beim Testen und beim Paketieren.
1.6.1	Fehlerkorrekturen und Verbesserungen • Fügt einen Sicherheitsfix für das Parsen von CLI-Argumenten hinzu. • Ermöglicht es dem GDK, den neuesten Release-Namen des Greengras s Testing Framework (GTF) als Standard-GTF-Version abzurufen. • Ermöglicht GDK, Kunden, die eine ältere Version von GTF verwenden, zu empfehlen, auf die neueste Version zu aktualisieren.
1.6.0	Neue Features • Fügt während der `component publish` Befehle und eine Überprüfu ng der Rezeptvalidierung anhand des Greengrass-Rezeptschemas `component build` hinzu. Dieses Update hilft Entwicklern dabei, umsetzbare Probleme in ihren Komponentenrezepten zu einem früheren Zeitpunkt der Komponentenerstellung zu identifizieren. • Fügt der Vorlage eine Suite für Vertrauenstests 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 Komponent entests gerecht zu werden. Fehlerkorrekturen und Verbesserungen • Aktualisiert die vom `test-e2e` Befehl verwendete Standardversion des Greengrass Testing Framework (GTF) auf Version 1.2.0.

Version	Änderungen
1.5.0	Fehlerkorrekturen und Verbesserungen Aktualisiert die von der `excludes` Build-Option erkannten Muster, wenn dies der `build_system` 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.
1.4.0	Neue Features • Fügt einen neuen `config` Befehl hinzu, der eine interaktive Aufforder ung zum Ändern von Feldern in einer vorhandenen GDK-Konfigurations datei startet. • Ändert die `gdk component publish` Befehle `gdk component build` und, um zu überprüfen, ob die Rezeptgröße den Greengrass-Anforderungen entspricht (<=16000 Byte), bevor Sie fortfahren. Fehlerkorrekturen und Verbesserungen • Fügt zusätzliche Protokollierung in der Ausgabe des `gdk component build` Befehls hinzu, wenn ein Syntaxfehler im Rezept verhindert, dass der Build aus Sicherheitsgründen abgeschlossen werden kann. • Benennt das `otf-options gtf-options` und `otf-version` in `gtf-version` bzw. um, aufgrund der Umbenennung von Open Test Framework in Greengrass Testing Framework.

Version	Änderungen
1.3.0	Neue Features • Fügt einen neuen `test-e2e` Befehl hinzu, der das end-to-end Testen von Komponenten mit Open Test Framework unterstützt. • Fügt eine neue Konfigurationsoption hinzu`zip_name`, um konfiguri erbare ZIP-Dateinamen mit dem Zip-Build-System zu unterstützen. • Macht die `region` Eigenschaft in der GDK-Konfigurationsdatei optional. Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem ein neues Verzeichnis erstellt wird, auch wenn die angegebene Vorlage oder das angegebene Repository nicht existiert, wenn ein GDK-Projekt mit dem `--name` Argument initialisiert wird.
1.2.3	Fehlerkorrekturen und Verbesserungen • Behebt ein Problem, bei dem die Bucket-Erstellung aufgrund einer falschen Fehlerbehandlung fehlschlägt. • Behebt ein Problem, bei dem Listenstrukturen im Komponentenrezept entfernt wurden.
1.2.2	Fehlerkorrekturen und Verbesserungen • Bei Rezeptschlüsseln wird nicht mehr zwischen Groß- und Kleinschr eibung unterschieden. • Fügt vor dem Erstellen eines neuen Buckets eine Prüfung hinzu, um festzustellen, ob ein Bucket in einem vorhanden ist AWS-Region und der Benutzer darauf zugreifen kann. Erfordert, dass der Benutzer über die `GetBucketLocation` entsprechende Berechtigung verfügt. • Behebt ein Problem mit dem `excludes` Schlüsselwort in der GDK-CLI-Konfigurationsdatei.
1.2.1	Fehlerkorrekturen und Verbesserungen • Akzeptiert den Konfigurationseintrag Kanada (Central`ca-central-1`) () AWS-Region in 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 den `options` Eintrag zur `build` Konfiguration in der GDK-CLI-Konfigurationsdatei hinzu. Unterstützt `excludes` Underoptions, 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 Build-Option hinzu. `gradle` • Fügt der `publish` Konfiguration in der GDK-CLI-Konfigurationsdatei einen `options` Eintrag hinzu. Unterstützt das `file_upload_args` Underoptions, um zusätzliche Argumente beim Hochladen von Dateien auf Amazon S3 bereitzustellen. Fehlerkorrekturen 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 zum Veröffentlichungsbefehl der Komponente hinzu. Sie können dieses Argument verwenden, um den genauen Bucket anzugeben, in den die GDK-CLI Komponentenartefakte hochlädt. • Fügt das `--name` Argument dem Befehl Component Init 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, aber nicht im lokalen Komponenten-Build-Ordner existieren. Sie können diese Funktion verwenden, um die Bandbreitenkosten für große Komponentenartefakte, wie z. B. Modelle für maschinelles Lernen, zu reduzieren. Fehlerkorrekturen und Verbesserungen • Aktualisiert den Befehl zum Veröffentlichen von Komponenten, um zu überprüfen, ob die Komponente erstellt wurde, bevor die Komponente veröffentlicht wird. Wenn die Komponente nicht erstellt wurde, erstellt dieser Befehl die Komponente jetzt für Sie. • Behebt ein Problem, bei dem das ZIP-Build-System auf Windows-Geräten nicht erstellt werden kann, wenn der Name der ZIP-Datei Großbuchstaben enthält. • Verbessert das Protokollnachrichtenformat und ändert die Standardp rotokollebene `INFO` auf Geräten, auf denen Python-Versionen vor 3.8 ausgeführt werden, auf. • Ändert die Mindestanforderung für die Python-Version auf Python 3.6.
1.0.0	Erste Version

Installieren oder aktualisieren Sie die AWS IoT Greengrass Development-Kit-Befehlszeilenschnittstelle

Das AWS IoT Greengrass Development Kit Command-Line Interface (GDK CLI) basiert auf Python, sodass Sie es auf Ihrem Entwicklungscomputer installieren können. `pip`

> **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.

Um die GDK-CLI zu installieren oder zu aktualisieren

1. Führen Sie den folgenden Befehl aus, um die neueste Version der GDK-CLI aus dem GitHubRepository 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, *`versionTag`* ersetzen Sie sie durch das zu installierende Versions-Tag. Sie können Versionstags für die GDK-CLI in ihrem GitHubRepository 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.

- Auf Linux-Geräten fügen Sie `/home/MyUser/.local/bin` es zu PATH hinzu und *MyUser* ersetzen Sie es durch den Namen Ihres Benutzers.
- Fügen Sie auf Windows-Geräten PATH *PythonPath*`\\Scripts` hinzu und *PythonPath* ersetzen Sie es durch 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 unterAWS IoT Greengrass Befehle für die Befehlszeilenschnittstelle des Development Kits.

AWS IoT Greengrass Befehle für die Befehlszeilenschnittstelle des Development Kits

Das 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 dem PATH `gdk` etwas hinzu, sodass Sie die GDK-CLI von der Befehlszeile aus 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 von 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 Zusammenfassung 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 im 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-Softwarekatalog und Komponentenvorlagen aus dem Component AWS IoT Greengrass Templates-Repository 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 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 existiert. 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 Name des ZIP-Artefakts mit dem Namen des Komponentenordners übereinstimmen. Wenn Sie GDK CLI Version 1.0.0 auf einem Windows-Gerät verwenden, dürfen die Namen des Komponentenordners und der ZIP-Dateien nur Kleinbuchstaben enthalten.

Wenn Sie eine Vorlage oder Community-Komponente, die das ZIP-Build-System verwendet, für 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 Name der Zip-Datei mit dem Namen des Komponentenordners übereinstimmt. Im folgenden Beispiel wird der Name der Zip-Datei 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 (aus der Komponentenvorlage initialisieren)

- `-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. Verwenden Sie den Befehl list, um die verfügbaren Vorlagen anzuzeigen.

  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 existiert. 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 aus initialisieren)

- `-r`, `--repository` — Die Community-Komponente, die in den lokalen Ordner ausgecheckt werden soll. Verwenden Sie den Befehl list, um die verfügbaren Community-Komponenten anzuzeigen.

  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 existiert. 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-Vorlage Hello World 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

Erstellen Sie aus der Quelle einer Komponente ein Rezept und Artefakte, die Sie im AWS IoT Greengrass Service veröffentlichen können. Die GDK-CLI führt das Build-System aus, das Sie in der GDK-CLI-Konfigurationsdatei angeben,. `gdk-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 in dem `greengrass-build` Ordner im Komponentenordner. Die GDK-CLI speichert das Rezept im `greengrass-build/recipes` Ordner und speichert 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 lokalen Komponenten-Build-Ordner. Sie können diese Funktion verwenden, um die Bandbreitennutzung zu reduzieren, wenn Sie Komponenten mit großen Artefakten entwickeln, z. B. Modelle für maschinelles Lernen.

Nachdem Sie eine Komponente erstellt haben, können Sie eine der folgenden Aktionen ausführen, um sie auf einem Greengrass-Core-Gerät zu testen:

- Wenn Sie auf einem anderen Gerät entwickeln als auf dem, 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-Core-Gerät bereit. Weitere Informationen finden Sie unter dem Befehl publish undErstellen 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 Dienst 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 erhalten Sie unter Greengrass-Befehlszeilenschnittstelle und Testen Sie AWS IoT Greengrass Komponenten mit lokalen Bereitstellungen. Wenn Sie das lokale Deployment erstellen, geben Sie `greengrass-build/recipes` als Ordner für Rezepte und `greengrass-build/artifacts` als Ordner für Artefakte an.

Syntax

```
$ gdk component build
```

Argumente

Keine

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 anhand des Rezepts eine neue Version der Komponente. Die GDK-CLI verwendet den S3-Bucket und die AWS Region, die Sie in der GDK-CLI-Konfigurationsdatei angeben. `gdk-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 `--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*`bucket`*`-`*`region`*`-`*`accountId`*, wo *`bucket`* und *`region`* sind die Werte, in denen Sie angebengdk-config.json, *`accountId`* ist und der Ihre AWS-Konto ID ist. Die GDK-CLI erstellt den Bucket, falls er nicht existiert.

Wenn Sie GDK CLI v1.2.0 oder höher verwenden, können Sie die in der GDK-CLI-Konfigurationsdatei AWS-Region angegebenen Werte mithilfe des Parameters überschreiben. `--region` Sie können mit dem Parameter auch zusätzliche Optionen angeben. `--options` Eine Liste der verfügbaren Optionen finden Sie unterCLI-Konfigurationsdatei für das 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 existiert. Semantische Versionen verwenden eine Hauptversion. geringfügig. 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 wurde, 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*bucket*-*region*-*accountId*, wo *bucket* und *region* sind die Werte, in denen Sie angebengdk-config.json, *accountId* ist und der Ihre AWS-Konto ID ist. Die GDK-CLI erstellt den Bucket, falls er nicht existiert.

  Die GDK-CLI erstellt den Bucket, falls er nicht existiert.

  Diese Funktion ist für GDK CLI v1.1.0 und höher verfügbar.
- `-r`, `--region` — (Optional) Geben Sie bei der Erstellung der Komponente AWS-Region den Namen des Ziels an. Dieses Argument überschreibt den Namen der Region in der GDK-CLI-Konfiguration.

  Diese Funktion ist für GDK CLI v1.2.0 und höher verfügbar.
- `-o`, `--options` (Optional) Geben Sie eine Liste mit Optionen für die Veröffentlichung 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-Softwarekatalog und Komponentenvorlagen aus dem Component AWS IoT Greengrass Templates-Repository ab. GitHub

Sie können die Ausgabe dieses Befehls an den Befehl init 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 die verfügbaren Komponentenvorlagen aufzulisten. Dieser Befehl gibt den Namen und die Sprache jeder Vorlage im Format aus*`name`*`-`*`language`*. Zum Beispiel ist in `HelloWorld-python` der Vorlagenname `HelloWorld` und die Sprache ist`python`.
- `-r`, `--repository` — (Optional) Geben Sie dieses Argument an, um die verfügbaren Repositorys für Community-Komponenten 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 im 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 GDK CLI die Maven-Modulvorlage aus dem AWS IoT Greengrass Component Templates-Repository ab. GitHub Dieses Maven-Modul ist von der JAR-Datei abhängig. `aws-greengrass-testing-standalone`

Dieser Befehl erstellt ein neues Verzeichnis, das `gg-e2e-tests` innerhalb des GDK-Projekts aufgerufen wird. Wenn das Verzeichnis des Testmoduls bereits existiert und nicht leer ist, wird der Befehl beendet, ohne etwas zu tun. Dieser `gg-e2e-tests` Ordner enthält die Cucumber-Funktion und die Schrittdefinitionen, die in einem Maven-Projekt strukturiert sind.

Standardmäßig versucht dieser Befehl, die neueste Release-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 Releases 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 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 unter der `gdk-config.json` `test-e2e` Eigenschaft angeben. Sie müssen diesen Befehl in demselben Ordner ausführen, in dem sich die `gdk-config.json` Datei befindet.

Standardmäßig verwendet GDK CLI das Maven-Build-System, um das Testmodul zu erstellen. Maven ist erforderlich, um den Befehl auszuführen. `gdk test-e2e build`

Sie müssen die Komponente erstellen, indem Sie gdk-component-build sie vor dem Erstellen des Testmoduls ausführen, 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

Keine

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 es 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 für die Ausführung 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 den in den Befehlsargumenten bereitgestellten Optionen zusammengeführt. Wenn eine Option an beiden Stellen vorhanden ist, hat die Option im Argument Vorrang vor der Option aus der Konfigurationsdatei.

Wenn die `tags` Option in diesem Befehl nicht angegeben ist, verwendet `Sample` GDK die Tags for. Wenn nicht `ggc-archive` angegeben, lädt GDK die neueste Version des Greengrass Nucleus-Archivs 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 für das Greengrass Development Kit

Das AWS IoT Greengrass Development Kit Command-Line Interface (GDK CLI) liest aus einer Konfigurationsdatei mit dem Namen Komponenten `gdk-config.json` erstellen und veröffentlichen. Diese Konfigurationsdatei muss im Stammverzeichnis des Komponenten-Repositorys vorhanden sein. Sie können den GDK CLI-Befehl init verwenden, um Komponenten-Repositorys mit dieser Konfigurationsdatei zu initialisieren.

Themen

- GDK CLI-Konfigurationsdateiformat
- Beispiele für GDK-CLI-Konfigurationsdateien

GDK CLI-Konfigurationsdateiformat

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 Releases 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 Dienst ab, um die neueste veröffentlichte Version der Komponente zu identifizieren. Anschließend wird die Version auf die nächste Patch-Version nach dieser Version gesetzt. 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 ermöglichen, müssen Sie stattdessen eine semantische Version angeben.
- Eine semantische Version, z. B. **`1.0.0`** Semantische Versionen verwenden eine Hauptversion. geringfügig. 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 den Quellcode dieser Komponente in Artefakte umzuwandeln. Dieses Objekt enthält die folgenden Informationen:

`build_system`

Das zu verwendende Build-System. Wählen Sie aus den folgenden Optionen aus:

- `zip`— Packt den Ordner der Komponente in eine ZIP-Datei, um ihn als einziges Artefakt der Komponente zu definieren. Wählen Sie diese Option für die folgenden Komponententypen:
  - Komponenten, die interpretierte Programmiersprachen verwenden, wie Python oder 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 die Namen des Komponentenordners und der ZIP-Dateien 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 (oder) `recipe.json` `recipe.yaml`
  - Erstellen Sie Ordner, wie `greengrass-build`
- `maven`— Führt den `mvn clean package` Befehl aus, um den Quellcode der Komponente in Artefakte umzuwandeln. Wählen Sie diese Option für Komponenten, die Maven verwenden, wie 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 den Quellcode der Komponente in Artefakte umzuwandeln. Wählen Sie diese Option für Komponenten, 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 den Quellcode der Komponente in Artefakte umzuwandeln. Wählen Sie diese Option für Komponenten, die den Gradle Wrapper verwenden.

  Diese Funktion ist für GDK CLI v1.2.0 und höher verfügbar.
- `custom`— Führt einen benutzerdefinierten Befehl aus, um den Quellcode der Komponente in ein Rezept und Artefakte umzuwandeln. 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**
>
> Mit diesem Befehl müssen ein Rezept und Artefakte in den folgenden Ordnern innerhalb des Komponentenordners erstellt werden. Die GDK-CLI erstellt diese Ordner für Sie, wenn Sie den Befehl component build ausführen.
>
> - Rezeptordner: `greengrass-build/recipes`
> - Ordner „Artefakte“: `greengrass-build/artifacts/`*`componentName`*`/`*`componentVersion`*
>
>   *`componentName`*Ersetzen Sie durch den Komponentennamen und *`componentVersion`* ersetzen Sie 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 im aws.greengrass.labs.LocalWebServer community component nein 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 Glob-Mustern, die definieren, welche Dateien beim Erstellen der ZIP-Datei aus dem Komponentenverzeichnis ausgeschlossen werden sollen. Nur gültig, wenn der `build_system` ist`zip`.

> **Note**
>
> In den GDK CLI-Versionen 1.4.0 und früher wird jede Datei, die einem Eintrag in der Ausnahmeliste entspricht, aus allen Unterverzeichnissen der Komponente ausgeschlossen. Um dasselbe Verhalten in den GDK-CLI-Versionen 1.5.0 und höher `**/` zu erreichen, stellen Sie sie den vorhandenen Einträgen in der Ausschlussliste voran. Schließt beispielsweise `*.txt` Textdateien nur aus dem Verzeichnis aus; schließt Textdateien aus allen `**/*.txt` Verzeichnissen und Unterverzeichnissen aus.
> In den GDK-CLI-Versionen 1.5.0 und höher wird möglicherweise während des Komponenten-Builds eine Warnung angezeigt, wenn dies in der GDK-Konfigurationsdatei definiert `excludes` ist. Um diese Warnung zu deaktivieren, setzen Sie die Umgebungsvariable auf. `GDK_EXCLUDES_WARN_IGNORE` `true`

Die GDK-CLI schließt immer die folgenden Dateien aus der Zip-Datei aus:

- Die Datei `gdk-config.json`
- Die Rezeptdatei (oder) `recipe.json` `recipe.yaml`
- Erstellen Sie Ordner, wie `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 versteckten 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 angegebenen Standarddateien und -ordner aus.

`zip_name`

Der Name der Zip-Datei, die verwendet werden soll, wenn Sie während des Build-Prozesses ein ZIP-Artefakt erstellen. Nur gültig, wenn der `build_system` ist`zip`. Wenn das leer `build_system` ist, wird der Komponentenname für den Namen der Zip-Datei verwendet.

`publish`

Die Konfiguration, die verwendet werden soll, um diese Komponente im AWS IoT Greengrass Service zu veröffentlichen.

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*`bucket`*`-`*`region`*`-`*`accountId`*, wo *`bucket`* und *`region`* sind die Werte, in denen Sie angebengdk`-config.json`, *`accountId`* ist und der Ihre AWS-Konto ID ist. Die GDK-CLI erstellt den Bucket, falls er nicht existiert.

Dieses Objekt enthält die folgenden Informationen:

`bucket`

Der S3-Bucket-Name, der zum Hosten von Komponentenartefakten verwendet werden soll.

`region`

Der AWS-Region Ort, an dem 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 mit Argumenten, 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 zum Erstellen des Testmoduls aus. Wählen Sie diese Option, um das Testmodul zu erstellen, das Maven verwendet.
- `gradle`— Führt den `gradle build` Befehl zum Erstellen des Testmoduls aus. Wählen Sie diese Option für das Testmodul, 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. Bei diesem Wert muss es sich um eine der GTF-Versionen aus Releases handeln. Die Standardeinstellung ist GTF-Version 1.1.0.

`gtf_options`

(Optional) Zusätzliche Konfigurationsoptionen, die beim end-to-end Testen der Komponente verwendet wurden.

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 spezifische regionale Endpunkte für AWS Dienste ab. Standardmäßig wird das verwendet, was das AWS SDK entdeckt.
- `credentials-path`— Optionaler Pfad für die AWS Profilanmeldedaten. Standardmäßig werden Anmeldeinformationen verwendet, die in der Host-Umgebung gefunden 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 aktiviert.
- `env-stage`— Zielt auf die Bereitstellungsumgebung von Greengrass ab. Standardmäßig auf 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`— Überschreibt die Version der Greengrass-CLI. Standardmäßig wird der Wert verwendet, der in gefunden wurde. `ggc.version`
- `gg-component-bucket`— Der Name eines vorhandenen Amazon S3 S3-Buckets, der Greengrass-Komponenten enthält.
- `gg-component-overrides`— Eine Liste von Greengrass-Komponenten-Overrides.
- `gg-persist`— Eine Liste von Testelementen, die nach einem Testlauf beibehalten werden sollen. Das Standardverhalten besteht darin, nichts beizubehalten. Zulässige Werte sind: `aws.resourcesinstalled.software`, und`generated.files`.
- `gg-runtime`— Eine Liste von Werten, die beeinflussen sollen, wie der Test mit den Testressourcen interagiert. Diese Werte haben Vorrang vor dem Parameter. `gg.persist` Wenn der Standardwert leer ist, wird davon ausgegangen, dass alle Testressourcen nach Testfällen verwaltet werden, einschließlich der installierten Greengrass-Runtime. Zulässige Werte sind: `aws.resourcesinstalled.software`, und. `generated.files`
- `ggc-archive`— Der Pfad zur archivierten Greengrass-Kernkomponente.
- `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`— Stellen Sie den Greengrass Nucleus Log Level für den Testlauf ein. Die Standardeinstellung ist „INFO“.

- `ggc-tes-rolename`— Die IAM-Rolle, die AWS IoT Greengrass Core für den Zugriff auf AWS Dienste übernehmen wird. 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, einschließlich AWS Ressourcennamen 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 JUnit resultierender 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.

Beispiele für GDK-CLI-Konfigurationsdateien

Sie können auf die folgenden Beispiele für GDK-CLI-Konfigurationsdateien verweisen, um Ihnen bei der Konfiguration von Greengrass-Komponentenumgebungen zu helfen.

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 packen, 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 packen, 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 durchsuchen.

Um die GDK-CLI-Konfigurationsdateien von Community-Komponenten anzuzeigen

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. *`community-component-name`*Ersetzen Sie es durch den Namen einer Community-Komponente aus dem vorherigen Schritt.

   ```
   https://github.com/awslabs/community-component-name
   ```

Greengrass-Befehlszeilenschnittstelle

Mit der Greengrass-Befehlszeilenschnittstelle (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 Kerngerät neu zu starten.

Stellen Sie die Greengrass-CLI-Komponente (`aws.greengrass.Cli`) bereit, um die Greengrass-CLI auf Ihrem Kerngerät zu installieren.

> **Important**
>
> Wir empfehlen, diese Komponente nur in Entwicklungsumgebungen zu verwenden, nicht in Produktionsumgebungen. 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

- Installieren Sie die Greengrass-CLI
- Greengrass-CLI-Befehle

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.

- AWS CLI Um die Greengrass-CLI bereitzustellen, müssen Sie die 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:

Um die Greengrass CLI-Komponente (Konsole) bereitzustellen

1. Melden Sie sich bei 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.
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": "2.16.1",
      "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 angeben. IDs `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 angeben. IDs `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 Kerngerät. 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 */greengrass/v2* im Ordner ausgeführt wird. Wenn Sie eine ausführbare Datei aufrufen, die sich nicht an diesem Speicherort befindet, oder wenn Sie mit der AWS IoT Greengrass Core-Software an einem anderen Speicherort interagieren möchten, müssen Sie eine der folgenden Methoden verwenden, um den Stammpfad der AWS IoT Greengrass Core-Software, mit der Sie interagieren möchten, explizit anzugeben:

- 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]
  ```

Sie können die folgenden Argumente mit jedem Befehl verwenden:

- Wird `--help` für Informationen über einen bestimmten Greengrass-CLI-Befehl verwendet.
- Wird `--version` für Informationen über die Greengrass-CLI-Version verwendet.

In diesem Abschnitt werden die Greengrass-CLI-Befehle beschrieben und Beispiele für diese Befehle bereitgestellt. Die Zusammenfassung 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 aller auf dem Gerät installierten Komponenten 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 die einzelnen 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 weitere Komponentennamen angeben, wobei Sie die einzelnen Namen durch ein Komma trennen.

Ausgabe

Keine

Bereitstellung

Verwenden Sie den `deployment` Befehl, um mit lokalen Komponenten auf Ihrem Kerngerät zu interagieren.

Verwenden Sie den `status` Unterbefehl, um den Fortschritt einer lokalen Bereitstellung zu überwachen. Sie können den Fortschritt einer lokalen Bereitstellung nicht mit der Konsole überwachen.

Unterbefehle

- create
- abbrechen
- auflisten
- Status

create

Erstellen oder aktualisieren Sie eine lokale Bereitstellung mithilfe bestimmter Komponentenrezepte, Artefakte und Laufzeitargumente.

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 Rezeptdateien der Komponenten enthält.
- `--artifactDir`,`-a`. Der vollständige Pfad zu dem Ordner, der die Artefaktdateien enthält, die Sie in Ihre Bereitstellung aufnehmen möchten. Der Ordner artefacts 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`und `RESET` unterscheiden zwischen Groß- und Kleinschreibung und müssen in Großbuchstaben geschrieben werden.
- `--groupId`,`-g`. Die Zielgruppe für den Einsatz.
- `--merge`,`-m`. Der Name und die Version der Zielkomponente, die Sie hinzufügen oder aktualisieren möchten. Sie müssen die Komponenteninformationen im folgenden Format angeben`<component>`=`<version>`. Verwenden Sie für jede weitere Komponente, die Sie angeben möchten, ein separates Argument. Verwenden Sie bei Bedarf das `--runWith` Argument, um die `posixUserposixGroup`, und `windowsUser` Informationen zum Ausführen der Komponente bereitzustellen.
- `--runWith`. Die `posixUserposixGroup`, und `windowsUser` Informationen zum Ausführen einer generischen Komponente oder einer Lambda-Komponente.

Sie müssen diese Informationen im Format *<component>*:{posixUser|windowsUser}=*<user>*[:*<=posixGroup>*] angeben. Sie könnten beispielsweise **HelloWorld:posixUser=ggc_user:ggc_group** oder angeben**HelloWorld:windowsUser=ggc_user**. Verwenden Sie für jede weitere zu spezifizierende Option ein separates Argument.

Weitere Informationen finden Sie unter Konfigurieren Sie den Benutzer, der die Komponenten ausführt.

- `--systemLimits`. Die Systemressourcenlimits, die für Prozesse generischer und nicht containerisierter Lambda-Komponenten auf dem Kerngerät gelten sollen. Sie können die maximale CPU- und RAM-Auslastung konfigurieren, die die Prozesse der einzelnen Komponenten 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 Menge an CPU-Zeit, die die Prozesse dieser Komponente 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 dieser 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 dieser 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 Komponente nicht.
- `memory`— Die maximale Menge an RAM (in Kilobyte), die die Prozesse dieser Komponente auf dem Kerngerät verwenden können.

Weitere Informationen finden Sie unter Konfigurieren Sie die Systemressourcenlimits für Komponenten.

Diese Funktion ist für Version 2.4.0 und höher der Greengrass Nucleus-Komponente und Greengrass CLI auf Linux-Core-Geräten verfügbar. AWS IoT Greengrass unterstützt diese Funktion derzeit nicht auf Windows Core-Geräten.

- `--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 Ziel-Dinggruppe 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 ergriffen wird, wenn eine Bereitstellung fehlschlägt. Es gibt zwei Aktionen, die Sie angeben können:
  - `ROLLBACK` –
  - `DO_NOTHING` –

  Diese Funktion ist für Version 2.11.0 und höher von verfügbar. Grüngraskern

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 Bereitstellung, die abgebrochen werden soll. Die Bereitstellungs-ID wird in der Ausgabe des `create` Befehls zurückgegeben.

Output

- Keine

auflisten

Ruft den Status der letzten 10 lokalen Bereitstellungen ab.

Syntax

```
greengrass-cli deployment list
```

Argumente

Keine

Ausgabe

Das folgende Beispiel zeigt die Ausgabe, die bei der Ausführung dieses Befehls erzeugt wird. Je nach 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. Je nach 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 Kerngerät zu analysieren.

Unterbefehle

- get
- Stichwörter auflisten
- list-log-files

get

Sammeln, filtern und visualisieren Sie Greengrass-Protokolldateien. Dieser Befehl unterstützt nur Protokolldateien im JSON-Format. Sie können das Logging-Format in der Nucleus-Konfiguration 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 zu dem Verzeichnis, in dem nach Protokolldateien gesucht werden soll, z. ***/greengrass/v2*****/logs** B. Nicht mit verwenden`--syslog`. Verwenden Sie für jedes weitere Verzeichnis, das Sie angeben möchten, ein separates Argument. 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 für jedes weitere Verzeichnis, das Sie angeben möchten, ein separates Argument. Sie müssen mindestens einen von `--log-dir` oder verwenden`--log-file`. Sie können auch beide Argumente in einem einzigen Befehl verwenden.
- `--follow`,`-fol`. Zeigt Protokollaktualisierungen an, sobald sie auftreten. Greengrass CLI läuft weiter und liest aus den angegebenen Protokollen. Wenn Sie ein Zeitfenster angeben, stoppt Greengrass CLI die Überwachung der Protokolle, nachdem alle Zeitfenster abgelaufen sind.
- `--filter`,`-f`. Das Schlüsselwort, die regulären Ausdrücke oder das 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 weiteren Filter, den Sie angeben möchten, ein separates Argument.

Bei der Auswertung werden mehrere Filter, die in einem einzigen Argument angegeben sind, durch OR-Operatoren getrennt, und in zusätzlichen Argumenten angegebene Filter werden mit AND-Operatoren kombiniert. Wenn Ihr Befehl beispielsweise beinhaltet`--filter "installed" --filter "name=alpha,name=beta"`, filtert und zeigt Greengrass CLI Protokollnachrichten an, die sowohl das Schlüsselwort `installed` als auch einen `name` Schlüssel mit den Werten `alpha` oder `beta` enthalten.

- `--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 folgenden Format angeben. *`<begin-time>`*`,`*`<end-time>` *Wenn Sie weder die Startzeit noch die Endzeit angeben, wird als Wert für diese Option standardmäßig das aktuelle Systemdatum und die aktuelle Systemzeit verwendet. Verwenden Sie für jedes weitere Zeitfenster, das Sie angeben möchten, ein separates Argument.

  Greengrass CLI unterstützt die folgenden Formate für Zeitstempel:

  - `yyyy-MM-DD`, zum Beispiel. `2020-06-30` Die Uhrzeit ist standardmäßig 00:00:00, wenn Sie dieses Format verwenden.

    `yyyyMMDD`, zum Beispiel. `20200630` Die Uhrzeit ist standardmäßig 00:00:00, wenn Sie dieses Format verwenden.

    `HH:mm:ss`, zum Beispiel. `15:30:45` Wenn Sie dieses Format verwenden, entspricht das Datum standardmäßig dem aktuellen Systemdatum.

    `HH:mm:ssSSS`, zum Beispiel. `15:30:45` Das Datum ist standardmäßig das aktuelle Systemdatum, wenn Sie dieses Format verwenden.

    `YYYY-MM-DD'T'HH:mm:ss'Z'`, zum Beispiel. `2020-06-30T15:30:45Z`

    `YYYY-MM-DD'T'HH:mm:ss`, zum Beispiel`2020-06-30T15:30:45`.

    `yyyy-MM-dd'T'HH:mm:ss.SSS`, zum Beispiel`2020-06-30T15:30:45.250`.

  Relative Offsets geben einen Zeitraum an, der von der aktuellen Systemzeit abweicht. 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 15 Minuten vor der aktuellen Uhrzeit. `--time-window -2h15min,-1hr`

- `--verbose`. Zeigt alle Felder aus den Protokollnachrichten an. Nicht mit verwenden`--syslog`.
- `--no-color`,`-nc`. Entfernen Sie die Farbcodierung. Die Standardfarbcodierung für Protokollnachrichten verwendet fett gedruckten roten 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`. Verarbeitet alle Protokolldateien mit dem Syslog-Protokoll, das von definiert ist. RFC3164 Nicht zusammen mit `--log-dir` und verwenden. `--verbose` Das Syslog-Protokoll verwendet das folgende Format:. `"<$Priority>$Timestamp $Host $Logger ($Class): $Message"` Wenn Sie keine Protokolldatei angeben, liest Greengrass CLI Protokollnachrichten von den folgenden Speicherorten:`/var/log/messages`,`/var/log/syslog`, oder der`/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. Die Standardeinstellung 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
```

Stichwörter auflisten

Zeigt vorgeschlagene Stichwörter an, die Sie zum Filtern von Protokolldateien verwenden können.

Syntax

```
greengrass-cli logs list-keywords [arguments]
```

Argumente

Keine

Ausgabe

Die folgenden Beispiele zeigen die Ausgabe, die bei der Ausführung dieses Befehls erzeugt wird.

```
$ 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

Zeigt Protokolldateien an, die sich in einem angegebenen Verzeichnis befinden.

Syntax

```
greengrass-cli logs list-log-files [arguments]
```

Argumente

`--log-dir,-ld`. Der Pfad zu dem Verzeichnis, in dem nach Protokolldateien gesucht werden soll.

Ausgabe

Das folgende Beispiel zeigt die Ausgabe, die bei der Ausführung dieses Befehls erzeugt wird.

```
$ 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 Sie den `get-debug-password` Befehl, um ein zufällig generiertes Passwort für die lokale Debug-Konsolenkomponente () `aws.greengrass.LocalDebugConsole` auszugeben. 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 bei der Ausführung 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 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
- Änderungsprotokoll
- Konfigurationsoptionen für das Greengrass Testing Framework
- Tutorial: Führen Sie end-to-end Tests mit dem Greengrass Testing Framework und dem Greengrass Development Kit durch
- Tutorial: Verwenden Sie einen Konfidenztest aus der Konfidenztestsuite

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-Testdatei 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 die Erstellung und Entfernung dieser Ressourcen für Sie übernehmen kann.
- `aws-greengrass-testing-platform`: Dieses Modul bietet Abstraktion auf Plattformebene für das zu testende Gerät während des Testlebenszyklus. Es APIs dient zur plattformunabhängigen Interaktion mit dem Betriebssystem und kann zur Simulation der Befehle verwendet werden, 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

- Änderungsprotokoll
- Konfigurationsoptionen für das Greengrass Testing Framework
- Tutorial: Führen Sie end-to-end Tests mit dem Greengrass Testing Framework und dem Greengrass Development Kit durch

- 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

Greengrass Testing Framework (GTF) ermöglicht es Ihnen, während des Starts des end-to-end Testprozesses bestimmte Parameter zu konfigurieren, um den Testablauf zu orchestrieren. 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 für AWS Dienste ab. Standardmäßig wird das verwendet, was das AWS SDK entdeckt.
- `credentials-path`— Optionaler Pfad für die AWS Profilanmeldedaten. Standardmäßig werden Anmeldeinformationen verwendet, die in der Host-Umgebung gefunden 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 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`— Überschreibt die Version der Greengrass-CLI. Standardmäßig wird der Wert verwendet, der in gefunden wurde. `ggc.version`

- `gg-component-bucket`— Der Name eines vorhandenen Amazon S3 S3-Buckets, der Greengrass-Komponenten enthält.
- `gg-component-overrides`— Eine Liste von Greengrass-Komponenten-Overrides.
- `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, undgenerated.files.
- `gg-runtime`— Eine Liste von Werten, die beeinflussen sollen, wie der Test mit den Testressourcen interagiert. Diese Werte haben Vorrang vor dem Parameter. `gg.persist` 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-Kernkomponente.
- `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`— Stellen Sie den Greengrass Nucleus Log Level für den Testlauf ein. Die Standardeinstellung ist „INFO“.
- `ggc-tes-rolename`— Die IAM-Rolle, die AWS IoT Greengrass Core für den Zugriff auf AWS Dienste übernehmen wird. 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`— Führt nur Feature-Tags aus. Kann mit '&' überschnitten werden

- `test-id-prefix`— Ein gemeinsames Präfix, das auf alle testspezifischen Ressourcen angewendet wird, einschließlich AWS Ressourcennamen 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 JUnit resultierender 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: Führen Sie end-to-end Tests mit dem Greengrass Testing Framework und dem Greengrass Development Kit durch

AWS IoT Greengrass Testing Framework (GTF) und Greengrass Development Kit (GDK) bieten Entwicklern Möglichkeiten, Tests durchzuführen. end-to-end 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 Testmodul. end-to-end
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
- Schritt 1: Initialisieren Sie ein GDK-Projekt mit einer Komponente
- Schritt 2: Initialisieren Sie ein GDK-Projekt mit einem Testmodul end-to-end
- Schritt 3: Erstellen Sie einen benutzerdefinierten Testfall
- Schritt 4: Fügen Sie dem neuen Testfall ein Tag hinzu
- Schritt 5: Erstellen Sie das Test-JAR
- Schritt 6: Führen Sie den Test aus
- Beispiel: Erstellen Sie einen benutzerdefinierten Testfall

Voraussetzungen

Zum Durcharbeiten dieses Tutorials ist Folgendes erforderlich:

- GDK Version 1.3.0 oder höher
- Java
- Maven
- Git

Schritt 1: Initialisieren Sie ein GDK-Projekt 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 Sie ein GDK-Projekt mit einem Testmodul end-to-end

- Mit GDK können Sie die Vorlage für das Testmodul herunterladen, die aus einer Funktion und einer schrittweisen Implementierung besteht. Führen Sie den folgenden Befehl aus, um das `HelloWorld` Verzeichnis zu öffnen und das vorhandene GDK-Projekt mithilfe eines Testmoduls 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-Testing-JAR abhängig ist.

Schritt 3: Erstellen Sie einen benutzerdefinierten Testfall

Das Schreiben eines benutzerdefinierten Testfalls besteht im Großen und Ganzen aus zwei Schritten: Erstellen Sie eine Feature-Datei mit einem Testszenario und implementieren Sie die Schrittdefinitionen. Ein Beispiel für die Erstellung eines benutzerdefinierten Testfalls finden Sie unterBeispiel: Erstellen Sie einen benutzerdefinierten Testfall. Gehen Sie wie folgt vor, um Ihren benutzerdefinierten Testfall zu erstellen:

1. Erstellen Sie eine 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, die als Szenarien bezeichnet werden, spezifiziert. 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 mit den Schlüsselwörtern „gegeben“, „wann“ und „dann“ geschrieben.
2. Implementieren Sie Schrittdefinitionen

   Eine Schrittdefinition verknüpft den Gherkin-Schritt im Klartext mit dem Programmcode. Wenn Cucumber in einem Szenario einen Gherkin-Schritt identifiziert, sucht es nach einer passenden Schrittdefinition, die ausgeführt werden kann.

Schritt 4: Fügen Sie dem neuen Testfall ein Tag hinzu

- 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 Hooks auch unter bestimmten Bedingungen auszuführen. Features und Szenarien können mehrere durch ein Leerzeichen getrennte Tags haben.

In diesem Beispiel verwenden wir die `HelloWorld` Komponente.

Fügen Sie in der Feature-Datei ein neues Tag mit dem Namen `@HelloWorld` neben dem `@Sample` Tag hinzu.

```
@Sample @HelloWorld
Scenario: As a developer, I can create a component and deploy it on my device
....
```

Schritt 5: Erstellen Sie das 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. Dieser Befehl erstellt die Test-JAR im `greengrass-build` Ordner.

   ```
   gdk test-e2e build
   ```

Schritt 6: Führen Sie den Test aus

Wenn Sie einen benutzerdefinierten Testfall ausführen, automatisiert GTF den Lebenszyklus des Tests sowie die Verwaltung der Ressourcen, die während des Tests erstellt wurden. Es stellt zunächst ein zu testendes Gerät (DUT) als AWS IoT Ding bereit und installiert die Greengrass-Kernsoftware darauf. Anschließend wird eine neue Komponente erstellt, die nach dem in diesem Pfad angegebenen Rezept benannt `HelloWorld` wird. Die `HelloWorld` Komponente wird dann über eine Greengrass-Dingbereitstellung auf dem Kerngerät bereitgestellt. Anschließend wird überprüft, ob die Bereitstellung erfolgreich ist. Der Bereitstellungsstatus wird `COMPLETED` innerhalb von 3 Minuten auf einen Wert geändert, wenn die Bereitstellung erfolgreich ist.

1. Gehen Sie zu der `gdk-config.json` Datei im Projektverzeichnis, um die Tests mit dem `HelloWorld` Tag als Ziel festzulegen. Aktualisieren Sie den `test-e2e` Schlüssel mit dem folgenden Befehl.

   ```
     "test-e2e":{
   ```

```
    "gtf_options" : {
         "tags":"HelloWorld"
     }
  }
```

2. Bevor Sie die Tests ausführen, müssen Sie AWS Anmeldeinformationen für das Hostgerät angeben. GTF verwendet diese Anmeldeinformationen, um die AWS Ressourcen während des Testprozesses zu verwalten. Stellen Sie sicher, dass die von Ihnen angegebene Rolle über die erforderlichen Berechtigungen verfügt, um die im Test enthaltenen erforderlichen Vorgänge zu automatisieren.

   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
   ```

   Mit diesem Befehl wird die neueste Version von Greengrass Nucleus in den `greengrass-build` Ordner heruntergeladen und Tests damit ausgeführt. Dieser Befehl zielt auch nur auf die Szenarien ab, die mit dem `HelloWorld` Tag gekennzeichnet sind, und generiert einen Bericht für diese Szenarien. Sie werden sehen, dass die AWS Ressourcen, die während dieses Tests erstellt wurden, am Ende des Tests verworfen werden.

Beispiel: Erstellen Sie einen benutzerdefinierten Testfall

Example

Das heruntergeladene Testmodul im GDK-Projekt besteht aus einer Beispielfunktion und einer Datei zur schrittweisen Implementierung.

Im folgenden Beispiel erstellen wir eine Feature-Datei zum Testen der Ding-Deployment-Funktion der Greengrass-Software. Wir testen die Funktionalität dieser Funktion teilweise mit einem Szenario, das die Bereitstellung einer Komponente über Greengrass AWS Cloud durchführt. Dies ist eine Reihe von Schritten, die uns helfen, die Interaktionen und die erwarteten Ergebnisse dieses Anwendungsfalls zu verstehen.

1. Erstellen Sie eine Feature-Datei

   Navigieren Sie zu dem `gg-e2e-tests/src/main/resources/greengrass/features` Ordner im aktuellen Verzeichnis. Sie können das Beispiel finden`component.feature`, das wie das folgende Beispiel aussieht.

   In dieser Feature-Datei können Sie die Thing-Bereitstellungsfunktion der Greengrass-Software testen. Sie können die Funktionalität dieser Funktion teilweise mit einem Szenario testen, das eine Bereitstellung einer Komponente über die Greengrass-Cloud durchführt. Das Szenario besteht aus einer Reihe von Schritten, die dazu beitragen, 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 genannten Schritts:`And I call my custom step`.

2. Implementieren Sie Schrittdefinitionen

   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, Tests durchzuführen. end-to-end 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 Konfidenztest aus der Confidence Test Suite zu verwenden. Nachdem Sie Ihren benutzerdefinierten Testfall erstellt haben, können Sie den Test ausführen.

Ein Konfidenztest ist ein generischer Test von Greengrass, der das Verhalten grundlegender Komponenten validiert. Diese Tests können modifiziert oder erweitert werden, um spezifischeren Anforderungen der Komponenten gerecht zu werden.

Für dieses Tutorial werden wir eine HelloWorld Komponente verwenden. 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 Testmodul. end-to-end
3. Verwenden Sie einen Test aus der Confidence Test Suite.
4. Fügen Sie dem neuen Testfall ein Tag hinzu.
5. Erstellen Sie das Test-JAR.
6. Führen Sie den Test aus.

Themen

- Voraussetzungen
- Schritt 1: Initialisieren Sie ein GDK-Projekt mit einer Komponente
- Schritt 2: Initialisieren Sie ein GDK-Projekt mit einem Testmodul end-to-end
- Schritt 3: Verwenden Sie einen Test aus der Confidence Test Suite
- Schritt 4: Fügen Sie dem neuen Testfall ein Tag hinzu
- Schritt 5: Erstellen Sie das Test-JAR
- Schritt 6: Führen Sie den Test aus
- Beispiel: Verwenden Sie einen Konfidenztest

Voraussetzungen

Zum Durcharbeiten dieses Tutorials ist Folgendes erforderlich:

- GDK Version 1.6.0 oder höher
- Java
- Maven
- Git

Schritt 1: Initialisieren Sie ein GDK-Projekt 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 Sie ein GDK-Projekt mit einem Testmodul end-to-end

- Mit GDK können Sie die Vorlage für das Testmodul herunterladen, die aus einer Funktion und einer schrittweisen Implementierung besteht. Führen Sie den folgenden Befehl aus, um das `HelloWorld` Verzeichnis zu öffnen und das vorhandene GDK-Projekt mithilfe eines Testmoduls 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-Testing-JAR abhängig ist.

Schritt 3: Verwenden Sie einen Test aus der Confidence Test Suite

Das Schreiben eines Konfidenztestfalls besteht darin, die bereitgestellte Feature-Datei zu verwenden und, falls erforderlich, die Szenarien zu ändern. Ein Beispiel für die Verwendung eines Konfidenztests finden Sie unterBeispiel: Erstellen Sie einen benutzerdefinierten Testfall. Gehen Sie wie folgt vor, um einen Konfidenztest 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 Konfidenztest zu verwenden.

Schritt 4: Fügen Sie dem neuen Testfall ein Tag hinzu

- 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 Hooks auch unter bestimmten Bedingungen auszuführen. Features und Szenarien können mehrere durch ein Leerzeichen getrennte Tags haben.

  In diesem Beispiel verwenden wir die `HelloWorld` Komponente.

  Jedes Szenario ist mit gekennzeichnet`@ConfidenceTest`. Ändern oder fügen Sie Tags hinzu, wenn Sie nur einen Teil der Testsuite ausführen möchten. Jedes Testszenario wird oben in jedem Konfidenztest beschrieben. Das Szenario besteht aus einer Reihe von Schritten, die dazu beitragen, die Interaktionen und erwarteten Ergebnisse der einzelnen Testfälle 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 Sie das 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: Führen Sie den Test aus

Wenn Sie einen Konfidenztest durchführen, automatisiert der GTF den Lebenszyklus des Tests sowie die Verwaltung der Ressourcen, die während des Tests erstellt wurden. Es stellt zunächst ein zu testendes Gerät (DUT) als AWS IoT Ding bereit und installiert die Greengrass-Kernsoftware darauf. Anschließend wird eine neue Komponente erstellt, die nach dem in diesem Pfad angegebenen Rezept benannt `HelloWorld` wird. Die `HelloWorld` Komponente wird dann über eine Greengrass-Dingbereitstellung auf dem Kerngerät bereitgestellt. Anschließend wird überprüft, ob die Bereitstellung erfolgreich ist. Der Bereitstellungsstatus wird `COMPLETED` innerhalb von 3 Minuten auf einen Wert geändert, wenn die Bereitstellung erfolgreich ist.

1. Gehen Sie zu der `gdk-config.json` Datei im Projektverzeichnis, um die Tests mit dem `ConfidenceTest` Tag oder dem in Schritt 4 angegebenen Tag als Ziel festzulegen. Aktualisieren Sie den `test-e2e` Schlüssel mit dem folgenden Befehl.

   ```
   "test-e2e":{
     "gtf_options" : {
          "tags":"ConfidenceTest"
      }
   }
   ```

2. Bevor Sie die Tests ausführen, müssen Sie AWS Anmeldeinformationen für das Hostgerät angeben. GTF verwendet diese Anmeldeinformationen, um die AWS Ressourcen während des Testprozesses zu verwalten. Stellen Sie sicher, dass die von Ihnen angegebene Rolle über die erforderlichen Berechtigungen verfügt, um die im Test enthaltenen erforderlichen Vorgänge zu automatisieren.

   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
   ```

   Mit diesem Befehl wird die neueste Version von Greengrass Nucleus in den `greengrass-build` Ordner heruntergeladen und Tests damit ausgeführt. Dieser Befehl zielt auch nur auf die Szenarien ab, die mit dem `ConfidenceTest` Tag gekennzeichnet sind, und generiert einen Bericht für diese Szenarien. Sie werden sehen, dass die AWS Ressourcen, die während dieses Tests erstellt wurden, am Ende des Tests verworfen werden.

Beispiel: Verwenden Sie einen Konfidenztest

Example

Das heruntergeladene Testmodul im GDK-Projekt besteht aus einer bereitgestellten Feature-Datei.

Im folgenden Beispiel verwenden wir eine Feature-Datei, um die Ding-Deployment-Funktion der Greengrass-Software zu testen. Wir testen die Funktionalität dieser Funktion teilweise mit einem Szenario, das die Bereitstellung einer Komponente über Greengrass AWS Cloud durchführt. 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 zu dem `gg-e2e-tests/src/main/resources/greengrass/features` Ordner im aktuellen Verzeichnis. Sie können das Beispiel finden`confidenceTest.feature`, das wie das folgende 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 Konfidenztest beschrieben. Das Szenario besteht aus einer Reihe von Schritten, die dazu beitragen, die Interaktionen und erwarteten Ergebnisse der einzelnen Testfälle 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.

AWS IoT Greengrass Komponenten entwickeln

Sie können Komponenten auf Ihrem Greengrass-Core-Gerät entwickeln und testen. Dadurch können Sie Ihre AWS IoT Greengrass Software erstellen und iterieren, ohne mit dem zu interagieren. AWS Cloud Wenn Sie eine Version Ihrer Komponente fertiggestellt haben, können Sie sie AWS IoT Greengrass in die Cloud hochladen, sodass Sie und Ihr Team die Komponente auf anderen Geräten in Ihrer Flotte bereitstellen können. Weitere Informationen zur Bereitstellung von Komponenten finden Sie unterAWS IoT Greengrass Komponenten auf Geräten bereitstellen.

Jede Komponente besteht aus einem Rezept und Artefakten.

- Rezepte

  Jede Komponente enthält eine Rezeptdatei, die ihre Metadaten definiert. Das Rezept spezifiziert auch die Konfigurationsparameter, die Komponentenabhängigkeiten, den Lebenszyklus und die Plattformkompatibilität der Komponente. Der Komponentenlebenszyklus definiert die Befehle, mit denen die Komponente installiert, ausgeführt und heruntergefahren wird. 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 umfassen, die eine Komponente verwendet. Komponenten können auch Artefakte aus Komponentenabhängigkeiten verwenden.

AWS IoT Greengrass stellt vorgefertigte Komponenten bereit, die Sie in Ihren Anwendungen verwenden und auf Ihren Geräten bereitstellen können. Sie können beispielsweise die Stream-Manager-Komponente verwenden, um Daten in verschiedene AWS Dienste hochzuladen, oder Sie können die CloudWatch Metrik-Komponente verwenden, um benutzerdefinierte Metriken auf Amazon zu veröffentlichen CloudWatch. Weitere Informationen finden Sie unter AWS-mitgelieferte Komponenten.

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.

Die AWS IoT Greengrass Core-Software führt Komponenten, die Sie auf dem Core-Gerät konfigurieren, als Systembenutzer `ggc_user` und als Gruppe aus`ggc_group`, z. B. und. Das bedeutet, dass die 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 beispielsweise nicht verwenden können, um Python-Pakete zu installieren. Wenn Sie das Tutorial „Erste Schritte“ befolgt haben, um Ihr Kerngerät einzurichten, hat Ihr Systembenutzer kein Home-Verzeichnis. Weitere Informationen zur Konfiguration des Benutzers und der Gruppe, die Komponenten ausführen, finden Sie unterKonfigurieren Sie den Benutzer, der die Komponenten ausführt.

> **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.

Themen

- Lebenszyklus der Komponenten
- Komponententypen
- AWS IoT Greengrass Komponenten erstellen
- Testen Sie AWS IoT Greengrass Komponenten mit lokalen Bereitstellungen
- Veröffentlichen Sie Komponenten zur Bereitstellung auf Ihren Kerngeräten
- Interagieren Sie mit AWS Diensten
- Führen Sie einen Docker-Container aus
- AWS IoT Greengrass Referenz zum Komponenten-Rezept
- Referenz zur Umgebungsvariablen der Komponente

Lebenszyklus der Komponenten

Der Komponentenlebenszyklus definiert die Phasen, in denen die AWS IoT Greengrass Core-Software Komponenten installiert und ausführt. Jede Phase definiert ein Skript und andere Informationen, die das Verhalten der Komponente spezifizieren. Wenn Sie beispielsweise eine Komponente installieren, führt die AWS IoT Greengrass Core-Software das `install` Lifecycle-Skript für diese Komponente aus. Komponenten auf Kerngeräten haben die folgenden Lebenszyklusstatus:

- `NEW`— Das Rezept und die Artefakte der Komponente sind auf das Kerngerät geladen, aber die Komponente ist nicht installiert. Nachdem eine Komponente diesen Status erreicht hat, führt sie ihr Installationsskript aus.
- `INSTALLED`— Die Komponente ist auf dem Kerngerät installiert. Die Komponente wechselt in diesen Status, nachdem sie ihr Installationsskript ausgeführt hat.
- `STARTING`— Die Komponente wird auf dem Kerngerät gestartet. Die Komponente wechselt in diesen Zustand, wenn sie ihr Startskript ausführt. Wenn der Start erfolgreich ist, wechselt die Komponente in den `RUNNING` Status.
- `RUNNING`— Die Komponente läuft auf dem Kerngerät. Die Komponente wechselt in diesen Zustand, wenn sie ihr Run-Skript ausführt oder wenn sie aktive Hintergrundprozesse in ihrem Startskript ausführt.
- `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 übergeht, führt sie ihr Wiederherstellungsskript aus. Anschließend wird die Komponente neu gestartet, um zu versuchen, zur normalen Verwendung zurückzukehren. Wenn die Komponente dreimal in den `ERRORED` Status wechselt, ohne dass sie erfolgreich ausgeführt wurde, wird `BROKEN` die Komponente.
- `BROKEN`— Die Komponente ist mehrfach auf Fehler gestoßen und kann nicht wiederhergestellt werden. Sie müssen die Komponente erneut bereitstellen, um das Problem zu beheben.

Komponententypen

Der Komponententyp gibt an, wie die AWS IoT Greengrass Core-Software die Komponente ausführt. Komponenten können die folgenden Typen haben:

- Kern (`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.

- Plug-in () `aws.greengrass.plugin`

  Der Greengrass-Nucleus führt eine Plugin-Komponente in derselben Java Virtual Machine (JVM) wie der Nucleus aus. Der Nucleus 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-Nucleus so konfigurieren, dass er als Systemdienst ausgeführt wird. Weitere Informationen finden Sie unter Den Greengrass Nucleus als Systemdienst konfigurieren.

  Einige Komponenten, die von bereitgestellt werden, AWS sind Plugin-Komponenten, sodass sie direkt mit dem Greengrass-Nucleus verbunden werden können. Plugin-Komponenten verwenden dieselbe Protokolldatei wie der Greengrass-Kern. Weitere Informationen finden Sie unter AWS IoT Greengrass Protokolle überwachen.

- Generisch () `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 mithilfe der Lambda-Launcher-Komponente eine Lambda-Funktionskomponente aus.

  Wenn Sie eine Komponente aus einer Lambda-Funktion erstellen, hat die Komponente diesen Typ. Weitere Informationen finden Sie unter AWS Lambda Funktionen ausführen.

> **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.

AWS IoT Greengrass Komponenten erstellen

Sie können benutzerdefinierte AWS IoT Greengrass Komponenten auf einem lokalen Entwicklungscomputer oder einem Greengrass-Core-Gerät entwickeln. AWS IoT Greengrass bietet das AWS IoT Greengrass Development Kit Command-Line Interface (GDK CLI), mit dem Sie Komponenten aus vordefinierten Komponentenvorlagen und Community-Komponenten erstellen, erstellen und veröffentlichen können. 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 Sie das Greengrass Development Kit CLI

  Verwenden Sie die GDK-CLI, um Komponenten auf einem lokalen Entwicklungscomputer zu entwickeln. Die GDK-CLI erstellt und verpackt den Quellcode von 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 Version und das Artefakt der Komponente automatisch aktualisiert werden, URIs 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 für das Entwicklungskit.
- Führen Sie die integrierten Shell-Befehle aus

  Sie können integrierte Shell-Befehle ausführen, um Komponenten auf einem lokalen Entwicklungscomputer oder auf einem Greengrass-Core-Gerät zu entwickeln. Sie verwenden Shell-Befehle, um den Quellcode von Komponenten zu kopieren oder in Artefakte umzuwandeln. 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 für jedes Komponentenartefakt im Rezept aktualisieren.

Themen

- Eine Komponente erstellen (GDK CLI)
- Erstellen Sie eine Komponente (Shell-Befehle)

Eine Komponente erstellen (GDK CLI)

Folgen Sie den Anweisungen in diesem Abschnitt, um eine Komponente mit der GDK-CLI zu erstellen und zu erstellen.

Um eine Greengrass-Komponente (GDK CLI) zu entwickeln

1. Falls Sie es noch nicht getan haben, installieren Sie die GDK-CLI auf Ihrem Entwicklungscomputer. Weitere Informationen finden Sie unter Installieren oder aktualisieren Sie die 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 Community-Komponente herunter, sodass Sie mit einem Funktionsbeispiel beginnen können. Verwenden Sie den Befehl component list, 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 die Community-Komponenten aufzulisten.

     ```
     gdk component list --repository
     ```

4. Erstellen Sie einen Komponentenordner, in den die GDK-CLI die Vorlage oder Community-Komponente herunterlädt, und wechseln Sie zu diesem. *HelloWorld*Ersetzen Sie ihn 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 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. *HelloWorld*Ersetzen Sie ihn durch den Namen der Vorlage und *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. *ComponentName*Ersetzen Sie ihn 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 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 existiert. 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 benannten GDK-CLI-Konfigurationsdatei, um `gdk-config.json` Komponenten zu erstellen und zu veröffentlichen. Diese Konfigurationsdatei befindet sich im Stammverzeichnis des Komponentenordners. Im vorherigen Schritt wurde 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 Dienst ab, um die neueste veröffentlichte Version der Komponente zu identifizieren. Anschließend wird die Version auf die nächste Patch-Version nach dieser Version gesetzt. 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 ermöglichen, müssen Sie stattdessen eine semantische Version angeben.

- Eine semantische Version, wie z. **`1.0.0`** Semantische Versionen verwenden eine Hauptversion. unbedeutend. 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 aufbaut. Wählen Sie aus den folgenden Optionen für`build_system`:

- `zip`— Packt den Ordner der Komponente in eine ZIP-Datei, um ihn als einziges Artefakt der Komponente zu definieren. Wählen Sie diese Option für die folgenden Komponententypen:
  - Komponenten, die interpretierte Programmiersprachen verwenden, wie Python oder 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 die Namen des Komponentenordners und der ZIP-Dateien 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 (oder) `recipe.json` `recipe.yaml`
  - Erstellen Sie Ordner, wie `greengrass-build`
- `maven`— Führt den `mvn clean package` Befehl aus, um den Quellcode der Komponente in Artefakte umzuwandeln. Wählen Sie diese Option für Komponenten, die Maven verwenden, wie 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 den Quellcode der Komponente in Artefakte umzuwandeln. Wählen Sie diese Option für Komponenten, 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 den Quellcode der Komponente in Artefakte umzuwandeln. Wählen Sie diese Option für Komponenten, die den Gradle Wrapper verwenden.

  Diese Funktion ist für GDK CLI v1.2.0 und höher verfügbar.
- `custom`— Führt einen benutzerdefinierten Befehl aus, um den Quellcode der Komponente in ein Rezept und Artefakte umzuwandeln. Geben Sie den benutzerdefinierten Befehl im `custom_build_command` Parameter an.

f. Wenn Sie `custom` für angeben`build_system`, fügen Sie `custom_build_command` dem `build` Objekt das hinzu. Geben Sie in `custom_build_command` 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 Folgendes **`["cmake", "--build", "build", "--config", "Release"]`** angeben:

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*`bucket`*`-`*`region`*`-`*`accountId`*, wo *`bucket`* und *`region`* sind die Werte, in denen Sie angeben`gdk-config.json`, *`accountId`* ist und der Ihre AWS-Konto ID ist. Die GDK-CLI erstellt den Bucket, falls er nicht existiert.

   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 an AWS-Region , wo die GDK-CLI die Komponente veröffentlicht.

Wenn Sie mit diesem Schritt fertig sind, sieht die `gdk-config.json` Datei möglicherweise dem folgenden Beispiel ähnlich.

```
{
  "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 Rezeptdatei der Komponente 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 Name der Zip-Datei mit dem Namen des Komponentenordners übereinstimmt. In den folgenden unvollständigen Rezeptbeispielen wird der Name der Zip-Datei 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 Komponentenbeschreibung, die Standardkonfiguration, die Artefakte, Lebenszyklusskripts 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 bauen 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
   ```

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-Kerngeräten bereitstellen. Weitere Informationen finden Sie unter Veröffentlichen Sie Komponenten zur Bereitstellung auf Ihren Kerngeräten.

Erstellen Sie eine Komponente (Shell-Befehle)

Folgen Sie den Anweisungen in diesem Abschnitt, um Rezept- und Artefaktordner zu erstellen, die Quellcode und Artefakte für mehrere Komponenten enthalten.

Um eine Greengrass-Komponente zu entwickeln (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 zu dem 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 Plattformfähigkeit Ihrer Komponente definiert. Nehmen Sie die Komponentenversion in den Namen der Rezeptdatei auf, damit Sie erkennen können, welches Rezept welche Komponentenversion widerspiegelt. Sie können das YAML- oder das JSON-Format für Ihr Rezept wählen.

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

   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 Hauptteil. unbedeutend. Patch-Nummernsystem. Version `1.0.0`

steht beispielsweise für die erste Hauptversion einer Komponente. 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. Es wird empfohlen, für die Artefakte jeder Komponentenversion einen eigenen Ordner zu verwenden, damit Sie feststellen können, welche Artefakte für jede Komponentenversion bestimmt sind. 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 Artefaktordnerpfad verwenden. Geben Sie den Komponentennamen und die Version an, 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 alle anderen Binärdateien umfassen, die Ihre Komponente verwendet.

   Wenn Ihre Komponente bereit ist, testen Sie Ihre Komponente.

Testen Sie AWS IoT Greengrass Komponenten mit lokalen Bereitstellungen

Wenn Sie eine Greengrass-Komponente auf einem Kerngerät entwickeln, können Sie eine lokale Bereitstellung erstellen, um sie zu installieren und zu testen. Folgen Sie den Schritten in diesem Abschnitt, um eine lokale Bereitstellung zu erstellen.

Wenn Sie die Komponente auf einem anderen Computer entwickeln, z. B. auf einem lokalen Entwicklungscomputer, können Sie keine lokale Bereitstellung erstellen. Veröffentlichen Sie die Komponente stattdessen im AWS IoT Greengrass Service, sodass Sie sie auf Greengrass-Core-Geräten bereitstellen können, um sie zu testen. Weitere Informationen erhalten Sie unter

Veröffentlichen Sie Komponenten zur Bereitstellung auf Ihren Kerngeräten und AWS IoT Greengrass Komponenten auf Geräten bereitstellen.

Um eine Komponente auf einem Greengrass-Core-Gerät zu testen

1. Das Kerngerät protokolliert Ereignisse wie Komponenten-Updates. Sie können diese Protokolldatei einsehen, um Fehler an Ihrer Komponente zu erkennen und zu beheben, z. B. ein ungültiges Rezept. In dieser Protokolldatei werden auch Meldungen angezeigt, die Ihre Komponente als Standardausgabe (stdout) ausgibt. Wir empfehlen Ihnen, eine zusätzliche Terminalsitzung auf Ihrem Kerngerät zu eröffnen, um neue Protokollnachrichten in Echtzeit zu beobachten. Öffnen Sie eine neue Terminalsitzung, z. B. über SSH, und führen Sie den folgenden Befehl aus, um die Protokolle anzuzeigen. */greengrass/v2*Ersetzen Sie es durch 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 einsehen.

   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 Kerngerät mit Ihrer Komponente zu aktualisieren. */greengrass/v2*Ersetzen Sie durch den Pfad zum AWS IoT Greengrass Stammordner und *~/greengrassv2* ersetzen Sie ihn 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 den `greengrass-cli deployment create` Befehl auch 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-Core-Gerät läuft. Wenn Sie diese Version Ihrer Komponente fertiggestellt haben, können Sie sie in den AWS IoT Greengrass Dienst 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::ComponentVersion im 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 das 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*bucket*`-`*region*`-`*accountId*, wo *bucket* und *region* sind die Werte, in denen Sie angebengdk`-config.json`, *accountId* ist und der Ihre AWS-Konto ID ist. 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 AWS IoT Greengrass Komponenten auf Geräten bereitstellen.

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 eindeutig 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. unbedeutend. 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 amzn-s3-demo-bucket durch den Namen des Buckets und durch *`artifacts/com.example.HelloWorld/1.0.0/artifact.py`* den Pfad zur Artefaktdatei.

      ```
      aws s3 cp artifacts/com.example.HelloWorld/1.0.0/artifact.py s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/artifact.py
      ```

> **Important**
>
> Core-Geräterollen ermöglichen 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. Der Amazon S3 Ihrer Artefakte URIs sollte dem folgenden Beispiel ähneln.

```
s3://amzn-s3-demo-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://amzn-s3-demo-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://amzn-s3-demo-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. *path/to/recipeFile*Ersetzen Sie es 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* durch die abzufragende 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 AWS IoT Greengrass Komponenten auf Geräten bereitstellen.

Interagieren Sie mit AWS Diensten

Greengrass-Core-Geräte verwenden X.509-Zertifikate, um AWS IoT Core mithilfe von TLS-Protokollen für die gegenseitige Authentifizierung eine Verbindung herzustellen. Mit diesen Zertifikaten können Geräte AWS IoT ohne AWS Anmeldeinformationen interagieren, die in der Regel aus einer Zugriffsschlüssel-ID und einem geheimen Zugriffsschlüssel bestehen. Andere AWS Dienste benötigen AWS Anmeldeinformationen anstelle von X.509-Zertifikaten, um API-Operationen an Dienstendpunkten aufzurufen. AWS IoT Core hat einen Anbieter für Anmeldeinformationen, der es Geräten ermöglicht, ihr X.509-Zertifikat zur Authentifizierung von Anfragen zu verwenden. AWS Der Anbieter AWS IoT für Anmeldeinformationen authentifiziert Geräte mithilfe eines X.509-Zertifikats und gibt AWS Anmeldeinformationen in Form eines temporären Sicherheitstokens mit eingeschränkten Rechten aus. Geräte können dieses Token verwenden, um jede Anfrage zu signieren und zu authentifizieren. AWS Dadurch entfällt die Notwendigkeit, AWS Anmeldeinformationen auf Greengrass-Core-Geräten zu speichern. Weitere Informationen finden Sie unter Autorisieren von direkten Aufrufen von AWS Diensten im AWS IoT Core Entwicklerhandbuch.

Um Anmeldeinformationen von Greengrass abzurufen AWS IoT, verwenden Kerngeräte einen AWS IoT Rollenalias, der auf eine IAM-Rolle verweist. Diese IAM-Rolle wird Token-Exchange-Rolle genannt. Sie erstellen den Rollenalias und die Token-Austauschrolle, wenn Sie die AWS IoT Greengrass Core-Software installieren. Um den Rollenalias anzugeben, den ein Core-Gerät verwendet, konfigurieren Sie den `iotRoleAlias` Parameter vonGrüngraskern.

Der Anbieter AWS IoT für Anmeldeinformationen übernimmt in Ihrem Namen die Rolle des Tokenaustauschs, um AWS Anmeldeinformationen für Kerngeräte bereitzustellen. Sie können dieser Rolle entsprechende IAM-Richtlinien zuordnen, um Ihren Kerngeräten den Zugriff auf Ihre AWS Ressourcen zu ermöglichen, z. B. auf Komponentenartefakte in S3-Buckets. Weitere Informationen zur Konfiguration der Token-Austauschrolle 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 den UpdateRoleAliasVorgang verwenden, um die Gültigkeitsdauer der Anmeldeinformationen zu konfigurieren.

AWS IoT Greengrass stellt eine öffentliche Komponente bereit, die Token-Exchange-Dienstkomponente, die Sie als Abhängigkeit in Ihrer benutzerdefinierten Komponente definieren können, um mit AWS Diensten zu interagieren. Der Token-Exchange-Dienst 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-Anfragen. Auf diese Weise können Sie andere Tools verwenden AWS SDKs , um AWS Dienste in Ihren Komponenten aufzurufen. Weitere Informationen finden Sie unter Token-Austauschdienst.

> **Important**
>
> Die Support für den Erwerb von AWS Anmeldeinformationen auf diese Weise wurde 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 Developer Guide.

Um AWS Anmeldeinformationen in Ihrer benutzerdefinierten Komponente abzurufen, definieren Sie `aws.greengrass.TokenExchangeService` sie als Abhängigkeit im Komponentenrezept. Das folgende Beispielrezept definiert eine Komponente, die boto3 installiert und ein Python-Skript ausführt, das AWS Anmeldeinformationen vom Token-Exchange-Service verwendet, um Amazon S3 S3-Buckets aufzulisten.

> **Note**
>
> Um diese Beispielkomponente ausführen zu können, muss Ihr Gerät über die entsprechende Berechtigung verfügen. `s3:ListAllMyBuckets` 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 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
- Führen Sie einen Docker-Container von einem öffentlichen Image in Amazon ECR oder Docker Hub aus
- Führen Sie einen Docker-Container von einem privaten Image in Amazon ECR aus
- Führen Sie einen Docker-Container von einem Image in Amazon S3 aus
- 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)

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 das Bild auch URIs als Komponentenartefakte hinzu. Das Bild URIs muss das Format haben, `docker:`*`registry`*`/`*`image`*`[`*`:tag|@digest`*`]` 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.

  JSON

  ```
  {
    "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.

  JSON

  ```
  {
    "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://amzn-s3-demo-bucket/folder/docker-compose.yaml"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.MyDockerComposeComponent
ComponentVersion: '1.0.0'
ComponentDescription: 'A component that uses Docker Compose to run images from
 public Amazon ECR and Docker Hub.'
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.DockerApplicationManager:
    VersionRequirement: ~2.0.0
Manifests:
```

```
  - Platform:
      os: all
    Lifecycle:
        Run: docker-compose -f {artifacts:path}/docker-compose.yaml up
    Artifacts:
      - URI: "docker:public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest"
      - URI: "docker:mysql:8.0"
      - URI: "s3://amzn-s3-demo-bucket/folder/docker-compose.yaml"
```

> **Note**
>
> 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*hello-world*Ersetzen Sie es 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://amzn-s3-demo-bucket/artifacts/
com.example.MyDockerComponent/1.0.0/hello-world.tar"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.MyS3DockerComponent
ComponentVersion: '1.0.0'
ComponentDescription: 'A component that runs a Docker container from an image in
 an S3 bucket.'
ComponentPublisher: Amazon
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install:
        Script: docker load -i {artifacts:path}/hello-world.tar
      Run:
        Script: docker run --rm hello-world
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.MyDockerComponent/1.0.0/hello-world.tar
```

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://amzn-s3-demo-bucket/artifacts/
com.example.python.docker.PublishToIoTCore/1.0.0/publish-to-iot-core.tar"
        }
      ]
    }
  ]
}
```

YAML

```
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.python.docker.PublishToIoTCore
ComponentVersion: 1.0.0
ComponentDescription: Uses interprocess communication to publish an MQTT message to
 IoT Core.
```

```
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    topic: 'test/topic/java'
    message: 'Hello, World!'
    qos: '1'
    accessControl:
      aws.greengrass.ipc.mqttproxy:
        'com.example.python.docker.PublishToIoTCore:pubsub:1':
          policyDescription: Allows access to publish to IoT Core on all topics.
          operations:
            - 'aws.greengrass#PublishToIoTCore'
          resources:
            - '*'
Manifests:
  - Platform:
      os: all
    Lifecycle:
      install: 'docker load -i {artifacts:path}/publish-to-iot-core.tar'
      Run: |
        docker run \
          -v $AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT:
$AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT \
          -e SVCUID \
          -e AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT \
          -e MQTT_TOPIC="{configuration:/topic}" \
          -e MQTT_MESSAGE="{configuration:/message}" \
          -e MQTT_QOS="{configuration:/qos}" \
          --rm publish-to-iot-core
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.python.docker.PublishToIoTCore/1.0.0/publish-to-iot-core.tar
```

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 Sie mit AWS Diensten.

> **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 Sie diese Umgebungsvariablen, um AWS Anmeldeinformationen abzurufen.

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://amzn-s3-demo-bucket/artifacts/
com.example.python.docker.ListS3Buckets/1.0.0/list-s3-buckets.tar"
        }
      ]
    }
  ]
```

```
}
```

YAML

```
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.python.docker.ListS3Buckets
ComponentVersion: 1.0.0
ComponentDescription: Uses the token exchange service to lists your S3 buckets.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.TokenExchangeService:
    VersionRequirement: ^2.0.0
    DependencyType: HARD
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: 'docker load -i {artifacts:path}/list-s3-buckets.tar'
      Run: |
        docker run \
          --network=host \
          -e AWS_CONTAINER_AUTHORIZATION_TOKEN \
          -e AWS_CONTAINER_CREDENTIALS_FULL_URI \
          --rm list-s3-buckets
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.python.docker.ListS3Buckets/1.0.0/list-s3-buckets.tar
```

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 Datenströme auf Greengrass-Kerngeräten verwalten.

> **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://amzn-s3-demo-bucket/artifacts/
com.example.python.docker.StreamFileToS3/1.0.0/stream-file-to-s3.tar"
        }
      ]
    }
  ]
}
```

YAML

```
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.python.docker.StreamFileToS3
ComponentVersion: 1.0.0
ComponentDescription: Creates a text file and uses stream manager to stream the file
 to S3.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.StreamManager:
    VersionRequirement: ^2.0.0
    DependencyType: HARD
ComponentConfiguration:
  DefaultConfiguration:
    bucketName: ''
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: 'docker load -i {artifacts:path}/stream-file-to-s3.tar'
      Run: |
        docker run \
          --network=host \
          -e AWS_CONTAINER_AUTHORIZATION_TOKEN \
          -e BUCKET_NAME="{configuration:/bucketName}" \
```

```
          -e WORK_PATH="{work:path}" \
          -v {work:path}:{work:path} \
          --rm stream-file-to-s3
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.python.docker.StreamFileToS3/1.0.0/stream-file-to-s3.tar
```

AWS IoT Greengrass 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
- Rezeptformat
- Rezeptvariablen
- Beispiele für Rezepte

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 für die Komponente bereitgestellt 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 AWS Lambda Funktionen ausführen.
- `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 AWS IoT Greengrass Protokolle überwachen.

  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. AWS-mitgelieferte 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 AWS Lambda Funktionen ausführen.

`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, wie z. `/windows|linux/` Der reguläre Ausdruck muss mit einem Schrägstrich () beginnen und enden. `/` Der reguläre Ausdruck `/.+/` entspricht beispielsweise jedem Wert, der kein Leerzeichen ist.

Dieses Objekt enthält die folgenden Informationen:

`runtime`

Die Greengrass Nucleus-Laufzeit für die Plattform, die dieses Manifest unterstützt. Wenn mehrere Manifeste mit der Plattform definiert werden`runtime`, sind die unterstützten Laufzeitwerte in einem Rezept `aws_nucleus_lite` und `*` nur. Um auf ein klassisches Gerät abzuzielen, DARF das Laufzeitfeld NICHT im Rezept angegeben werden. Zu den unterstützten Greengrass Nucleus-Laufzeiten gehören die folgenden Werte:

- `*`
- `aws_nucleus_lite`

`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. *`Key`*Ersetzen Sie es 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 startenstartup, 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 setzentrue, 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`— Anfrage 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://amzn-s3-demo-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://amzn-s3-demo-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://amzn-s3-demo-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:

- Lebenszyklusdefinitionen.
- 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://amzn-s3-demo-bucket/hello_world.zip",
        "Unarchive": "ZIP"
      },
      {
        "Uri": "s3://amzn-s3-demo-bucket/hello_world_linux.py"
      }
    ]
```

```
    },
    {
      "Lifecycle": {
        "install": {
          "Skipif": "onpath git",
          "Script": "sudo apt-get install git",
          "RequiresPrivilege": "true"
        }
      },
      "Artifacts": [
        {
          "Uri": "s3://amzn-s3-demo-bucket/hello_world.py"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.FooService
ComponentDescription: Complete recipe for AWS IoT Greengrass components
ComponentPublisher: Amazon
ComponentVersion: 1.0.0
ComponentConfiguration:
  DefaultConfiguration:
    TestParam: TestValue
ComponentDependencies:
  BarService:
    VersionRequirement: ^1.1.0
    DependencyType: SOFT
  BazService:
    VersionRequirement: ^2.0.0
Manifests:
  - Platform:
      os: linux
      architecture: amd64
    Lifecycle:
      install:
        SkipIf: onpath git
        Script: sudo apt-get install git
```

```
      SetEnv:
        environment_variable1: variable_value1
        environment_variable2: variable_value2
    Artifacts:
      - Uri: 's3://amzn-s3-demo-bucket/hello_world.zip'
        Unarchive: ZIP
      - Uri: 's3://amzn-s3-demo-bucket/hello_world_linux.py'
  - Lifecycle:
      install:
        SkipIf: onpath git
        Script: sudo apt-get install git
        RequiresPrivilege: 'true'
    Artifacts:
      - Uri: 's3://amzn-s3-demo-bucket/hello_world.py'
```

Referenz zur Umgebungsvariablen der Komponente

Die AWS IoT Greengrass Core-Software legt Umgebungsvariablen fest, wenn sie Lebenszyklusskripts für Komponenten ausführt. Sie können diese Umgebungsvariablen in Ihren Komponenten abrufen, AWS-Region um den Namen der Sache und die Greengrass Nucleus-Version zu erhalten. Die Software legt auch Umgebungsvariablen fest, die Ihre Komponente benötigt, um das Interprozesskommunikations-SDK zu verwenden und mit AWS Diensten zu interagieren.

Sie können auch benutzerdefinierte Umgebungsvariablen für die Lebenszyklusskripts Ihrer Komponente festlegen. Weitere Informationen finden Sie unter Setenv.

Die AWS IoT Greengrass Core-Software legt die folgenden Umgebungsvariablen fest:

`AWS_IOT_THING_NAME`

Der Name der AWS IoT Sache, die dieses Greengrass-Core-Gerät repräsentiert.

`AWS_REGION`

Der AWS-Region Ort, an dem dieses Greengrass-Core-Gerät arbeitet.

AWS SDKs Sie verwenden diese Umgebungsvariable, um die zu verwendende Standardregion zu identifizieren. Diese Variable ist äquivalent zu`AWS_DEFAULT_REGION`.

`AWS_DEFAULT_REGION`

Der AWS-Region Ort, an dem dieses Greengrass-Core-Gerät arbeitet.

Der AWS 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-Core-Gerät läuft.

`GG_ROOT_CA_PATH`

Diese Funktion ist für Version 2.5.5 und höher der Greengrass Nucleus-Komponente verfügbar.

Der Pfad zum Root-Zertifikat der Zertifizierungsstelle (CA), das der Greengrass-Nucleus verwendet.

`AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT`

Der Pfad zum IPC-Socket, über den Komponenten mit der AWS IoT Greengrass Core-Software 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.

`SVCUID`

Das geheime Token, das Komponenten verwenden, um eine Verbindung zum IPC-Socket herzustellen und mit der AWS 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`

Der URI, den die Komponenten zum Abrufen von Anmeldeinformationen von der Token-Exchange-Dienstkomponente anfordern.

AWS IoT Greengrass Komponenten auf Geräten bereitstellen

Sie können AWS IoT Greengrass es verwenden, um Komponenten auf Geräten oder Gerätegruppen bereitzustellen. Mithilfe von Bereitstellungen definieren Sie die Komponenten und Konfigurationen, die an die Geräte gesendet werden. AWS IoT Greengrass wird für Ziele, AWS IoT Dinge oder Dinggruppen bereitgestellt, die Greengrass-Kerngeräte repräsentieren. AWS IoT Greengrass

verwendet AWS IoT Core Jobs zur Bereitstellung auf Ihren Kerngeräten. Sie können konfigurieren, wie der Job auf Ihren Geräten ausgeführt wird.

Bereitstellungen von Kerngeräten

Auf jedem Kerngerät werden die Komponenten der Bereitstellungen für dieses Gerät ausgeführt. Eine neue Bereitstellung auf demselben Ziel überschreibt die vorherige Bereitstellung auf dem Ziel. Wenn Sie eine Bereitstellung erstellen, definieren Sie die Komponenten und Konfigurationen, die auf die vorhandene Software des Kerngeräts angewendet werden sollen.

Wenn Sie eine Bereitstellung für ein Ziel überarbeiten, ersetzen Sie die Komponenten der vorherigen Version durch die Komponenten in der neuen Version. Beispielsweise stellen Sie die Geheimer Manager Komponenten Protokollmanager und für die Dinggruppe `TestGroup` bereit. Dann erstellen Sie eine weitere Bereitstellung`TestGroup`, für die nur die Secret Manager-Komponente angegeben ist. Das hat zur Folge, dass auf den Kerngeräten in dieser Gruppe der Log Manager nicht mehr ausgeführt wird.

Auflösung der Plattformabhängigkeit

Wenn ein Kerngerät bereitgestellt wird, überprüft es, ob die Komponenten mit dem Kerngerät kompatibel sind. Wenn Sie das beispielsweise Firehose auf einem Windows-Ziel bereitstellen, schlägt die Bereitstellung fehl.

Auflösung der Abhängigkeit von Komponenten

Während einer Komponentenbereitstellung überprüft das Kerngerät die Kompatibilität der Abhängigkeiten und Versionsanforderungen aller Komponenten in einer Dinggruppe. Diese Überprüfung stellt sicher, dass die Versionsbeschränkungen für alle Komponenten und ihre Abhängigkeiten erfüllt sind, bevor mit der Bereitstellung fortgefahren wird.

Der Prozess zur Auflösung von Abhängigkeiten beginnt mit der Identifizierung von Zielkomponenten, deren Rezepturen keine Abhängigkeiten enthalten. Anschließend erstellt das System mithilfe der Breadth-First-Suche (BFS) einen Abhängigkeitsbaum, der jeden Zielknoten systematisch untersucht und zuerst seine Abhängigkeiten findet, bevor zum nächsten Knoten übergegangen wird. Jeder Knoten enthält die Zielkomponente als Schlüssel und die Versionsanforderungen als Wert.

Die Versionsanforderungen kombinieren drei Gruppen von Einschränkungen:

- Die Versionsanforderungen, die bereits in der vorhandenen Dinggruppe festgelegt sind.

- Die für die Bereitstellung erforderliche Komponentenversion. Sie müssen eine Komponentenversion auswählen, wenn Sie eine Bereitstellung erstellen oder aktualisieren.
- Alle Einschränkungen der Komponentenversion, die im Abhängigkeitsabschnitt des Rezepts definiert sind.

Löst Abhängigkeiten zwischen Komponenten auf

Während einer Bereitstellung versucht der Greengrass-Nucleus zunächst, den lokalen Kandidaten zu finden, der derzeit auf dem Gerät läuft, das die Anforderungen erfüllt. Wenn die laufende Komponente die Anforderungen erfüllt, ruft der Nucleus den gespeicherten Rezeptpfad aus dem Rezeptordner ab und sucht im lokalen Speicher nach der neuesten lokalen Version.

Bei AWS Cloud Bereitstellungen ruft der Nucleus dann die ResolveComponentCandidates API auf. Diese API beginnt mit der neuesten verfügbaren Version und prüft, ob sie die Abhängigkeiten und Anforderungen erfüllt. Wenn der Nucleus die Antwort von der API erhält, wählt er die neueste Version aus. Wenn von der keine Version gefunden wird AWS Cloud , die die Anforderungen erfüllt, schlägt die Bereitstellung fehl. Wenn das Gerät offline ist, wird auf den ursprünglich gefundenen lokalen Kandidaten zurückgegriffen. Wenn kein lokaler Kandidat gefunden wird, der die Anforderungen erfüllt, schlägt die Bereitstellung fehl.

Bei lokalen Einsätzen setzt der Nukleus ausschließlich lokale Kandidaten ein, sofern diese vorhanden sind und die Anforderungen erfüllen, ohne mit ihnen zu verhandeln. AWS Cloud Wenn es keinen solchen Kandidaten gibt, schlägt der Einsatz fehl.

> **Note**
>
> Alle aufgelösten Rezepte werden lokal gespeichert, um später darauf future zu können.

Weitere Informationen finden Sie im Abschnitt zur Auflösung von Abhängigkeiten unter GitHub.

Wenn der Greengrass-Kern in der Lage ist, alle Komponenten erfolgreich aufzulösen, enthält das Nukleus-Logbuch die folgende Zeile.

```
resolve-all-group-dependencies-finish. Finish resolving all groups dependencies.
```

Example

Im Folgenden finden Sie ein Beispiel dafür, wie der Nucleus die Anforderungen an die Komponenten lösen kann.

- Sie stellen ComponentA bereit, das von den ComponentC-Versionen 1.0.0-1.9.0 abhängt.
- Sie stellen auch Komponente B bereit, die von den ComponentC-Versionen 1.4.0-1.9.5 abhängt.

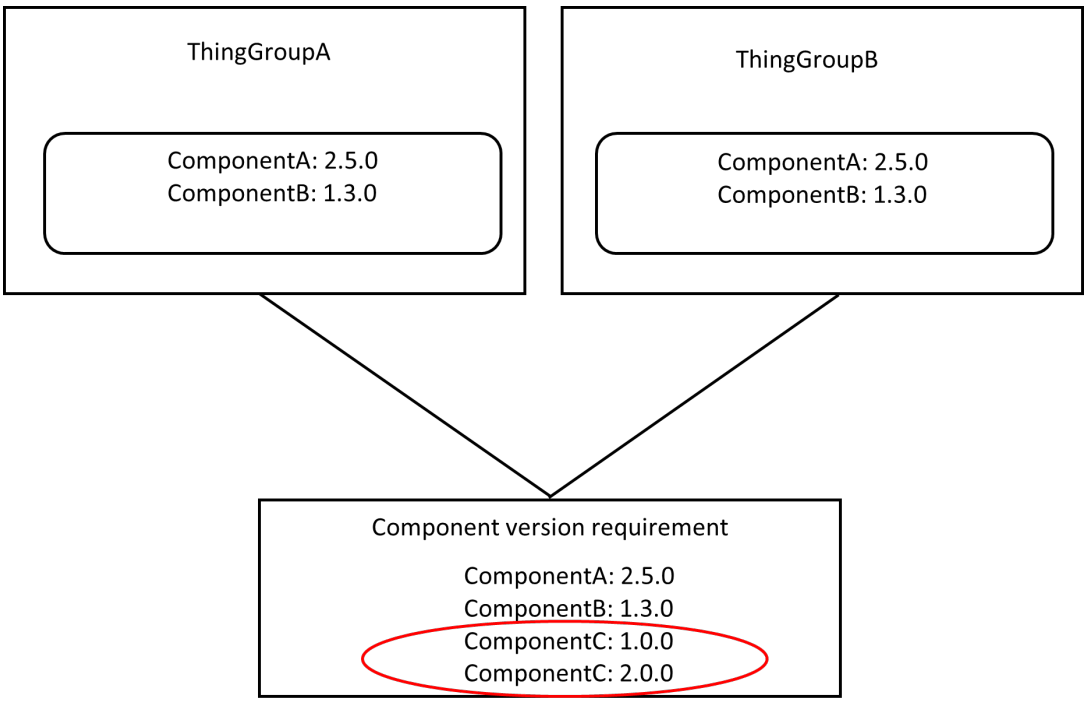
Bei der Auflösung der Komponentenabhängigkeiten stellt der Nucleus die neueste Version der ComponentC-Version bereit, um die Anforderungen von ComponentA und ComponentB zu erfüllen. Diese neueste Version von ComponentC ist Version 1.9.0.

Häufige Fehler bei der Auflösung von Komponentenabhängigkeiten

Die Auflösung von Komponentenabhängigkeiten kann aus zwei Hauptgründen fehlschlagen: Konflikt mit Zielversionsanforderungen oder Konflikt mit Komponentenabhängigkeitsanforderungen.

Szenario 1: Konflikt mit den Anforderungen der Zielversion

- Ein Ding ist in einer Dinggruppe vorhanden und Sie möchten dieses Ding auch zu einer neuen Dinggruppe hinzufügen. Die Bereitstellung schlägt fehl, wenn für die neue Dinggruppe eine andere Dingversion erforderlich ist.
- Eine Bereitstellung kann auch fehlschlagen, wenn ein Ding zu einer Dinggruppe gehört und die Komponentenversion durch eine Dingbereitstellung aktualisiert werden soll.



Beispiel für ein Fehlerprotokoll:

```
2025-04-11T06:16:03.315Z [ERROR] (pool-3-thread-27)
 com.aws.greengrass.componentmanager.ComponentManager: Failed to negotiate version
 with cloud and no local version to fall back to. {componentName=ComponentC,
 versionRequirement={thing/ABC==2.0.0, thinggroup/ThingGroupA==1.0.0}}
2025-04-11T06:16:03.316Z [ERROR] (pool-3-thread-26)
 com.aws.greengrass.deployment.DeploymentService: Error occurred while
 processing deployment. {deploymentId=fbac24de-4ef9-44b0-a685-fdc63b0f02b8,
 serviceName=DeploymentService, currentState=RUNNING}
java.util.concurrent.ExecutionException:
 com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException:
 No local or cloud component version satisfies the requirements Check whether the
 version constraints conflict and that the component exists in your AWS account with
 a version that matches the version constraints. If the version constraints conflict,
 revise deployments to resolve the conflict. Component ComponentC version constraints:
 thing/ABC requires =2.0.0, thinggroup/ThingGroupA requires =1.0.0.
    at java.base/java.util.concurrent.FutureTask.report(FutureTask.java:122)
    at java.base/java.util.concurrent.FutureTask.get(FutureTask.java:191)
    at
 com.aws.greengrass.deployment.DefaultDeploymentTask.call(DefaultDeploymentTask.java:127)
    at
 com.aws.greengrass.deployment.DefaultDeploymentTask.call(DefaultDeploymentTask.java:50)
    at java.base/java.util.concurrent.FutureTask.run(FutureTask.java:264)
```

```
    at java.base/
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1128)
    at java.base/java.util.concurrent.ThreadPoolExecutor
$Worker.run(ThreadPoolExecutor.java:628)
    at java.base/java.lang.Thread.run(Thread.java:829)
Caused by:
 com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException:
 No local or cloud component version satisfies the requirements Check whether the
 version constraints conflict and that the component exists in your AWS account with
 a version that matches the version constraints. If the version constraints conflict,
 revise deployments to resolve the conflict. Component ComponentC version constraints:
 thing/ABC requires =2.0.0, thinggroup/ThingGroupA requires =1.0.0.
    at
 com.aws.greengrass.componentmanager.ComponentManager.negotiateVersionWithCloud(ComponentManage
    at
 com.aws.greengrass.componentmanager.ComponentManager.resolveComponentVersion(ComponentManager.
    at com.aws.greengrass.componentmanager.DependencyResolver.lambda
$resolveDependencies$0(DependencyResolver.java:134)
    at
 com.aws.greengrass.componentmanager.DependencyResolver.resolveComponentDependencies(Dependency
    at
 com.aws.greengrass.componentmanager.DependencyResolver.resolveDependencies(DependencyResolver.
    at com.aws.greengrass.deployment.DefaultDeploymentTask.lambda$call
$2(DefaultDeploymentTask.java:125)
    ... 4 more
```

Die Protokolle deuten auf einen Versionskonfliktfehler hin, da der Nucleus keine Komponentenversion finden kann, die gleichzeitig zwei widersprüchliche Anforderungen erfüllt.

Wie kann das Problem gelöst werden

- Wenn Sie die Komponente in jeder Dinggruppe behalten möchten, wählen Sie in jeder Dinggruppe dieselbe Version dieser Komponente aus.
- Wählen Sie eine Komponentenversion aus, die den Bereitstellungsanforderungen entspricht.
- Wenn Sie eine Komponentenversion verwenden möchten, die nicht beide Dinggruppenanforderungen erfüllt, wählen Sie die Komponentenversion aus, die die Anforderungen an die Dinggruppenversion erfüllt, und verwenden Sie diese Komponente nur in dieser Dinggruppe.

Szenario 2: Konflikt zwischen den Versionsanforderungen der Komponentenabhängigkeit

Wenn eine Komponente von verschiedenen Komponenten abhängig ist und die Komponenten unterschiedliche Versionen oder unterschiedliche Versionsbereiche dieser Komponente erfordern, besteht die Möglichkeit, dass keine Versionen verfügbar sind, die alle Versionsanforderungen erfüllen. In diesem Szenario schlägt die Bereitstellung fehl.

Example

Bereitstellung von ComponentA (v2.5.0), ComponentB (v1.3.0) und ComponentC (v1.0.0)

- ComponentA erfordert ComponentB-Version >=1.0.0.

```
---
...
ComponentName: ComponentA
ComponentVersion: "2.5.0"
ComponentDependencies:
    ComponentB:
        VersionRequirement: ">=1.0.0"
        DependencyType: "HARD"
...
```

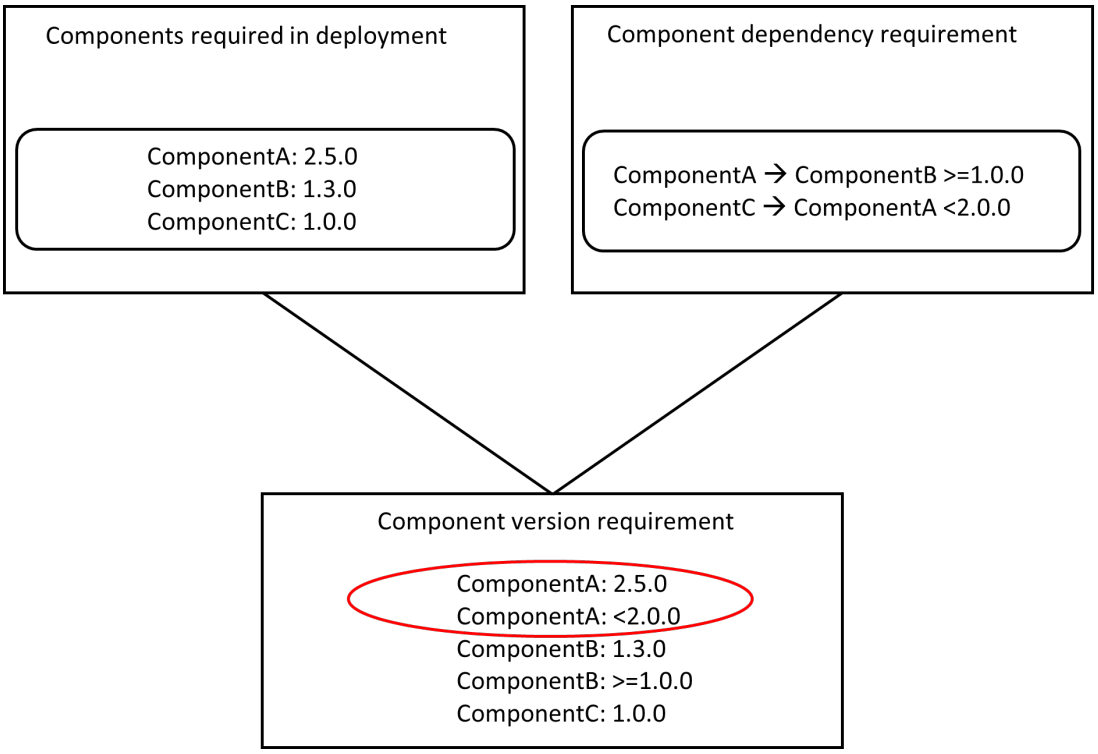
- ComponentC benötigt ComponentA-Version <2.0.0.

```
---
...
ComponentName: ComponentC
ComponentVersion: "1.0.0"
ComponentDependencies:
    ComponentA:
        VersionRequirement: "<2.0.0"
        DependencyType: "HARD"
...
```

Es besteht ein Versionskonflikt zwischen zwei Anforderungen für ComponentA:

- ComponentA benötigt Version 2.5.0 in dieser Bereitstellung
- ComponentC benötigt ComponentA-Versionen unter 2.0.0

Diese beiden Anforderungen widersprechen sich gegenseitig und machen es dem Nucleus unmöglich, eine ComponentA-Version zu finden, die beide Anforderungen erfüllt. Daher schlägt die Abhängigkeitsauflösung fehl.



Beispiel für ein Fehlerprotokoll:

```
2025-04-11T06:07:18.291Z [ERROR] (pool-3-thread-25)
 com.aws.greengrass.componentmanager.ComponentManager: Failed to negotiate version
 with cloud and no local version to fall back to. {componentName=ComponentA,
 versionRequirement={ComponentC=<2.0.0, thinggroup/ThingGroupA==2.5.0}}
2025-04-11T06:07:18.292Z [ERROR] (pool-3-thread-24)
 com.aws.greengrass.deployment.DeploymentService: Error occurred while
 processing deployment. {deploymentId=2ffac4df-1ac9-405c-8c11-28494a1b4382,
 serviceName=DeploymentService, currentState=RUNNING}
java.util.concurrent.ExecutionException:
 com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException:
 No local or cloud component version satisfies the requirements Check whether the
 version constraints conflict and that the component exists in your AWS account with
 a version that matches the version constraints. If the version constraints conflict,
 revise deployments to resolve the conflict. Component ComponentA version constraints:
 ComponentC requires <2.0.0, thinggroup/ThingGroupA requires =2.5.0.
    at java.base/java.util.concurrent.FutureTask.report(FutureTask.java:122)
    at java.base/java.util.concurrent.FutureTask.get(FutureTask.java:191)
```

```
    at
 com.aws.greengrass.deployment.DefaultDeploymentTask.call(DefaultDeploymentTask.java:127)
    at
 com.aws.greengrass.deployment.DefaultDeploymentTask.call(DefaultDeploymentTask.java:50)
    at java.base/java.util.concurrent.FutureTask.run(FutureTask.java:264)
    at java.base/
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1128)
    at java.base/java.util.concurrent.ThreadPoolExecutor
$Worker.run(ThreadPoolExecutor.java:628)
    at java.base/java.lang.Thread.run(Thread.java:829)
Caused by:
 com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException:
 No local or cloud component version satisfies the requirements Check whether the
 version constraints conflict and that the component exists in your AWS account with
 a version that matches the version constraints. If the version constraints conflict,
 revise deployments to resolve the conflict. Component ComponentA version constraints:
 ComponentC requires <2.0.0, thinggroup/ThingGroupA requires =2.5.0.
    at
 com.aws.greengrass.componentmanager.ComponentManager.negotiateVersionWithCloud(ComponentManage
    at
 com.aws.greengrass.componentmanager.ComponentManager.resolveComponentVersion(ComponentManager.
    at com.aws.greengrass.componentmanager.DependencyResolver.lambda
$resolveDependencies$0(DependencyResolver.java:134)
    at
 com.aws.greengrass.componentmanager.DependencyResolver.resolveComponentDependencies(Dependency
    at
 com.aws.greengrass.componentmanager.DependencyResolver.resolveDependencies(DependencyResolver.
    at com.aws.greengrass.deployment.DefaultDeploymentTask.lambda$call
$2(DefaultDeploymentTask.java:125)
    ... 4 more
```

Aus den Protokollen geht hervor, dass der Nucleus keine Version von ComponentA finden kann, die die folgenden Anforderungen erfüllt.

- Die Anforderungen für ComponentA müssen exakt Version 2.5.0 (von A) sein. ThingGroup
- Die Anforderung, mit ComponentC-Versionen unter 2.0.0 zu arbeiten.

Wie löst man das

- Wenn Sie die Komponente in jeder Dinggruppe behalten möchten, wählen Sie in jeder Dinggruppe dieselbe Version dieser Komponente aus.

- Wählen Sie eine Komponentenversion aus, die den Bereitstellungsanforderungen entspricht.
- Wenn Sie eine Komponentenversion verwenden möchten, die nicht beide Dinggruppenanforderungen erfüllt, wählen Sie die Komponentenversion aus, die die Anforderungen an die Dinggruppenversion erfüllt, und verwenden Sie diese Komponente nur in dieser Dinggruppe.

> **Tip**
>
> Wenn Sie diesen Fehler bei einer der AWS bereitgestellten Komponenten sehen, können Sie ihn beheben, indem Sie die in Konflikt stehenden Komponenten auf die neueste Version aktualisieren.

Ein Gerät aus einer Dinggruppe entfernen

Wenn Sie ein Kerngerät aus einer Dinggruppe entfernen, hängt das Verhalten der Komponentenbereitstellung von der Version des Greengrass-Nukleus ab, die auf dem Kerngerät ausgeführt wird.

2.5.1 and later

Wenn Sie ein Kerngerät aus einer Dinggruppe entfernen, hängt das Verhalten davon ab, ob die AWS IoT Richtlinie die `greengrass:ListThingGroupsForCoreDevice` Erlaubnis erteilt. Weitere Informationen zu dieser Berechtigung und zu AWS IoT Richtlinien für Kerngeräte finden Sie unterGeräteauthentifizierung und Autorisierung für AWS IoT Greengrass.

- Wenn die AWS IoT Richtlinie diese Berechtigung gewährt

  Wenn Sie ein Kerngerät aus einer Dinggruppe entfernen, werden bei der nächsten Bereitstellung auf dem Gerät die Komponenten der Dinggruppe AWS IoT Greengrass entfernt. 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 gewährt

  Wenn Sie ein Kerngerät aus einer Dinggruppe entfernen, werden die Komponenten dieser Dinggruppe AWS IoT Greengrass nicht vom Gerät gelöscht.

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 Dinggruppe mit dem `--groupId` Argument an.

2.5.0

Wenn Sie ein Kerngerät aus einer Dinggruppe entfernen, werden bei der nächsten Bereitstellung auf dem Gerät die Komponenten der Dinggruppe AWS IoT Greengrass entfernt. Wenn eine Komponente auf dem Gerät in der nächsten Bereitstellung enthalten ist, wird diese Komponente nicht vom Gerät entfernt.

Dieses Verhalten setzt voraus, dass die AWS IoT Richtlinie des Kerngeräts die `greengrass:ListThingGroupsForCoreDevice` Genehmigung 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 Kerngerät aus einer Dinggruppe entfernen, werden die Komponenten dieser Dinggruppe AWS IoT Greengrass nicht vom Gerät gelöscht.

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 Dinggruppe mit dem `--groupId` Argument an.

Bereitstellungen

Die Bereitstellungen erfolgen kontinuierlich. Wenn Sie eine Bereitstellung erstellen, wird AWS IoT Greengrass die Bereitstellung auf Zielgeräten bereitgestellt, die online sind. Wenn ein Zielgerät nicht online ist, empfängt es die Bereitstellung, wenn es das nächste Mal eine Verbindung herstellt AWS IoT Greengrass. Wenn Sie einer Ziel-Dinggruppe ein Kerngerät hinzufügen, AWS IoT Greengrass sendet das Gerät die neueste Bereitstellung für diese Dinggruppe.

Bevor ein Kerngerät eine Komponente bereitstellt, benachrichtigt es standardmäßig jede Komponente auf dem Gerät. Greengrass-Komponenten können auf die Benachrichtigung reagieren, um die Bereitstellung zu verzögern. Möglicherweise möchten Sie die Bereitstellung verschieben, wenn der Akkuladestand des Geräts niedrig ist oder ein Prozess läuft, der nicht unterbrochen werden kann.

Weitere Informationen finden Sie unter Tutorial: Entwickeln Sie eine Greengrass-Komponente, die Komponenten-Updates verzögert. Wenn Sie eine Einrichtung erstellen, können Sie sie so konfigurieren, dass sie ohne Benachrichtigung der Komponenten bereitgestellt wird.

Jede Zielsache oder Dinggruppe kann jeweils nur eine Bereitstellung haben. Das heißt, wenn Sie eine Bereitstellung für ein Ziel erstellen, wird die vorherige Version der Bereitstellung dieses Ziels nicht AWS IoT Greengrass mehr bereitgestellt.

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 Greengrass Komponenten sind Softwaremodule, die Sie auf Greengrass-Kerngeräten bereitstellen und ausführen. Geräte erhalten Komponenten nur, wenn die Komponente die Plattform des Geräts unterstützt. Auf diese Weise können Sie die Bereitstellung für Gerätegruppen durchführen, 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 AWS bereitgestellte Komponenten auf Ihren Geräten bereitstellen. Wenn Sie eine Komponente bereitstellen, AWS IoT Greengrass identifiziert es alle Abhängigkeiten der Komponenten und stellt sie ebenfalls bereit. Weitere Informationen erhalten Sie unter AWS IoT Greengrass Komponenten entwickeln und AWS-mitgelieferte Komponenten.

  Sie definieren die Version und das Konfigurationsupdate, das für jede Komponente bereitgestellt werden soll. Das Konfigurationsupdate gibt an, wie die bestehende Konfiguration der Komponente auf dem Kerngerät oder die Standardkonfiguration der Komponente geändert werden soll, falls die Komponente auf dem Kerngerät nicht vorhanden ist. Sie können angeben, welche Konfigurationswerte auf die Standardwerte zurückgesetzt werden sollen und welche neuen Konfigurationswerte auf dem Kerngerät zusammengeführt werden sollen. Wenn ein Kerngerät Bereitstellungen für verschiedene Ziele empfängt und in jeder Bereitstellung kompatible Komponentenversionen angegeben sind, wendet das Kerngerät die Konfigurationsupdates in der

Reihenfolge an, die auf dem Zeitstempel basiert, zu dem Sie die Bereitstellung erstellt haben. Weitere Informationen finden Sie unter Komponentenkonfigurationen aktualisieren.

> **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 Sie die AWS IoT Greengrass Core-Software (OTA).

- Richtlinien für die Bereitstellung

  Definieren Sie, wann die Bereitstellung einer Konfiguration sicher ist und was zu tun ist, wenn die Bereitstellung fehlschlägt. Sie können angeben, ob Sie warten möchten, bis die Komponenten melden, dass sie aktualisiert werden können. Sie können auch angeben, ob Geräte auf ihre vorherige Konfiguration zurückgesetzt werden sollen oder nicht, wenn sie eine fehlgeschlagene Bereitstellung anwenden.

- Konfiguration beenden

  Definieren Sie, wann und wie eine Bereitstellung beendet 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 beendet wird, wenn ein Prozentsatz der Geräte diese Bereitstellung nicht anwendet, nachdem eine Mindestanzahl von Geräten sie erhalten hat.

- Rollout-Konfiguration

  Definieren Sie die Geschwindigkeit, mit der eine Bereitstellung auf den Zielgeräten eingeführt wird. Sie können eine exponentielle Ratenerhöhung mit Mindest- und Höchstratengrenzen 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 herunter. AWS Cloud Core-Geräterollen ermöglichen standardmäßig keinen Zugriff auf S3-Buckets. Um benutzerdefinierte Komponenten bereitzustellen, die Artefakte in einem S3-Bucket definieren, muss die Core-Geräterolle Berechtigungen zum Herunterladen von Artefakten aus diesem Bucket gewähren. Weitere Informationen finden Sie unter Erlauben Sie den Zugriff auf S3-Buckets für Komponentenartefakte.

Themen

- Erstellen von Bereitstellungen
- Unterbereitstellungen erstellen
- Bereitstellungen überarbeiten
- Bereitstellungen stornieren
- Überprüfen Sie den Bereitstellungsstatus

Erstellen von Bereitstellungen

Sie können eine Bereitstellung erstellen, die auf ein Ding oder eine Dinggruppe abzielt.

Wenn Sie eine Bereitstellung erstellen, konfigurieren Sie die Softwarekomponenten, die bereitgestellt werden sollen, und die Art und Weise, wie der Bereitstellungsauftrag auf den Zielgeräten ausgeführt wird. Sie können die Bereitstellung in der JSON-Datei definieren, die Sie dem zur Verfügung stellen AWS CLI.

Das Bereitstellungsziel bestimmt die Geräte, auf denen Sie Ihre Komponenten ausführen möchten. Um die Bereitstellung auf einem Kerngerät durchzuführen, geben Sie eine Sache an. Um die Bereitstellung auf mehreren Kerngeräten durchzuführen, geben Sie eine Dinggruppe an, die diese

Geräte umfasst. Weitere Informationen zur Konfiguration von Dinggruppen finden Sie unter Statische Dinggruppen und Dynamische Dinggruppen im AWS IoT Entwicklerhandbuch.

Folgen Sie den Schritten in diesem Abschnitt, um eine Bereitstellung für ein Ziel zu erstellen. Weitere Informationen zum Aktualisieren der Softwarekomponenten auf einem Ziel, das über eine Bereitstellung verfügt, finden Sie unterBereitstellungen überarbeiten.

> **Warning**
>
> Bei CreateDeploymentdiesem Vorgang können Komponenten von Kerngeräten deinstalliert werden. Wenn eine Komponente in der vorherigen Bereitstellung und nicht in der neuen Bereitstellung vorhanden ist, deinstalliert das Kerngerät diese Komponente. Um die Deinstallation von Komponenten zu vermeiden, überprüfen Sie anhand des ListDeploymentsVorgangs zunächst, ob das Ziel für die Bereitstellung bereits über eine vorhandene Bereitstellung verfügt. Verwenden Sie dann den GetDeploymentVorgang, um mit dieser vorhandenen Bereitstellung zu beginnen, wenn Sie eine neue Bereitstellung erstellen.

Um eine Bereitstellung zu erstellen (AWS CLI)

1. Erstellen Sie eine Datei mit dem Namen `deployment.json` und kopieren Sie dann das folgende JSON-Objekt in die Datei. *`targetArn`*Ersetzen Sie es durch den ARN des Dings AWS IoT oder der Dinggruppe, auf die die Bereitstellung ausgerichtet werden soll. Ding und Dinggruppe 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. Prüfen Sie, ob das Bereitstellungsziel über eine bestehende 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. Ersetze es *`targetArn`* durch den ARN der AWS IoT Zielsache oder der Ziel-Dinggruppe.

```
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 nicht über eine bestehende Bereitstellung, und Sie können zu wechselnStep 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 Version 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. *deploymentId*Ersetzen Sie es 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 Jobkonfiguration der Bereitstellung.
- `tags`— Die Tags der Bereitstellung.

3. (Optional) Definieren Sie einen Namen für die Bereitstellung. *deploymentName*Ersetzen Sie ihn 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 für jede Komponente, die Sie hinzufügen, die folgenden Details an:

   - `version`— Die bereitzustellende Komponentenversion.
   - `configurationUpdate`— Das bereitzustellende Konfigurationsupdate. Das Update ist ein Patch-Vorgang, der die bestehende Konfiguration der Komponente auf jedem Zielgerät oder die Standardkonfiguration der Komponente ändert, falls sie auf dem Zielgerät nicht vorhanden ist. Sie können die folgenden Konfigurationsupdates angeben:
     - Updates zurücksetzen (`reset`) — (Optional) Eine Liste von JSON-Zeigern, die die Konfigurationswerte definieren, die auf dem Zielgerät auf ihre Standardwerte zurückgesetzt werden sollen. Die AWS IoT Greengrass Core-Software wendet Reset-Updates an, bevor sie Merge-Updates anwendet. Weitere Informationen finden Sie unter Updates zurücksetzen.
     - Updates 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 Nucleus-Komponente konfigurieren.

     Sie können eine der folgenden Optionen angeben:

     - `posixUser`— Der POSIX-Systembenutzer und optional die Gruppe, die verwendet werden sollen, um diese Komponente auf Linux-Kerngeräten auszuführen. 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 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.

  Diese Funktion ist für Version 2.5.0 und höher der Greengrass Nucleus-Komponente verfügbar.

- `systemResourceLimits`— Die Systemressourcenlimits, die für die Prozesse dieser Komponente gelten 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 Menge an CPU-Zeit, die die Prozesse dieser Komponente 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 dieser 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 dieser 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 Komponente nicht.
  - `memory`— Die maximale Menge an RAM (in Kilobyte), die die Prozesse dieser Komponente auf dem Kerngerät verwenden können.

  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.

Example Beispiel für ein Update der Grundkonfiguration

Das folgende `components` Beispielobjekt gibt an, dass eine Komponente bereitgestellt werden soll`com.example.PythonRuntime`, die einen Konfigurationsparameter mit dem Namen erwartet`pythonVersion`.

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
    "com.example.PythonRuntime": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"pythonVersion\":\"3.7\"}"
      }
    }
  }
}
```

Example Beispiel für ein Konfigurationsupdate mit Updates zum Zurücksetzen und Zusammenführen

Stellen Sie sich ein Beispiel für eine industrielle Dashboard-Komponente vor`com.example.IndustrialDashboard`, die die folgende Standardkonfiguration hat.

```
{
  "name": null,
  "mode": "REQUEST",
  "network": {
    "useHttps": true,
    "port": {
      "http": 80,
      "https": 443
    },
  },
  "tags": []
}
```

Das folgende Konfigurationsupdate spezifiziert die folgenden Anweisungen:

1. Setzen Sie die HTTPS-Einstellung auf ihren Standardwert zurück (`true`).
2. Setzt die Liste der Industrietags auf eine leere Liste zurück.
3. Führen Sie eine Liste von Industriekennzeichnungen zusammen, die Temperatur- und Druckdatenströme für zwei Kessel 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, dass diese industrielle Dashboard-Komponente und das Konfigurationsupdate bereitgestellt werden sollen.

```
{
  "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 Kerngeräte eine Bereitstellung sicher anwenden können oder was zu tun ist, wenn ein Kerngerät die Bereitstellung nicht anwendet. Fügen Sie dazu ein

`deploymentPolicies` Objekt hinzu `deployment.json` und führen Sie dann einen der folgenden Schritte aus:

1. (Optional) Geben Sie die Richtlinie für das Komponenten-Update an (`componentUpdatePolicy`). Diese Richtlinie definiert, ob die Bereitstellung es Komponenten ermöglicht, ein Update aufzuschieben, bis sie zur Aktualisierung bereit sind. Beispielsweise müssen Komponenten möglicherweise Ressourcen bereinigen oder wichtige Aktionen abschließen, bevor sie neu gestartet werden können, um ein Update zu installieren. Diese Richtlinie legt auch fest, wie lange Komponenten Zeit haben, um auf eine Aktualisierungsbenachrichtigung zu reagieren.

   Diese Richtlinie ist ein Objekt mit den folgenden Parametern:

   - `action`— (Optional) Ob Komponenten benachrichtigt und darauf gewartet werden sollen, dass sie melden, wenn sie zur 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 den SubscribeToComponentUpdates IPC-Vorgang verwenden, um diese Benachrichtigungen zu empfangen.
     - `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 auf eine Aktualisierungsbenachrichtigung mit dem DeferComponentUpdate IPC-Vorgang zu reagieren. Wenn die Komponente innerhalb dieser Zeit nicht reagiert, wird die Bereitstellung auf dem Kerngerät fortgesetzt.

     Die Standardeinstellung ist 60 Sekunden.

2. (Optional) Geben Sie die Richtlinie zur Konfigurationsvalidierung an (`configurationValidationPolicy`). Diese Richtlinie definiert, wie lange jede Komponente Zeit hat, um ein Konfigurationsupdate aus einer Bereitstellung zu validieren. Komponenten können den SubscribeToValidateConfigurationUpdates IPC-Vorgang verwenden, um Benachrichtigungen für ihre eigenen Konfigurationsupdates zu abonnieren. Anschließend können die Komponenten den SendConfigurationValidityReport IPC-Vorgang 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 jeder Komponente zur Verfügung steht, um ein Konfigurationsupdate zu validieren. Wenn die Komponente innerhalb dieser Zeit nicht reagiert, wird die Bereitstellung auf dem Kerngerät fortgesetzt.

  Die Standardeinstellung 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, falls die Bereitstellung fehlschlägt. Wählen Sie aus den folgenden Optionen aus:

   - `ROLLBACK`— Wenn die Bereitstellung auf einem Kerngerät fehlschlägt, setzt die AWS IoT Greengrass Core-Software dieses Kerngerät auf die vorherige Konfiguration zurück.
   - `DO_NOTHING`— Wenn die Bereitstellung auf einem Kerngerät fehlschlägt, behält die AWS IoT Greengrass Core-Software die neue Konfiguration bei. Dies kann zu defekten 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 beendet, eingeführt oder wie das Timeout abläuft. AWS IoT Greengrass verwendet AWS IoT Core Jobs, um Bereitstellungen an Kerngeräte zu senden, sodass diese Optionen mit den Konfigurationsoptionen für AWS IoT Core Jobs identisch sind. Weitere Informationen finden Sie unter Job-Rollout und Konfiguration abbrechen im AWS IoT Developer Guide.

   Um die Joboptionen zu definieren, fügen Sie ein `iotJobConfiguration` Objekt 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 Kennzeichnen Sie Ihre AWS IoT Greengrass Version 2 Ressourcen.

8. Führen Sie den folgenden Befehl aus, um die Bereitstellung von 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 Überprüfen Sie den 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. Konfigurationsupdates sind Patch-Operationen, was 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 Sie mit der Komponentenkonfiguration.

Themen

- Updates zurücksetzen
- Updates zusammenführen
- Beispiele

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 das AWS CLI oder verwenden AWS SDKs, 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 interpolateComponentConfigurationKonfigurationsoption von Greengrass Nucleus auf setzen`true`, können Sie in Merge-Updates andere Rezeptvariablen als die *`component_dependency_name`*`:configuration:`*`json_pointer`* Rezeptvariable verwenden. Beispielsweise können Sie die `{iot:thingName}` Rezeptvariable 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 Kerngerät.

```
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 Kerngerät.

```
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",
  ]
}
```

Unterbereitstellungen erstellen

> **Note**
>
> Die Subdeployment-Funktion ist in Greengrass Nucleus Version 2.9.0 und höher verfügbar. Es ist nicht möglich, eine Konfiguration für eine Unterbereitstellung mit früheren Komponentenversionen von Greengrass Nucleus 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 fehlgeschlagene übergeordnete Bereitstellung erneut zu versuchen, wenn ein oder mehrere Geräte in dieser übergeordneten Bereitstellung ausfallen. Mit dieser Funktion können Sie Geräte auswählen, die in dieser übergeordneten Bereitstellung

ausgefallen 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.

Folgen Sie den Schritten in diesem Abschnitt, um eine Unterbereitstellung zu erstellen und ihren Status zu überprüfen. Weitere Informationen zum Erstellen von Bereitstellungen finden Sie unter Bereitstellungen erstellen.

So erstellen Sie eine Unterbereitstellung ()AWS CLI

1. Führen Sie den folgenden Befehl aus, um die neuesten Bereitstellungen für eine Dinggruppe abzurufen. Ersetzen Sie den ARN im Befehl durch den ARN der abzufragenden Dinggruppe. Stellen Sie `--history-filter` auf **`LATEST_ONLY`** ein, um die neueste Bereitstellung dieser Dinggruppe zu sehen.

   ```
   aws greengrassv2 list-deployments --target-arn arn:aws:iot:region:account-id:thinggroup/thingGroupName --history-filter LATEST_ONLY
   ```

2. Kopieren Sie den `deploymentId` aus der Antwort in den list-deployments Befehl, um ihn im nächsten Schritt zu verwenden.
3. Führen Sie den folgenden Befehl aus, um den Status einer Bereitstellung abzurufen. *`deploymentId`*Ersetzen Sie ihn durch die ID der Bereitstellung, die abgefragt werden soll.

   ```
   aws greengrassv2 get-deployment --deployment-id deploymentId
   ```

4. Kopieren Sie den `iotJobId` aus der Antwort in den get-deployment Befehl, der im folgenden Schritt verwendet werden soll.
5. Führen Sie den folgenden Befehl aus, um die Liste der Auftragsausführungen für den angegebenen Job abzurufen. *`jobID`*Ersetzen Sie durch den `iotJobId` aus dem vorherigen Schritt. *`status`*Ersetzen Sie 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 Dinggruppe oder verwenden Sie eine vorhandene Dinggruppe für Ihre Unterbereitstellung. Fügen Sie dann dieser Dinggruppe ein AWS IoT Ding hinzu. Sie verwenden Dinggruppen, um Flotten von Greengrass-Kerngeräten zu verwalten. Wenn Sie Softwarekomponenten auf Ihren Geräten bereitstellen, können Sie entweder einzelne Geräte oder Gerätegruppen als Ziel auswählen. Sie können ein Gerät zu einer Dinggruppe mit einer aktiven Greengrass-Bereitstellung hinzufügen. Nach dem Hinzufügen können Sie dann die Softwarekomponenten dieser Dinggruppe auf diesem Gerät bereitstellen.

   Gehen Sie wie folgt vor, um eine neue Dinggruppe zu erstellen und ihr Ihre Geräte hinzuzufügen:

   a. Erstelle eine AWS IoT Dinggruppe. *MyGreengrassCoreGroup*Ersetzen Sie durch den Namen für die neue Dinggruppe. Sie können keinen Doppelpunkt (:) in einem Dinggruppennamen verwenden.

      > **Note**
      >
      > Wenn eine Dinggruppe für eine Unterbereitstellung zusammen mit einer anderen Gruppe verwendet wird`parentTargetArn`, kann sie nicht mit einer anderen übergeordneten Flotte wiederverwendet werden. Wenn eine Dinggruppe bereits verwendet wurde, um eine Unterbereitstellung für eine andere Flotte zu erstellen, gibt die API einen Fehler zurück.

      ```
      aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
      ```

      Wenn die Anfrage erfolgreich ist, sieht die Antwort dem folgenden Beispiel ähnlich:

      ```
      {
        "thingGroupName": "MyGreengrassCoreGroup",
        "thingGroupArn": "arn:aws:iot:us-
      west-2:123456789012:thinggroup/MyGreengrassCoreGroup",
        "thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"
      ```

```
}
```

b. Fügen Sie Ihrer Dinggruppe einen bereitgestellten Greengrass-Kern hinzu. Führen Sie den folgenden Befehl mit diesen Parametern aus:

- *MyGreengrassCore*Ersetzen Sie durch den Namen Ihres bereitgestellten Greengrass-Kerns.
- *MyGreengrassCoreGroup*Ersetzen Sie es durch den Namen Ihrer 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.

7. Erstellen Sie eine Datei mit dem Namen `deployment.json` und kopieren Sie dann das folgende JSON-Objekt in die Datei. *targetArn*Ersetzen Sie es durch den ARN der Dinggruppe AWS IoT , die als Ziel für die Unterbereitstellung verwendet werden soll. Ein Unterbereitstellungsziel kann nur eine Dinggruppe sein. Dinggruppen ARNs haben das folgende Format:

   - Dinggruppe — `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 die Auftragskonfiguration. *deploymentId*Ersetzen Sie durch die ID vonStep 1. Sie können diese Bereitstellungskonfiguration verwenden, um Ihre Unterbereitstellung zu konfigurieren und bei 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 Hinweise zu den Einzelheiten 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 des Einsatzes.
- `iotJobConfiguration`— Die Jobkonfiguration der Bereitstellung.
- `parentTargetArn`— Das Ziel der übergeordneten Bereitstellung.
- `tags`— Die Tags der Bereitstellung.

9. Führen Sie den folgenden Befehl aus, um die Unterbereitstellung aus `deployment.json` zu erstellen. Ersetzen Sie es *`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 die übergeordnete Bereitstellung zu überarbeiten. Kopieren Sie die`deployment.json`, die Sie im vorherigen Schritt verwendet haben. Ersetzen Sie das `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, ersetzt diese alle Bereitstellungsrevisionen und Unterbereitstellungen für diese übergeordnete Bereitstellung. Weitere Informationen finden Sie unter Bereitstellungen überarbeiten.

```
aws greengrassv2 create-deployment --cli-input-json file://deployment.json
```

   Die Antwort enthält ein`deploymentId`, das diese Bereitstellung identifiziert. Sie können die Bereitstellungs-ID verwenden, um den Status der Bereitstellung zu überprüfen. Weitere Informationen finden Sie unter Überprüfen Sie den Bereitstellungsstatus.

Bereitstellungen überarbeiten

Jedes Zielobjekt oder jede Zielgruppe kann jeweils nur eine aktive Bereitstellung haben. Wenn Sie eine Bereitstellung für ein Ziel erstellen, für das bereits eine Bereitstellung vorhanden ist, ersetzen die Softwarekomponenten in der neuen Bereitstellung die Softwarekomponenten der vorherigen Bereitstellung. Wenn die neue Bereitstellung keine Komponente definiert, die in der vorherigen Bereitstellung definiert wurde, entfernt die AWS IoT Greengrass Core-Software diese Komponente von den Core-Zielgeräten. Sie können eine bestehende Bereitstellung überarbeiten, sodass Sie die Komponenten, die auf Kerngeräten ausgeführt werden, nicht aus einer früheren Bereitstellung auf ein Ziel entfernen.

Um eine Bereitstellung zu überarbeiten, erstellen Sie eine Bereitstellung, die von denselben Komponenten und Konfigurationen ausgeht, die in einer früheren Bereitstellung vorhanden waren. Sie verwenden den CreateDeploymentVorgang, bei dem es sich um denselben Vorgang handelt, den Sie zum Erstellen von Bereitstellungen verwenden.

Um eine Bereitstellung zu überarbeiten ()AWS CLI

1. Führen Sie den folgenden Befehl aus, um die Bereitstellungen für das Bereitstellungsziel aufzulisten. Ersetze es *targetArn* durch den ARN der AWS IoT Zielsache oder der Ziel-Dinggruppe.

   ```
   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 Version 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 die Auftragskonfiguration. *deploymentId*Ersetzen Sie es 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`

   Für den CreateDeploymentVorgang wird eine Nutzlast mit der folgenden Struktur erwartet.

   ```
   {
     "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 Jobkonfiguration der Bereitstellung (`iotJobConfiguration`).
   - Ändern Sie die Tags der Bereitstellung (`tags`).

   Weitere Informationen zur Definition dieser Bereitstellungsdetails finden Sie unterErstellen von Bereitstellungen.

6. Führen Sie den folgenden Befehl aus, um die Bereitstellung von 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 Überprüfen Sie den Bereitstellungsstatus.

Bereitstellungen stornieren

Sie können eine aktive Bereitstellung abbrechen, um zu verhindern, dass die zugehörigen Softwarekomponenten auf AWS 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, ändern Sie die Komponenten auf diesem Gerät nicht, wenn Sie die Bereitstellung stornieren. 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 neuesten Bereitstellungsversion für ein Ziel zu ermitteln. Die neueste Version ist die einzige Bereitstellung, die für ein Ziel aktiv sein kann, da frühere Bereitstellungen abgebrochen werden, wenn Sie eine neue Revision erstellen. Ersetze es *`targetArn`* durch den ARN der AWS IoT Zielsache oder der Ziel-Dinggruppe.

   ```
   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.

2. Führen Sie den folgenden Befehl aus, um die Bereitstellung abzubrechen. *`deploymentId`*Ersetzen Sie es durch die ID aus dem vorherigen Schritt.

   ```
   aws greengrassv2 cancel-deployment --deployment-id deploymentId
   ```

   Wenn der Vorgang erfolgreich ist, ändert sich der Bereitstellungsstatus auf`CANCELED`.

Überprüfen Sie den Bereitstellungsstatus

Sie können den Status einer Bereitstellung überprüfen, die Sie in erstellen AWS IoT Greengrass. Sie können auch den Status der AWS IoT Jobs überprüfen, mit denen die Bereitstellung auf jedem Kerngerät bereitgestellt wird. Solange eine Bereitstellung aktiv ist, lautet der Status des AWS IoT Jobs`IN_PROGRESS`. Nachdem Sie eine neue Version einer Bereitstellung erstellt haben, ändert sich der Status des AWS IoT Jobs der vorherigen Revision in`CANCELLED`.

Themen

- Überprüfen Sie den Bereitstellungsstatus
- Überprüfen Sie den Bereitstellungsstatus des Geräts

Überprüfen Sie den Bereitstellungsstatus

Sie können den Status einer Bereitstellung überprüfen, die Sie anhand ihres Ziels oder ihrer ID identifizieren.

Um den Bereitstellungsstatus nach Ziel (AWS CLI) zu überprüfen

- Führen Sie den folgenden Befehl aus, um den Status der letzten Bereitstellung für ein Ziel abzurufen. *`targetArn`*Ersetzen Sie es durch den Amazon-Ressourcennamen (ARN) der AWS IoT Sache oder der Dinggruppe, auf die die Bereitstellung abzielt.

  ```
  aws greengrassv2 list-deployments --target-arn targetArn
  ```

  Die Antwort enthält eine Liste mit der neuesten Bereitstellung 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. *`deploymentId`*Ersetzen Sie ihn durch die ID der Bereitstellung, die abgefragt werden soll.

  ```
  aws greengrassv2 get-deployment --deployment-id deploymentId
  ```

  Die Antwort enthält den Status der Bereitstellung.

Überprüfen Sie den Bereitstellungsstatus des Geräts

Sie können den Status eines Bereitstellungsauftrags überprüfen, der für ein einzelnes Core-Gerät gilt. Sie können auch den Status eines Bereitstellungsauftrags für eine Dinggruppen-Bereitstellung überprüfen.

Um den Status eines Bereitstellungsauftrags für ein Kerngerät zu überprüfen ()AWS CLI

- Führen Sie den folgenden Befehl aus, um den Status aller Bereitstellungsaufträge für ein Kerngerät 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 den Job für eine Bereitstellung anhand des Auftrags `deploymentId` oder identifizieren`targetArn`. Jeder Bereitstellungsauftrag enthält den Status des Auftrags 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. Ersetze es *targetArn* durch den ARN der Ziel-Dinggruppe.

   ```
   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 Bereitstellung für das Ziel auflisten. Geben Sie das `--history-filter ALL` Argument an, um alle Bereitstellungen 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 Details der Bereitstellung abzurufen. *deploymentID*Ersetzen Sie es durch die ID aus dem vorherigen Schritt.

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

Die Antwort enthält Informationen über die Bereitstellung. Kopieren Sie die `iotJobId` aus der Antwort, um sie im folgenden Schritt zu verwenden.

3. Führen Sie den folgenden Befehl aus, um die Auftragsausführung eines Kerngeräts für die Bereitstellung zu beschreiben. Ersetzen Sie *`iotJobId`* und *`coreDeviceThingName`* durch die Job-ID aus dem vorherigen Schritt und das Kerngerät, dessen Status Sie überprüfen möchten.

```
aws iot describe-job-execution --job-id iotJobId --thing-name coreDeviceThingName
```

Die Antwort enthält den Status der Ausführung des Bereitstellungsauftrags auf dem Kerngerät sowie Einzelheiten zum Status. Die `detailsMap` enthält die folgenden Informationen:

- `detailed-deployment-status`— Der Status des Bereitstellungsergebnisses, der einer der folgenden Werte sein kann:
  - `SUCCESSFUL`— Die Bereitstellung war erfolgreich.
  - `FAILED_NO_STATE_CHANGE`— Die Bereitstellung schlug fehl, während das Kerngerät sich darauf vorbereitete, die Bereitstellung anzuwenden.
  - `FAILED_ROLLBACK_NOT_REQUESTED`— Die Bereitstellung schlug fehl, und bei der Bereitstellung wurde nicht angegeben, zu einer früheren funktionierenden Konfiguration zurückzukehren, sodass das Kerngerät möglicherweise nicht ordnungsgemäß funktioniert.
  - `FAILED_ROLLBACK_COMPLETE`— Die Bereitstellung ist fehlgeschlagen, und das Kerngerät wurde erfolgreich auf eine frühere funktionierende Konfiguration zurückgesetzt.
  - `FAILED_UNABLE_TO_ROLLBACK`— Die Bereitstellung ist fehlgeschlagen, und das Kerngerät konnte nicht zu einer früheren funktionierenden Konfiguration zurückgesetzt werden, sodass das Kerngerät möglicherweise nicht ordnungsgemäß 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 unterAWS IoT Greengrass Protokolle überwachen.
- `deployment-failure-cause`— Eine Fehlermeldung, die zusätzliche Informationen darüber enthält, warum die Auftragsausführung fehlgeschlagen ist.

Die Antwort sieht dem folgenden Beispiel ähnlich.


```
{
  "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
  }
}
```

Einloggen und Überwachen AWS IoT Greengrass

Die Überwachung ist ein wichtiger Bestandteil der Aufrechterhaltung der Zuverlässigkeit, Verfügbarkeit AWS IoT Greengrass und Leistung Ihrer AWS Lösungen. Sie sollten Überwachungsdaten aus allen Teilen Ihrer AWS Lösung sammeln, damit Sie einen etwaigen Ausfall an mehreren Stellen leichter debuggen können. Bevor Sie mit der Überwachung beginnen AWS IoT Greengrass, sollten Sie einen Überwachungsplan erstellen, der Antworten auf die folgenden Fragen enthält:

- Was sind Ihre Überwachungsziele?
- Welche Ressourcen möchten Sie überwachen?
- Wie oft werden diese Ressourcen überwacht?
- Welche Überwachungs-Tools möchten Sie verwenden?
- Wer soll die Überwachungsaufgaben ausführen?
- Wer soll benachrichtigt werden, wenn Fehler auftreten?

Themen

- Überwachungstools
- AWS IoT Greengrass Protokolle überwachen
- AWS IoT Greengrass V2 API-Aufrufe protokollieren mit AWS CloudTrail
- Erfassung von Telemetriedaten zur Systemintegrität von AWS IoT Greengrass Kerngeräten
- Erhalten Sie Benachrichtigungen über den Bereitstellungs- und Komponentenstatus
- Überprüfen Sie den Status des Greengrass Core-Geräts

Überwachungstools

AWS stellt Tools bereit, die Sie zur Überwachung verwenden können AWS IoT Greengrass. Sie können einige dieser Tools für die Überwachung konfigurieren. Einige der Tools erfordern manuelle Eingriffe. Wir empfehlen, dass Sie die Überwachungsaufgaben möglichst automatisieren.

Sie können die folgenden automatisierten Überwachungstools verwenden, um Probleme zu überwachen AWS IoT Greengrass und zu melden:

- Amazon CloudWatch Logs — Überwachen, speichern und greifen Sie auf Ihre Protokolldateien aus AWS CloudTrail oder anderen Quellen zu. Weitere Informationen finden Sie unter Überwachung von Protokolldateien im CloudWatch Amazon-Benutzerhandbuch.
- AWS CloudTrail Protokollüberwachung — Teilen Sie Protokolldateien zwischen Konten, überwachen CloudTrail Sie Protokolldateien in Echtzeit, indem Sie sie an CloudWatch Logs senden, schreiben Sie Protokollverarbeitungsanwendungen in Java und stellen Sie sicher, dass sich Ihre Protokolldateien nach der Lieferung von nicht geändert haben CloudTrail. Weitere Informationen finden Sie unter Arbeiten mit CloudTrail Protokolldateien im AWS CloudTrail Benutzerhandbuch.
- Greengrass-Systemzustandstelemetrie — Abonnieren Sie den Empfang von Telemetriedaten, die vom Greengrass-Core gesendet werden. Weitere Informationen finden Sie unter the section called “Sammeln Sie Telemetriedaten zur Systemintegrität”.
- Benachrichtigungen zum Gerätestatus Erstellen Sie Ereignisse mithilfe von Amazon EventBridge , um Status-Updates zu Bereitstellungen und Komponenten zu erhalten. Weitere Informationen finden Sie unter Erhalten Sie Benachrichtigungen über den Bereitstellungs- und Komponentenstatus.
- Flottenstatus-Service — Verwenden Sie die Flottenstatus-API-Operationen, um den Status der Kerngeräte und ihrer Greengrass-Komponenten zu überprüfen. Sie können die Informationen zum Flottenstatus auch in der AWS IoT Greengrass Konsole einsehen. Weitere Informationen finden Sie unter Überprüfen Sie den Status des Greengrass Core-Geräts.

AWS IoT Greengrass Protokolle überwachen

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 Kerngeräts schreiben. Greengrass-Komponenten, die auf dem Kerngerät ausgeführt werden, können auch Protokolle in CloudWatch Logs und das lokale Dateisystem schreiben. Sie können die Protokolle verwenden, um Ereignisse zu überwachen und Probleme zu beheben. Alle AWS IoT Greengrass Protokolleinträge enthalten einen Zeitstempel, eine Protokollebene und 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, die Sie entwickeln und bereitstellen, debuggen können. Sie können ein Core-Gerät auch so konfigurieren, dass es Protokolle in Logs schreibt, sodass Sie Fehler auf dem Core-Gerät

beheben können, ohne auf das lokale Dateisystem zugreifen zu müssen. CloudWatch Weitere Informationen finden Sie unter Aktivieren Sie die Protokollierung in CloudWatch Logs.

Themen

- Greifen Sie auf Dateisystemprotokolle zu
- Zugriffs-Logs CloudWatch
- Greifen Sie auf Systemdienstprotokolle zu
- Aktivieren Sie die Protokollierung in CloudWatch Logs
- Konfigurieren Sie die Protokollierung für AWS IoT Greengrass
- AWS CloudTrail Logs

Greifen Sie auf Dateisystemprotokolle zu

Die AWS IoT Greengrass Core-Software speichert Protokolle in dem */greengrass/v2*/logs Ordner auf einem Core-Gerät, in dem */greengrass/v2* sich der Pfad zum AWS IoT Greengrass Stammordner befindet. Der Protokollordner 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-Nucleus, den AWS IoT Greengrass Kern der Core-Software, und Plugin-Komponenten wie Log Manager und Secret Manager.
- *ComponentName*`.log`— Protokolldateien der Greengrass-Komponente. Verwenden Sie Komponenten-Protokolldateien, um Echtzeitinformationen über eine Greengrass-Komponente anzuzeigen, die auf dem Kerngerät ausgeführt wird. Generische Komponenten und Lambda-Komponenten schreiben Standardausgaben (stdout) und Standardfehler (stderr) in diese Protokolldateien.
- `main.log`— Die Protokolldatei für den `main` Dienst, der die Lebenszyklen von Komponenten verwaltet. Diese Protokolldatei wird immer leer sein.

Weitere Hinweise zu den Unterschieden zwischen Plug-in-, generischen und Lambda-Komponenten finden Sie unterKomponententypen.

Die folgenden Hinweise gelten für die Verwendung von Dateisystemprotokollen:

- Root-Benutzerberechtigungen

  Sie benötigen Root-Rechte, um AWS IoT Greengrass Protokolle im Dateisystem lesen zu können.

- Rotation der Protokolldateien

  Die AWS IoT Greengrass Core-Software rotiert die Protokolldateien jede Stunde oder wenn sie eine Dateigrößenbeschränkung überschreiten. Rotierte Protokolldateien enthalten einen Zeitstempel in ihrem Dateinamen. Beispielsweise könnte eine rotierte AWS IoT Greengrass Core-Software-Protokolldatei benannt `greengrass_2021_09_14_15_0.log` werden. Die Standarddateigrößenbeschränkung beträgt 1.024 KB (1 MB). Sie können die Dateigrößenbeschränkung für die Greengrass Nucleus-Komponente konfigurieren.

- Löschen der Protokolldatei

  Die AWS IoT Greengrass Core-Software bereinigt frühere Protokolldateien, wenn die Größe der AWS IoT Greengrass Core-Software-Protokolldateien oder der Protokolldateien von Greengrass-Komponenten, einschließlich rotierter Protokolldateien, eine Speicherplatzbeschränkung überschreitet. Das Standardlimit für den Festplattenspeicher 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-Software-Protokoll auf der Greengrass Nucleus-Komponente oder der Log Manager-Komponente konfigurieren. Sie können die Speicherplatzbeschränkung für die einzelnen Komponenten in der Log Manager-Komponente konfigurieren.

Um die AWS IoT Greengrass Core-Software-Protokolldatei anzuzeigen

- Führen Sie den folgenden Befehl aus, um die Protokolldatei in Echtzeit anzuzeigen. */greengrass/v2*Ersetzen Sie es durch 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
```

Um die Protokolldatei für eine Komponente anzuzeigen

- 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 *com.example.HelloWorld* ersetzen Sie ihn durch 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 Kerngerät zu analysieren. Um den `logs` Befehl verwenden zu können, müssen Sie den Greengrass-Nucleus so konfigurieren, dass er Protokolldateien im JSON-Format ausgibt. Weitere Informationen erhalten Sie unter Greengrass-Befehlszeilenschnittstelle und Protokolle.

Zugriffs-Logs CloudWatch

Sie können die Log Manager-Komponente bereitstellen, um das Kerngerät so zu konfigurieren, dass es in CloudWatch Protokolle schreibt. Weitere Informationen finden Sie unter Aktivieren Sie die Protokollierung in CloudWatch Logs. Anschließend können Sie die Protokolle auf der Logs-Seite der CloudWatch Amazon-Konsole oder mithilfe der CloudWatch Logs-API anzeigen.

Name der Protokollgruppe

```
/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 umfasst Protokolle für die Nucleus- und Plugin-Komponenten, die in derselben JVM wie der Greengrass-Nucleus ausgeführt werden. Die Komponente ist Teil des Greengrass-Kerns.
  - `UserComponent`— Diese Protokollgruppe umfasst 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 Kerngerät verwendet.
- `componentName`— Der Name der Komponente. Für Systemprotokolle ist dieser Wert`System`.

Name des Protokollstreams

```
/date/thing/thingName
```

Der Name des Protokolldatenstroms verwendet die folgenden Variablen:

- `date`— Das Datum des Protokolls, z. `2020/12/15` B. Die Log-Manager-Komponente verwendet das `yyyy/MM/dd` Format.
- `thingName`— Der Name des Kerngeräts.

> **Note**
>
> Wenn ein Dingname einen Doppelpunkt (`:`) enthält, ersetzt der Logmanager den Doppelpunkt durch ein Plus (+).

Die folgenden Überlegungen gelten, wenn Sie die Log Manager-Komponente zum Schreiben in CloudWatch Logs verwenden:

- Verzögerungen protokollieren

> **Note**
>
> Wir empfehlen Ihnen, auf Log Manager Version 2.3.0 zu aktualisieren, um Protokollverzögerungen bei rotierten und aktiven Protokolldateien zu reduzieren. Wenn Sie auf Log Manager 2.3.0 aktualisieren, empfehlen wir Ihnen, auch auf Greengrass Nucleus 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 die Protokolldateien jede Stunde oder nachdem sie 1.024 KB groß sind. Daher lädt die Log-Manager-Komponente Protokolle erst hoch, nachdem die AWS IoT Greengrass Core-Software oder eine Greengrass-Komponente Protokolle im Wert von über 1.024 KB geschrieben hat. Sie können eine niedrigere Größenbeschränkung für Protokolldateien konfigurieren, damit Protokolldateien häufiger rotieren. Dies führt dazu, dass die Log-Manager-Komponente Logs häufiger in CloudWatch Logs hochlädt.

Die Log Manager-Komponente Version 2.3.0 (und höher) verarbeitet und lädt alle Protokolle hoch. Wenn Sie ein neues Protokoll schreiben, verarbeitet Log Manager Version 2.3.0 (und höher) diese aktive Protokolldatei und lädt sie 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 Log-Manager-Komponente Logs häufiger in Logs hochlädt, CloudWatch indem Sie das konfigurieren. `periodicUploadIntervalSec` Weitere Informationen zur Konfiguration dieses periodischen 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 zum Schreiben von Protokollen verwenden, kehrt der Log Manager zum Verhalten der Protokollmanager-Komponentenversionen 2.2.8 und

früher zurück. Informationen zum Zugriff auf Dateisystemprotokolle finden Sie unter Zugriff auf Dateisystemprotokolle.

- Zeitversatz

  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 Kerngerät um mehr als 15 Minuten nicht mehr synchron ist, lehnt CloudWatch Logs die Anfragen ab. Weitere Informationen finden Sie unter Signaturprozess mit Signaturversion 4 in derAllgemeine AWS-Referenz .

Greifen Sie auf Systemdienstprotokolle zu

Wenn Sie die AWS IoT Greengrass Core-Software als Systemdienst konfigurieren, können Sie Systemdienstprotokolle einsehen, um Probleme zu beheben, z. B. wenn die Software nicht gestartet werden kann.

So zeigen Sie Systemdienstprotokolle (CLI) an

1. Führen Sie den folgenden Befehl aus, um die Systemdienstprotokolle der AWS IoT Greengrass Core-Software 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 Fehlerprotokolle des Systemdienstes 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 Systemdienstprotokolle anzuzeigen.

So zeigen Sie Windows-Dienstprotokolle an (Event Viewer)

1. Öffnen Sie die Event Viewer-Anwendung.
2. Wählen Sie Windows-Protokolle aus, um es zu erweitern.
3. Wählen Sie Anwendung, um die Anwendungsdienstprotokolle anzuzeigen.
4. Suchen und öffnen Sie die Ereignisprotokolle, deren Quelle ist greengrass.

Aktivieren Sie die Protokollierung in CloudWatch Logs

Sie können die Log Manager-Komponente einsetzen, um ein Kerngerät so zu konfigurieren, dass es CloudWatch Protokolle in Logs schreibt. Sie können CloudWatch Logs for AWS IoT Greengrass Core Software Logs aktivieren, und Sie können CloudWatch Logs für bestimmte Greengrass-Komponenten aktivieren.

> **Note**
>
> Die Token-Austauschrolle des Greengrass-Core-Geräts muss es dem Kerngerät ermöglichen, in CloudWatch Logs zu schreiben, wie in der folgenden IAM-Beispielrichtlinie gezeigt. Wenn Sie die AWS IoT Greengrass Core-Software mit automatischer Ressourcenbereitstellung installiert haben, verfügt Ihr Kerngerät 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 so zu konfigurieren, dass es AWS IoT Greengrass Core-Softwareprotokolle in CloudWatch Logs schreibt, erstellen Sie eine Bereitstellung, die ein Konfigurationsupdate spezifiziert, das `true` für die `aws.greengrass.LogManager` Komponente `uploadToCloudWatch` auf eingestellt ist. AWS IoT Greengrass Zu den wichtigsten Softwareprotokollen gehören Protokolle für den Greengrass-Nucleus und die Plugin-Komponenten.

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true"
    }
  }
}
```

Um ein Core-Gerät so zu konfigurieren, dass es die Logs einer Greengrass-Komponente in Logs schreibt, erstellen Sie ein Deployment, das ein Konfigurationsupdate spezifiziert, das die Komponente zur Liste der Komponenten-Logging-Konfigurationen hinzufügt. CloudWatch Wenn Sie dieser Liste eine Komponente hinzufügen, schreibt die Log-Manager-Komponente ihre Protokolle in CloudWatch Logs. Zu den Komponentenprotokollen gehören Protokolle für generische Komponenten und Lambda-Komponenten.

```
{
  "logsUploaderConfiguration": {
    "componentLogsConfigurationMap": {
      "com.example.HelloWorld": {

      }
    }
```

```
  }
}
```

Wenn Sie die Log Manager-Komponente bereitstellen, können Sie auch Festplattenspeicherbeschränkungen konfigurieren und festlegen, ob das Kerngerät Protokolldateien löscht, nachdem sie in Logs geschrieben wurden. CloudWatch Weitere Informationen finden Sie unter Konfigurieren Sie die Protokollierung für AWS IoT Greengrass.

Konfigurieren Sie die 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 ein Konfigurationsupdate für die Greengrass Nucleus- oder Log Manager-Komponenten spezifiziert.

- Protokolle in Logs schreiben CloudWatch

  Um Probleme mit Kerngeräten per Fernzugriff zu beheben, können Sie die Kerngeräte so konfigurieren, dass sie die AWS IoT Greengrass Core-Software- und CloudWatch Komponentenprotokolle in Logs schreiben. Stellen Sie dazu die Log Manager-Komponente bereit und konfigurieren Sie sie. Weitere Informationen finden Sie unter Aktivieren Sie die Protokollierung in CloudWatch Logs.
- Löschen hochgeladener Protokolldateien

  Um den Speicherverbrauch zu reduzieren, können Sie die Kerngeräte so konfigurieren, dass Protokolldateien nach dem Schreiben der Protokolldateien in CloudWatch Logs gelöscht werden. Weitere Informationen finden Sie im `deleteLogFileAfterCloudUpload` Parameter der Log Manager-Komponente, den Sie für AWS IoT Greengrass Core-Softwareprotokolle und Komponentenprotokolle angeben können.
- Speicherplatzbeschränkungen protokollieren

  Um die Speicherbelegung zu begrenzen, können Sie den maximalen Festplattenspeicher für jedes Protokoll, einschließlich der rotierten Protokolldateien, auf einem Core-Gerät konfigurieren. Sie können beispielsweise den maximalen kombinierten Speicherplatz für Dateien `greengrass.log` und rotierte `greengrass.log` Dateien konfigurieren. Weitere Informationen finden Sie in den Parametern der Greengrass Nucleus-Komponente und in den `logging.totalLogsSizeKB` Parametern der Log `diskSpaceLimit` Manager-Komponente, die Sie für AWS IoT Greengrass Core-Softwareprotokolle und Komponentenprotokolle angeben können.
- Größenbeschränkung für Protokolldateien

Sie können die maximale Dateigröße für jede Protokolldatei konfigurieren. Wenn 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 Logs häufiger in CloudWatch Logs zu schreiben. Die Protokollmanager-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 im Parameter zur Größenbeschränkung für Protokolldateien (`logging.fileSizeKB`) der Greengrass Nucleus-Komponente.

- Minimale Protokollebenen

  Sie können die minimale Protokollebene konfigurieren, die die Greengrass Nucleus-Komponente in Dateisystemprotokolle schreibt. Sie können beispielsweise `DEBUG` Level-Logs angeben, um bei der Problembehandlung zu helfen, oder Sie können `ERROR` Level-Logs angeben, um die Anzahl der Logs zu reduzieren, die ein Core-Gerät erstellt. Weitere Informationen finden Sie im Log-Level-Parameter (`logging.level`) der Greengrass-Nukleus-Komponente.

  Sie können auch die minimale Protokollebene konfigurieren, die die Log-Manager-Komponente in CloudWatch Logs schreibt. Sie können beispielsweise eine höhere Protokollebene angeben, um die Protokollierungskosten zu senken. Weitere Informationen finden Sie im `minimumLogLevel` Parameter der Log Manager-Komponente, den Sie für AWS IoT Greengrass Core-Softwareprotokolle und Komponentenprotokolle angeben können.

- Intervall, in dem nach Protokollen gesucht werden soll, die in die CloudWatch Protokolle geschrieben werden sollen

  Um die Häufigkeit, mit der die Log Manager-Komponente Protokolle in Logs schreibt, zu CloudWatch erhöhen oder zu verringern, können Sie das Intervall konfigurieren, in dem sie nach neuen Protokolldateien sucht, die geschrieben werden sollen. Sie können beispielsweise ein niedrigeres Intervall angeben, um CloudWatch Protokolle früher als mit dem Standardintervall von 5 Minuten in Logs anzuzeigen. Sie können ein höheres Intervall angeben, um die Kosten zu senken, da die Log Manager-Komponente Protokolldateien in weniger Anfragen stapelt. Weitere Informationen finden Sie im Upload-Intervallparameter (`periodicUploadIntervalSec`) der Log Manager-Komponente.

- Format des Protokolls

Sie können wählen, ob die AWS IoT Greengrass Core-Software Protokolle im Text- oder JSON-Format schreibt. Wählen Sie das Textformat, wenn Sie Protokolle lesen, oder das JSON-Format, wenn Sie eine Anwendung zum Lesen oder Analysieren von Protokollen verwenden. Weitere Informationen finden Sie im Protokollformatparameter (`logging.format`) der Greengrass-Nukleus-Komponente.

- Ordner mit lokalen Dateisystemprotokollen

  Sie können den Protokollordner von */greengrass/v2*`/logs` einem anderen Ordner auf dem Kerngerät ändern. Weitere Informationen finden Sie im Ausgabeverzeichnisparameter (`logging.outputDirectory`) der Greengrass-Nukleus-Komponente.

AWS CloudTrail Logs

AWS IoT Greengrass integriert sich in AWS CloudTrail, einen Dienst, der eine Aufzeichnung der Aktionen bereitstellt, die von einem Benutzer, einer Rolle oder AWS-Service in ausgeführt wurden AWS IoT Greengrass. Weitere Informationen finden Sie unter AWS IoT Greengrass V2 API-Aufrufe protokollieren mit AWS CloudTrail.

AWS IoT Greengrass V2 API-Aufrufe protokollieren mit AWS CloudTrail

AWS IoT Greengrass V2 ist in einen Dienst integriert AWS CloudTrail, der eine Aufzeichnung der Aktionen bereitstellt, die ein Benutzer, eine Rolle oder ein AWS Dienst in AWS IoT Greengrass Version 2 ausgeführt hat. CloudTrail erfasst alle API-Aufrufe AWS IoT Greengrass als Ereignisse. Zu den aufgezeichneten Aufrufen gehören Aufrufe von der AWS IoT Greengrass Konsole und Codeaufrufen für die AWS IoT Greengrass API-Operationen.

Wenn Sie einen Trail erstellen, können Sie die kontinuierliche Bereitstellung von CloudTrail Ereignissen an einen S3-Bucket aktivieren, einschließlich Ereignissen für AWS IoT Greengrass. Wenn Sie keinen Trail konfigurieren, können Sie die neuesten Ereignisse trotzdem in der CloudTrail Konsole im Ereignisverlauf anzeigen. Anhand der von gesammelten Informationen können Sie die Anfrage ermitteln CloudTrail, an die die Anfrage gestellt wurde AWS IoT Greengrass, die IP-Adresse, von der aus die Anfrage gestellt wurde, wer die Anfrage gestellt hat, wann sie gestellt wurde, und weitere Details.

Weitere Informationen zu CloudTrail finden Sie im AWS CloudTrail Benutzerhandbuch.

Themen

- AWS IoT Greengrass V2 Informationen in CloudTrail
- AWS IoT Greengrass Datenereignisse in CloudTrail
- AWS IoT Greengrass Verwaltungsereignisse in CloudTrail
- Grundlegendes zu AWS IoT Greengrass V2 Einträgen in Protokolldateien

AWS IoT Greengrass V2 Informationen in CloudTrail

CloudTrail ist auf Ihrem aktiviert AWS-Konto , wenn Sie das Konto erstellen. Wenn eine Aktivität in stattfindet AWS IoT Greengrass, wird diese Aktivität zusammen mit anderen AWS Serviceereignissen in der CloudTrail Ereignishistorie in einem Ereignis aufgezeichnet. Sie können in Ihrem AWS-Konto die neusten Ereignisse anzeigen, suchen und herunterladen. Weitere Informationen finden Sie unter Ereignisse mit dem CloudTrail Ereignisverlauf anzeigen.

Für eine fortlaufende Aufzeichnung der Ereignisse in Ihrem AWS-Konto, einschließlich der Ereignisse für AWS IoT Greengrass, erstellen Sie einen Trail. Ein Trail ermöglicht CloudTrail die Übermittlung von Protokolldateien an einen S3-Bucket. Wenn Sie einen Trail in der Konsole erstellen, gilt der Trail standardmäßig für alle AWS-Region s. Der Trail protokolliert Ereignisse aus allen Regionen der AWS Partition und übermittelt die Protokolldateien an den von Ihnen angegebenen S3-Bucket. Darüber hinaus können Sie andere AWS Dienste konfigurieren, um die in den CloudTrail Protokollen gesammelten Ereignisdaten weiter zu analysieren und darauf zu reagieren. Weitere Informationen finden Sie hier:

- Übersicht zum Erstellen eines Trails
- CloudTrail unterstützte Dienste und Integrationen
- Konfiguration von Amazon SNS SNS-Benachrichtigungen für CloudTrail
- Empfangen von CloudTrail Protokolldateien aus mehreren Regionen und Empfangen von CloudTrail Protokolldateien von mehreren Konten

Alle AWS IoT Greengrass V2 Aktionen werden von der AWS IoT Greengrass V2 API-Referenz protokolliert CloudTrail und sind in dieser dokumentiert. Beispielsweise generieren Aufrufe von `CreateDeployment` und `CancelDeployment` Aktionen Einträge in den CloudTrail Protokolldateien. `CreateComponentVersion`

Jeder Ereignis- oder Protokolleintrag enthält Informationen zu dem Benutzer, der die Anforderung generiert hat. Die Identitätsinformationen unterstützen Sie bei der Ermittlung der folgenden Punkte:

- Ob die Anfrage mit Root- oder AWS Identity and Access Management (IAM-) Benutzeranmeldedaten gestellt wurde.
- Gibt an, ob die Anforderung mit temporären Sicherheitsanmeldeinformationen für eine Rolle oder einen Verbundbenutzer gesendet wurde.
- Ob die Anfrage von einem anderen AWS Dienst gestellt wurde.

Weitere Informationen finden Sie unter CloudTrail -Element userIdentity.

AWS IoT Greengrass Datenereignisse in CloudTrail

Datenereignisse liefern Informationen über die Ressourcenoperationen, die auf 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. Protokolliert standardmäßig CloudTrail keine Datenereignisse. Der CloudTrail Ereignisverlauf zeichnet keine Datenereignisse auf.

Für Datenereignisse werden zusätzliche Gebühren fällig. Weitere Informationen zur CloudTrail Preisgestaltung finden Sie unter AWS CloudTrail Preisgestaltung.

Sie können Datenereignisse für die AWS IoT Greengrass Ressourcentypen mithilfe der CloudTrail Konsole oder CloudTrail API-Operationen protokollieren. AWS CLI Die Tabelle in diesem Abschnitt zeigt die verfügbaren Ressourcentypen für AWS IoT Greengrass.

- Um Datenereignisse mithilfe 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 aus der Liste Datenereignistyp den Ressourcentyp aus, für den Sie Datenereignisse protokollieren möchten.
  3. Wählen Sie die Protokollauswahlvorlage aus, die Sie verwenden möchten. Sie können alle Datenereignisse für den Ressourcentyp protokollieren, alle `readOnly` Ereignisse protokollieren, alle `writeOnly` Ereignisse protokollieren oder eine benutzerdefinierte Protokollauswahlvorlage erstellen, um nach den Feldern `readOnly`eventName, und `resources.`ARN zu filtern.
- Um Datenereignisse mithilfe von zu protokollieren AWS CLI, konfigurieren Sie den `--advanced-event-selectors` Parameter so, dass das `eventCategory` Feld dem Ressourcentypwert

entspricht `Data` und das `resources.type` Feld dem Ressourcentypwert entspricht (siehe Tabelle). Sie können Bedingungen hinzufügen, um nach den Werten der `resources.ARN` Felder `readOnlyeventName`, und zu filtern.

- Um einen Trail zum Protokollieren von Datenereignissen zu konfigurieren, führen Sie den put-event-selectorsBefehl. Weitere Informationen finden Sie unter Protokollieren von Datenereignissen für Pfade mit dem AWS CLI.
- Um einen Ereignisdatenspeicher für die Protokollierung von Datenereignissen zu konfigurieren, führen Sie den create-event-data-storeBefehl zum Erstellen eines neuen Ereignisdatenspeichers zum Protokollieren von Datenereignissen, oder führen Sie den update-event-data-storeBefehl zum Aktualisieren eines vorhandenen Ereignisdatenspeichers. Weitere Informationen finden Sie unter Protokollieren von Datenereignissen für Ereignisdatenspeicher mit dem 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 auf der CloudTrail Konsole ausgewählt werden kann. In der Wertspalte resources.type wird der `resources.type` Wert angezeigt, den Sie bei der Konfiguration erweiterter Event-Selektoren mithilfe von oder angeben würden. AWS CLI CloudTrail APIs In der CloudTrail Spalte APIs Protokollierte Daten werden die API-Aufrufe angezeigt, die CloudTrail für den Ressourcentyp protokolliert wurden.

Typ des Datenereignisses (Konsole)	resources.type-Wert	Daten, die APIs protokolliert wurden CloudTrail
IoT-Zertifikat	`AWS::IoT::Certificate`	• VerifyClientDeviceIdentity • VerifyClientDeviceIoTCertificateAssociation
IoT Greengrass-Komponentenversion	`AWS::GreengrassV2::ComponentVersion`	• ResolveComponentCandidates
Einsatz von IoT Greengrass	`AWS::GreengrassV2::Deployment`	• GetDeploymentConfiguration
IoT-Sache	`AWS::IoT::Thing`	• ListThingGroupsForCoreDevices • PutCertificateAuthorities

Typ des Datenereignisses (Konsole)	resources.type-Wert	Daten, die APIs protokolliert wurden CloudTrail
		• VerifyClientDeviceIoTCertificateAssociation

Sie können erweiterte Event-Selektoren so konfigurieren, dass sie nach den Feldern `eventName`, `readOnly` und `resources.ARN` filtern, sodass nur die Ereignisse protokolliert werden, die für Sie wichtig sind.

Fügen Sie einen Filter hinzu`eventName`, um bestimmte Daten ein- oder auszuschließen APIs.

Weitere Informationen zu diesen Feldern finden Sie unter AdvancedFieldSelector.

Die folgenden Beispiele zeigen, wie Sie erweiterte Selektoren mit dem AWS CLI konfigurieren. Ersetzen Sie *`TrailName`* und *`region`* durch Ihre eigenen Informationen.

Example— Datenereignisse für IoT-Dinge protokollieren

```
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 Sie nach einer bestimmten IoT-Ding-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 Sie alle 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 enthalten Informationen über Verwaltungsvorgänge, die mit Ressourcen in Ihrem AWS Konto ausgeführt werden. Sie werden auch als Vorgänge auf Steuerebene bezeichnet. CloudTrail Protokolliert standardmäßig Verwaltungsereignisse.

AWS IoT Greengrass protokolliert alle Operationen auf der AWS IoT Greengrass Steuerungsebene als Verwaltungsereignisse. Eine Liste der Vorgänge auf der AWS IoT Greengrass Steuerungsebene, bei AWS IoT Greengrass denen die Anmeldung CloudTrail erfolgt, finden Sie in der AWS IoT Greengrass API-Referenz, Version 2.

Grundlegendes zu AWS IoT Greengrass V2 Einträgen in Protokolldateien

Ein Trail ist eine Konfiguration, die die Übertragung 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 enthalten Informationen über die angeforderte Aktion, Datum und Uhrzeit der Aktion, Anforderungsparameter usw. CloudTrail Protokolldateien sind kein geordneter Stack-Trace der öffentlichen API-Aufrufe, sodass sie nicht in einer bestimmten Reihenfolge angezeigt werden.

Das folgende Beispiel zeigt einen CloudTrail Protokolleintrag, der die `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"
}
```


Erfassung von Telemetriedaten zur Systemintegrität von AWS IoT Greengrass Kerngeräten

Telemetriedaten zur Systemintegrität sind Diagnosedaten, die Ihnen helfen können, die Leistung kritischer Operationen auf Ihren Greengrass-Kerngeräten zu überwachen. Sie können Projekte und Anwendungen erstellen, um Telemetriedaten von Ihren Edge-Geräten abzurufen, zu analysieren, zu transformieren und zu melden. Branchenexperten, wie z. B. Verfahrenstechniker, können diese Anwendungen nutzen, um Einblicke in den Zustand der Flotte zu gewinnen.

Sie können die folgenden Methoden verwenden, um Telemetriedaten von Ihren Greengrass-Core-Geräten zu sammeln:

- Nucleus-Telemetrie-Emitter-Komponente — Die Nucleus-Telemetrie-Emitter-Komponente (`aws.greengrass.telemetry.NucleusEmitter`) auf einem Greengrass-Core-Gerät veröffentlicht standardmäßig Telemetriedaten zum Thema. `$local/greengrass/telemetry` Sie können die zu diesem Thema veröffentlichten Daten verwenden, um lokal auf Ihrem Core-Gerät zu agieren, auch wenn Ihr Gerät nur 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 Nucle-Emitter-Komponente auf einem Kerngerät bereitstellen, um Telemetriedaten zu veröffentlichen. Mit der Veröffentlichung von Telemetriedaten zum lokalen Thema sind keine Kosten verbunden. Die Verwendung eines MQTT-Themas zur Veröffentlichung von Daten im AWS Cloud ist jedoch kostenpflichtig.AWS IoT Core

  AWS IoT Greengrass bietet mehrere Community-Komponenten, mit denen Sie Telemetriedaten mithilfe von InfluxDB und Grafana lokal auf Ihrem Kerngerät analysieren und visualisieren können. Diese Komponenten verwenden Telemetriedaten von der Nukleus-Emitter-Komponente. Weitere Informationen finden Sie in der README-Datei für die InfluxDB-Publisher-Komponente.
- Telemetrieagent — Der Telemetrieagent auf Greengrass-Core-Geräten sammelt lokale Telemetriedaten und veröffentlicht sie an Amazon, EventBridge ohne dass eine Kundeninteraktion erforderlich ist. Kerngeräte veröffentlichen Telemetriedaten nach EventBridge bestem Wissen und Gewissen. Beispielsweise können Core-Geräte Telemetriedaten möglicherweise nicht liefern, wenn sie offline sind.

  Die Telemetrie-Agent-Funktion ist standardmäßig für alle Greengrass-Core-Geräte aktiviert. Sie beginnen automatisch mit dem Empfang von Daten, sobald Sie ein Greengrass-Core-Gerät eingerichtet haben. Abgesehen von den Kosten für die Datenverbindung AWS IoT Core ist die Datenübertragung vom Kerngerät zum Gerät kostenlos. Das liegt daran, dass der Agent zu einem AWS reservierten Thema veröffentlicht. Abhängig von Ihrem Anwendungsfall können jedoch Kosten anfallen, wenn Sie die Daten erhalten oder verarbeiten.

  > **Note**
  >
  > Amazon EventBridge ist ein Eventbus-Service, mit dem Sie Ihre Anwendungen mit Daten aus einer Vielzahl von Quellen verbinden können, z. B. mit Greengrass-Kerngeräten. Weitere Informationen finden Sie unter Was ist Amazon EventBridge? im EventBridge Amazon-Benutzerhandbuch.

Um sicherzustellen, dass die AWS IoT Greengrass Core-Software ordnungsgemäß funktioniert, werden die Daten für Entwicklungs- und Qualitätsverbesserungszwecke AWS IoT Greengrass verwendet. Diese Funktion hilft auch bei der Entwicklung neuer und verbesserter Edge-Funktionen. AWS IoT Greengrass speichert Telemetriedaten für bis zu sieben Tage.

In diesem Abschnitt wird beschrieben, wie der Telemetrieagent konfiguriert und verwendet wird. Informationen zur Konfiguration der Nucleus-Telemetrie-Emitter-Komponente finden Sie unter. Nucleus-Telemetrie-Sender

Themen

- Telemetrie-Metriken
- Konfigurieren Sie die Einstellungen des Telemetrie-Agents
- Abonnieren Sie Telemetriedaten in EventBridge

Telemetrie-Metriken

In der folgenden Tabelle werden die Metriken beschrieben, die vom Telemetrieagent veröffentlicht werden.

Name	Beschreibung	
System (System)		
`SystemMemUsage`	Die Menge an Speicher, die derzeit von allen Anwendung en auf dem Greengrass-Core-Gerät verwendet wird, einschließlich des Betriebss ystems.	
`CpuUsage`	Die Menge an CPU, die derzeit von allen Anwendung en auf dem Greengrass-Core-Gerät verwendet wird, einschließlich des Betriebss ystems.	

Name	Beschreibung
`TotalNumberOfFDs`	Die Anzahl der Dateidesk riptoren, die vom Betriebss ystem des Greengrass-Core-Geräts gespeichert werden. Ein Dateideskriptor identifiziert eindeutig eine geöffnete Datei.
Grüngraskern	
`NumberOfComponents Running`	Die Anzahl der Komponenten, die auf dem Greengrass-Core-Gerät ausgeführt werden.
`NumberOfComponents Errored`	Die Anzahl der Komponenten, die sich auf dem Greengras s-Core-Gerät im Fehlerstatus befinden.
`NumberOfComponents Installed`	Die Anzahl der Komponenten, die auf dem Greengrass-Core-Gerät installiert sind.
`NumberOfComponents Starting`	Die Anzahl der Komponenten, die auf dem Greengrass-Core-Gerät gestartet werden.
`NumberOfComponents New`	Die Anzahl der Komponenten, die auf dem Greengrass-Core-Gerät neu sind.
`NumberOfComponents Stopping`	Die Anzahl der Komponenten, die auf dem Greengrass-Core-Gerät gestoppt werden.
`NumberOfComponents Finished`	Die Anzahl der Komponenten, die auf dem Greengrass-Core-Gerät fertiggestellt wurden.

Name	Beschreibung	
`NumberOfComponents Broken`	Die Anzahl der Komponenten, die auf dem Greengrass-Core-Gerät defekt sind.	
`NumberOfComponents Stateless`	Die Anzahl der Komponenten, die auf dem Greengrass-Core-Gerät zustandslos sind.	
Authentifizierung auf Clientgeräten — Für diese Funktion ist Version 2.4.0 oder höher der Authentifizierungskomponente für das Clientgerät erforderlich.		
`VerifyClientDeviceIdentity.Success`	Gibt an, wie oft überprüft wurde, ob die Identität des Client-Geräts erfolgreich war.	
`VerifyClientDeviceIdentity.Failure`	Gibt an, wie oft bei der Überprüfung, ob die Identität des Client-Geräts fehlgeschlagen ist, fehlgeschlagen ist.	
`AuthorizeClientDeviceActions.Success`	Gibt an, wie oft das Client-Gerät autorisiert wurde, angeforderte Aktionen auszuführen.	
`AuthorizeClientDeviceActions.Failure`	Gibt an, wie oft das Client-Gerät nicht autorisiert ist, angeforderte Aktionen auszuführen.	

Name	Beschreibung
`GetClientDeviceAuthToken.Success`	Gibt an, wie oft das Client-Gerät erfolgreich authentifiziert wurde.
`GetClientDeviceAuthToken.Failure`	Gibt an, wie oft das Client-Gerät nicht authentifiziert werden kann.
`SubscribeToCertificateUpdates.Success`	Die Anzahl der erfolgreichen Abonnements für Zertifikatsupdates.
`SubscribeToCertificateUpdates.Failure`	Die Anzahl der erfolglosen Versuche, Zertifikatsaktualisierungen zu abonnieren.
`ServiceError`	Die Anzahl der unbehandelten internen Fehler bei der Authentifizierung auf dem Client-Gerät.
Stream-Manager — Für diese Funktion ist Version 2.7.0 oder höher der Greengrass Nucleus-Komponente erforderlich.	
`BytesAppended`	Die Anzahl der Datenbytes, die an den Stream-Manager angehängt wurden.
`BytesUploadedToIoTAnalytics`	Die Anzahl der Datenbytes, die der Stream-Manager in AWS IoT Analytics Kanäle exportiert.

Name	Beschreibung	
`BytesUploadedToKinesis`	Die Anzahl der Datenbytes, die Stream Manager in Streams in Amazon Kinesis Data Streams exportiert.	
`BytesUploadedToIoTSiteWise`	Die Anzahl der Datenbytes, in die Stream Manager in AWS IoT SiteWise Asset-Eigenschaften exportiert.	
`BytesUploadedToS3`	Die Anzahl der Datenbytes, die der Stream-Manager in Objekte in Amazon S3 exportiert.	
Systemmetriken — Für diese Funktion ist Version 2.15.0 oder höher der Greengrass Nucleus-Komponente erforderlich.		
`CPUArchitecture`	Architektur der Zentraleinheit des Geräts.	
`Family`	Gerätebetriebssystemfamilie (nur Windows).	
`KernelVersion`	Version des Gerätekernels (nur Unix).	
`KnowledgeBaseArticles`	Knowledge Base-Artikel, die auf einem Gerät installiert sind (nur Windows).	
`OSBuildMajor`	Haupt-Buildnummer der Betriebssystemversion (nur Windows).	

Name	Beschreibung
`OSBuildMinor`	Untergeordnete Buildnummer der Betriebssystemversion (nur Windows).
`OSName`	Name des Betriebssystems des Geräts.
`OSVersion`	Marketingversion des Gerätebetriebssystems.

Konfigurieren Sie die Einstellungen des Telemetrie-Agents

Der Telemetrieagent verwendet die folgenden Standardeinstellungen:

- Der Telemetrieagent aggregiert jede Stunde Telemetriedaten.
- Der Telemetrieagent veröffentlicht alle 24 Stunden eine Telemetriemeldung.

Der Telemetrieagent veröffentlicht Daten mithilfe des MQTT-Protokolls mit einer QoS-Stufe (Quality of Service) von 0, was bedeutet, dass er weder die Zustellung bestätigt noch versucht, die Veröffentlichung zu wiederholen. Telemetrienachrichten teilen sich eine MQTT-Verbindung mit anderen Nachrichten für Abonnements, die dafür bestimmt sind. AWS IoT Core

Abgesehen von den Kosten für die Datenverbindung AWS IoT Core ist der Datentransfer vom Core zum Core kostenlos. Das liegt daran, dass der Agent zu einem AWS reservierten Thema veröffentlicht. Abhängig von Ihrem Anwendungsfall können jedoch Kosten anfallen, wenn Sie die Daten erhalten oder verarbeiten.

Sie können die Telemetrie-Agent-Funktion für jedes Greengrass Core-Gerät aktivieren oder deaktivieren. Sie können auch die Intervalle konfigurieren, in denen das Kerngerät Daten aggregiert und veröffentlicht. Um Telemetrie zu konfigurieren, passen Sie den Telemetriekonfigurationsparameter an, wenn Sie die Greengrass Nucleus-Komponente bereitstellen.

Abonnieren Sie Telemetriedaten in EventBridge

Sie können in Amazon Regeln erstellen EventBridge , die definieren, wie Telemetriedaten verarbeitet werden, die vom Telemetrie-Agenten auf dem Greengrass-Core-Gerät veröffentlicht wurden.

Wenn es die Daten EventBridge empfängt, ruft es die in Ihren Regeln definierten Zielaktionen auf. Sie können beispielsweise Ereignisregeln erstellen, die Benachrichtigungen senden, Ereignisinformationen speichern, Korrekturmaßnahmen ergreifen oder andere Ereignisse auslösen.

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"
        }
      ]
    }
```

```
      {
        "TS": 1750104449426,
        "NS": "SystemMetrics",
        "M": [
          {
            "N": "KernelVersion",
            "Sum": 1,
            "U": "6.1.140-154.222.amzn2023.x86_64"
          },
          {
            "N": "OSVersion",
            "Sum": 1,
            "U": "2023.7.20250609"
          },
          {
            "N": "OSName",
            "Sum": 1,
            "U": "Amazon Linux"
          },
          {
            "N": "CPUArchitecture",
            "Sum": 1,
            "U": "Broadwell (Server)"
          },
        ]
      }
    ]
  }
}
```

Das ADP Array enthält eine Liste von aggregierten Datenpunkten mit den folgenden Eigenschaften:

TS

Der Zeitstempel, zu dem die Daten gesammelt wurden.

NS

Der Metrik-Namespace.

M

Die Liste der Metriken. Eine Metrik enthält die folgenden Eigenschaften:

`N`

Name der Metrik.

`Sum`

Die Summe der Werte der Metrik in diesem Telemetrieereignis.

`U`

Die Einheit des metrischen Werts.

Weitere Informationen zu den einzelnen Metriken finden Sie unterTelemetrie-Metriken.

Voraussetzungen für die Erstellung von EventBridge Regeln

Bevor Sie eine EventBridge Regel für erstellen AWS IoT Greengrass, sollten Sie Folgendes tun:

- Machen Sie sich mit Ereignissen, Regeln und Zielen in vertraut EventBridge.
- Erstellen und konfigurieren Sie die Ziele, die durch Ihre EventBridge Regeln aufgerufen werden. Regeln können viele Arten von Zielen aufrufen, z. B. Amazon Kinesis Kinesis-Streams, AWS Lambda Funktionen, Amazon SNS SNS-Themen und Amazon SQS SQS-Warteschlangen.

  Ihre EventBridge Regel und die zugehörigen Ziele müssen sich dort befinden, AWS-Region wo Sie Ihre Greengrass-Ressourcen erstellt haben. Weitere Informationen finden Sie unter Dienstendpunkte und Kontingente in der. Allgemeine AWS-Referenz

Weitere Informationen finden Sie unter Was ist Amazon EventBridge? und Erste Schritte mit Amazon EventBridge im EventBridge Amazon-Benutzerhandbuch.

Erstellen Sie eine Ereignisregel zum Abrufen von Telemetriedaten (Konsole)

Verwenden Sie die folgenden Schritte, um mithilfe der eine EventBridge Regel AWS-Managementkonsole zu erstellen, die vom Greengrass-Core-Gerät veröffentlichte Telemetriedaten empfängt. Auf diese Weise können Webserver, E-Mail-Adressen und andere Themenabonnenten auf das Ereignis reagieren. Weitere Informationen finden Sie im EventBridge Amazon-Benutzerhandbuch unter Erstellen einer EventBridge Regel, die bei einem Ereignis von einer AWS Ressource ausgelöst wird.

1. Öffnen Sie die EventBridgeAmazon-Konsole und wählen Sie Regel erstellen.

2. Geben Sie unter Name and description (Name und Beschreibung) einen Namen und eine Beschreibung für die Regel ein.
3. 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 als Ereignistyp Greengrass Telemetry Data aus.
4. Behalten Sie unter Select event bus (Ereignisbus auswählen) die standardmäßigen Ereignisbusoptionen bei.
5. Konfigurieren Sie unter Select targets (Ziele auswählen) das Ziel. Das folgende Beispiel verwendet eine Amazon SQS SQS-Warteschlange, Sie können jedoch auch andere Zieltypen konfigurieren.
   a. Wählen Sie für Target die Option SQS-Warteschlange aus.
   b. Wählen Sie für Queue* Ihre Zielwarteschlange aus.
6. Definieren Sie unter Tags - optional Tags für die Regel oder lassen Sie die Felder leer.
7. Wählen Sie Erstellen aus.

Erstellen Sie eine Ereignisregel zum Abrufen von Telemetriedaten (CLI)

Verwenden Sie die folgenden Schritte, um mithilfe der eine EventBridge Regel AWS CLI zu erstellen, die 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.
   - *thing-name*Ersetzen Sie es durch den Dingnamen des Kerngeräts.

   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 auch andere Zieltypen konfigurieren.

   - *queue-arn*Ersetzen Sie es durch den ARN Ihrer Amazon SQS 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 EventBridge Ihre Zielwarteschlange aufrufen kann, müssen Sie Ihrem Thema eine ressourcenbasierte Richtlinie hinzufügen. Weitere Informationen finden Sie unter Amazon SQS SQS-Berechtigungen im EventBridge Amazon-Benutzerhandbuch.

Weitere Informationen finden Sie unter Ereignisse und Ereignismuster EventBridge im EventBridge Amazon-Benutzerhandbuch.

Erhalten Sie Benachrichtigungen über den Bereitstellungs- und Komponentenstatus

Mit den EventBridge Amazon-Veranstaltungsregeln erhalten Sie Benachrichtigungen über Statusänderungen Ihrer Greengrass-Bereitstellungen, die auf Ihren Geräten eingegangen sind, 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
- Ereignis zur Änderung des Komponentenstatus
- Voraussetzungen für die Erstellung von EventBridge Regeln
- Benachrichtigungen zum Gerätestatus konfigurieren (Konsole)
- Benachrichtigungen zum Gerätestatus (CLI) konfigurieren
- Benachrichtigungen zum Gerätestatus konfigurieren (CloudFormation)
- Weitere Informationen finden Sie auch unter

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` `COMPLETEDREJECTED`, 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 Fehlercodes für die Bereitstellung.

Bereitstellungsstatus

- `FAILED`. Die Bereitstellung ist fehlgeschlagen.
- `SUCCEEDED`. Die Bereitstellung für eine Dinggruppe 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 Fehlercodes für die Bereitstellung undDetaillierte Statuscodes für Komponenten.

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 Statuscodes für Komponenten.

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 Regeln aus.
3. Wählen Sie Regel erstellen aus.
4. Geben Sie einen Namen und eine Beschreibung für die Regel ein.

   Eine Regel darf nicht denselben Namen wie eine andere Regel in derselben Region und auf demselben Event Bus haben.
5. Wählen Sie als Event bus (Event Bus) den Event Bus aus, den Sie dieser Regel zuordnen möchten. Wenn Sie möchten, dass diese Regel mit Ereignissen aus Ihrem eigenen Konto übereinstimmt, wählen Sie AWS -Standard-Event-Bus aus. Wenn ein AWS Service in Ihrem Konto ein Ereignis ausgibt, wird es immer an den Standard-Event-Bus Ihres Kontos weitergeleitet.
6. Bei Regeltyp wählen Sie Regel mit einem Ereignismuster aus.
7. Wählen Sie Weiter.
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.
13. Bei 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 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.

   - *topic-arn*Ersetzen Sie es durch den ARN Ihres Amazon SNS SNS-Themas.

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

> **Note**
>
> Damit Amazon EventBridge Ihr Zielthema aufrufen kann, müssen Sie Ihrem Thema eine ressourcenbasierte Richtlinie hinzufügen. Weitere Informationen finden Sie unter Amazon SNS SNS-Berechtigungen im EventBridge Amazon-Benutzerhandbuch.

Weitere Informationen finden Sie unter Ereignisse und Ereignismuster EventBridge im EventBridge Amazon-Benutzerhandbuch.

Benachrichtigungen zum Gerätestatus konfigurieren (CloudFormation)

Verwenden Sie CloudFormation Vorlagen, um EventBridge Regeln zu erstellen, die Benachrichtigungen über Statusänderungen für Ihre Greengrass-Gruppenbereitstellungen senden. Weitere Informationen finden Sie in der Referenz zum EventBridge Amazon-Ressourcentyp im AWS CloudFormation Benutzerhandbuch.

Weitere Informationen finden Sie auch unter

- Überprüfen Sie den Bereitstellungsstatus des Geräts
- 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
- Überprüfen Sie den Zustand einer Kerngerätegruppe
- Überprüfen Sie den Status der Komponenten des Kerngeräts

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 unterAWS IoT Greengrass Komponenten entwickeln.

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`.

AWS Lambda Funktionen ausführen

> **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 Greengrass Kerngeräten ausgeführt werden. In den folgenden Fällen sollten Sie dies möglicherweise tun:

- Sie haben Anwendungscode in Lambda-Funktionen, den Sie auf Kerngeräten bereitstellen möchten.
- Sie haben AWS IoT Greengrass V1-Anwendungen, die Sie auf AWS IoT Greengrass V2 Kerngeräten ausführen möchten. Weitere Informationen finden Sie unter Schritt 2: AWS IoT Greengrass V2 Komponenten für die Migration AWS IoT Greengrass V1 von Anwendungen erstellen und bereitstellen.

Lambda-Funktionen beinhalten Abhängigkeiten von den folgenden Komponenten. Sie müssen diese Komponenten nicht als Abhängigkeiten definieren, wenn Sie die Funktion importieren. Wenn Sie die Lambda-Funktionskomponente bereitstellen, umfasst die Bereitstellung diese Lambda-Komponentenabhängigkeiten.

- Die Lambda-Launcher-Komponente (`aws.greengrass.LambdaLauncher`) kümmert sich um Prozesse und die Umgebungskonfiguration.
- 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

- Voraussetzungen
- Lambda-Funktionslebenszyklus konfigurieren
- Containerisierung der Lambda-Funktion konfigurieren
- Eine Lambda-Funktion als Komponente importieren (Konsole)

- Eine Lambda-Funktion als Komponente importieren ()AWS CLI

Voraussetzungen

Ihre Kerngeräte und Lambda-Funktionen müssen die folgenden Anforderungen erfüllen, damit Sie die Funktionen auf der AWS IoT Greengrass Core-Software ausführen können:

- 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.
- Sie müssen die Programmiersprachen, die die Lambda-Funktion verwendet, auf Ihren Kerngeräten installieren.

> **Tip**
>
> Sie können eine Komponente erstellen, die die Programmiersprache installiert, und diese Komponente dann als Abhängigkeit von Ihrer Lambda-Funktionskomponente angeben. Greengrass unterstützt alle von Lambda unterstützten Versionen von Python-, Node.js- und Java-Runtimes. Greengrass wendet keine zusätzlichen Einschränkungen auf veraltete Lambda-Laufzeitversionen an. Sie können Lambda-Funktionen, die diese veralteten Laufzeiten verwenden AWS IoT Greengrass, ausführen, aber Sie können sie nicht darin erstellen. AWS Lambda Weitere Informationen zur AWS IoT Greengrass Unterstützung von Lambda-Laufzeiten finden Sie unter. AWS Lambda Funktionen ausführen

Lambda-Funktionslebenszyklus konfigurieren

Der Greengrass Lambda-Funktionslebenszyklus 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 enden, wenn keine Aufgaben mehr zur Ausführung übrig sind. Bei jedem Aufruf der Funktion wird ein separater Container, auch Sandbox genannt, für die Verarbeitung von Aufrufen erstellt, sofern kein vorhandener Container zur Wiederverwendung verfügbar ist. 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 fixierte) Funktionen beginnen, wenn die AWS IoT Greengrass Core-Software gestartet 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 bei jedem Aufruf des Handlers beibehalten.

  Verwenden Sie langlebige Lambda-Funktionen, wenn Sie ohne anfängliche Eingaben mit der Arbeit beginnen müssen. Beispielsweise kann eine langlebige Funktion ein Modell für maschinelles Lernen laden und mit der Verarbeitung beginnen, sodass es bereit ist, wenn die Funktion Gerätedaten empfängt.

  > **Note**
  >
  > Langlebige Funktionen haben Timeouts, die mit jedem Aufruf ihres Handlers verknüpft sind. Wenn Sie Code aufrufen möchten, der unbegrenzt läuft, müssen Sie ihn außerhalb des Handlers starten. Stellen Sie sicher, dass es außerhalb des Handlers keinen Blockierungscode gibt, der die Initialisierung der Funktion verhindern könnte.
  > Diese Funktionen werden ausgeführt, sofern die AWS IoT Greengrass Core-Software nicht angehalten wird, z. B. während einer Bereitstellung oder eines Neustarts. Diese Funktionen können nicht ausgeführt werden, wenn die Funktion auf eine nicht abgefangene Ausnahme stößt, ihre Speichergrenzen überschreitet oder in einen Fehlerstatus übergeht, z. B. ein Handler-Timeout.

Weitere Informationen zur Wiederverwendung von Containern finden Sie AWS Lambda im AWS Compute-Blog unter Understanding Container Reuse.

Containerisierung der Lambda-Funktion konfigurieren

Standardmäßig werden Lambda-Funktionen innerhalb eines AWS IoT Greengrass Containers ausgeführt. Greengrass-Container sorgen für eine Isolierung zwischen Ihren Funktionen und dem Host. Diese Isolierung erhöht die Sicherheit sowohl für den Host als auch für die Funktionen im Container.

Wir empfehlen, dass Sie Lambda-Funktionen in einem Greengrass-Container ausführen, es sei denn, Ihr Anwendungsfall erfordert, dass sie ohne Containerisierung ausgeführt werden. Wenn Sie Ihre Lambda-Funktionen in einem Greengrass-Container ausführen, haben Sie mehr Kontrolle 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 laufen, das den Container-Modus nicht unterstützt. Ein Beispiel wäre, wenn Sie eine spezielle Linux-Distribution verwenden oder eine frühere Kernelversion verwenden möchten, die veraltet ist.
- Sie möchten Ihre Lambda-Funktion in einer anderen Containerumgebung mit eigenem OverlayFS ausführen, stoßen aber bei der Ausführung in einem Greengrass-Container auf OverlayFS-Konflikte.
- Sie benötigen Zugriff auf lokale Ressourcen 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 steckbares Gerät.
- Sie haben eine frühere Anwendung, die als Prozess geschrieben wurde, und Sie stoßen auf Probleme, 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.

Containerisierung	Hinweise
	• Lambda-Funktionen, die in einem Greengrass-Container ausgeführt werden, haben keinen Zugriff auf den bereitgestellten Code anderer Lambda-Funktionen, selbst wenn sie mit derselben Systemgruppe ausgeführt werden. Mit anderen Worten, Ihre Lambda-Funktionen werden stärker voneinander isoliert ausgeführt. • Da die AWS IoT Greengrass Core-Software alle untergeordneten Prozesse im selben Container wie die Lambda-Funktion ausführt, werden die untergeordneten Prozesse gestoppt, wenn die Lambda-Funktion beendet wird.

Containerisierung	Hinweise

Kein Container	• Die folgenden Funktionen sind für Lambda-Funktionen, die nicht containerisiert sind, nicht verfügbar:
	  • Speichergrenzen für Lambda-Funktionen.
	  • Lokale Geräte- und Volume-Ressourcen Sie müssen auf diese Ressourcen über ihre Dateipfade auf dem Kerngerät zugreifen und nicht als Lambda-Fu nktionsressourcen.
	• Wenn Ihre nicht containerisierte Lambda-Funktion auf eine maschinelle Lernresso urce zugreift, müssen Sie einen Ressource nbesitzer identifizieren und Zugriffsberechtigu ngen für die Ressource festlegen, nicht für die Lambda-Funktion.
	• Lambda-Funktionen, die nicht containerisiert sind, haben nur Lesezugriff auf den bereitges tellten Code anderer Lambda-Funktionen, 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 ändern, dass eine Lambda-Funktion nicht mehr in einem Greengrass-Container ausgeführt wird, sondern ohne Containerisierung ausgeführt wird, werden die Speicherlimits der Funktion verworfen. Sie müssen direkt auf das Dateisystem zugreifen, anstatt verknüpfte lokale Ressourcen zu verwenden. Sie müssen alle angehängten Ressourcen entfernen, bevor Sie die Lambda-Funktion bereitstellen.
- Wenn Sie eine Lambda-Funktion von der Ausführung ohne Containerisierung auf die Ausführung in einem Container ändern, verliert Ihre Lambda-Funktion den direkten Zugriff auf das Dateisystem.

Sie müssen für jede Funktion ein Speicherlimit definieren oder das standardmäßige Speicherlimit 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, setzen Sie den Wert des `containerMode` Konfigurationsparameters auf eine der folgenden Optionen, wenn Sie die Komponente bereitstellen.

- `NoContainer`— Die Komponente wird nicht in einer isolierten Laufzeitumgebung ausgeführt.
- `GreengrassContainer`— Die Komponente läuft in einer isolierten Laufzeitumgebung innerhalb des AWS IoT Greengrass Containers.

Weitere Informationen zur Bereitstellung und Konfiguration von Komponenten finden Sie unter AWS IoT Greengrass Komponenten auf Geräten bereitstellen und Komponentenkonfigurationen aktualisieren.

Eine Lambda-Funktion als Komponente importieren (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, um eine Komponente zu erstellen, die auf Ihrem Greengrass-Gerät ausgeführt wird.

Bevor Sie beginnen, überprüfen Sie die Anforderungen für die Ausführung von Lambda-Funktionen auf Greengrass-Geräten.

Aufgaben

- Schritt 1: Wählen Sie eine Lambda-Funktion für den Import
- Schritt 2: Lambda-Funktionsparameter konfigurieren
- Schritt 3: (Optional) Geben Sie die unterstützten Plattformen für die Lambda-Funktion an
- Schritt 4: (Optional) Komponentenabhängigkeiten für die Lambda-Funktion angeben
- Schritt 5: (Optional) Führen Sie die Lambda-Funktion in einem Container aus
- Schritt 6: Erstellen Sie die Lambda-Funktionskomponente

Schritt 1: Wählen Sie eine Lambda-Funktion für den Import

1. Wählen Sie im Navigationsmenü der AWS IoT Greengrass Konsole die Option 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 der Lambda-Funktionsversion die zu importierende Version aus. Sie können keine Lambda-Aliase wie wählen. `$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: Lambda-Funktionsparameter konfigurieren

Konfigurieren Sie auf der Seite Komponente erstellen unter Lambda-Funktionskonfiguration die folgenden Parameter für die Ausführung der Lambda-Funktion.

1. (Optional) Fügen Sie die Liste der Ereignisquellen hinzu, die die Lambda-Funktion für geschäftliche Nachrichten abonniert. Sie können Ereignisquellen angeben, um diese Funktion für lokale Veröffentlichungs-/Abonnementnachrichten und MQTT-Nachrichten zu abonnieren. AWS IoT Core Die Lambda-Funktion wird aufgerufen, wenn sie eine Nachricht von einer Ereignisquelle empfängt.

   > **Note**
   >
   > Um diese Funktion für Nachrichten von anderen Lambda-Funktionen oder -Komponenten zu abonnieren, stellen Sie die ältere Abonnement-Router-Komponente bereit, wenn Sie diese Lambda-Funktionskomponente bereitstellen. Wenn Sie die ältere Abonnement-Router-Komponente 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, zu dem Nachrichten abonniert werden sollen.
- Typ — Der Typ der Ereignisquelle. Wählen Sie aus den folgenden Optionen aus:
  - Lokales Veröffentlichen/Abonnieren — Abonnieren Sie Nachrichten zum lokalen Veröffentlichen/Abonnieren.

    Wenn Sie Greengrass Nucleus v2.6.0 oder höher und Lambda Manager v2.2.5 oder höher verwenden, können Sie MQTT-Themen-Platzhalter (**+**und**#**) im Topic verwenden, wenn Sie diesen Typ angeben.
  - AWS IoT Core AWS IoT Core MQTT — Abonnieren Sie MQTT-Nachrichten.

    Sie können Platzhalter (+und#) für MQTT-Themen 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, klicken Sie neben der Ereignisquelle, die Sie entfernen möchten, auf Entfernen.

2. Geben Sie für Timeout (Sekunden) die maximale Zeit in Sekunden ein, die eine nicht fixierte Lambda-Funktion ausführen kann, bevor das Timeout eintritt. Der Standardwert ist 3 Sekunden.
3. Wählen Sie für Pinned aus, ob die Lambda-Funktionskomponente angeheftet ist. Die Standardeinstellung ist True.

   - Eine fixierte (oder langlebige) Lambda-Funktion AWS IoT Greengrass startet beim Start und läuft in ihrem eigenen Container weiter.
   - Eine Lambda-Funktion ohne Fixierung (oder On-Demand-Funktion) startet nur, wenn sie ein Arbeitselement empfängt, und wird beendet, wenn sie für eine festgelegte maximale Leerlaufzeit inaktiv bleibt. Wenn die Funktion mehrere Arbeitselemente hat, erstellt die AWS IoT Greengrass Core-Software mehrere Instanzen der Funktion.
4. (Optional) Stellen Sie unter Zusätzliche Parameter die folgenden Lambda-Funktionsparameter ein.

   - 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. Die Standardeinstellung ist 1.000 Nachrichten.
- Maximale Anzahl von Instanzen — Die maximale Anzahl von Instanzen, die eine nicht gepinnte Lambda-Funktion gleichzeitig ausführen kann. Die Standardeinstellung ist 100 Instanzen.
- Maximale Leerlaufzeit (Sekunden) — Die maximale Zeitspanne in Sekunden, die eine nicht fixierte Lambda-Funktion inaktiv sein kann, bevor die AWS IoT Greengrass Core-Software ihren Prozess stoppt. Standardmäßig ist ein Zeitraum von 60 Sekunden festgelegt.
- Kodierungstyp — Der Payload-Typ, den 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 an die Lambda-Funktion übergeben werden sollen, wenn sie ausgeführt wird.

   a. Wählen Sie unter Zusätzliche Parameter, Prozessargumente 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, die Option Entfernen aus.

6. (Optional) Geben Sie die Umgebungsvariablen an, die für die Lambda-Funktion verfügbar sind, wenn sie ausgeführt wird. 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 neben der Umgebungsvariablen, die Sie entfernen möchten, die Option Entfernen aus.

Schritt 3: (Optional) Geben Sie die unterstützten Plattformen für die Lambda-Funktion an

Alle Kerngeräte verfügen über 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 Kerngerät, um festzustellen, ob die Lambda-Funktion auf diesem Gerät unterstützt wird.

> **Note**
>
> Sie können auch benutzerdefinierte Plattformattribute angeben, wenn Sie die Greengrass Nucleus-Komponente auf einem Core-Gerät bereitstellen. Weitere Informationen finden Sie unter dem Parameter Platform Overrides der Greengrass Nucleus-Komponente.

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, die Option Entfernen aus.

Schritt 4: (Optional) Komponentenabhängigkeiten für die Lambda-Funktion angeben

Komponentenabhängigkeiten identifizieren zusätzliche, von AWS Ihnen bereitgestellte Komponenten oder benutzerdefinierte Komponenten, die Ihre Funktion verwendet. Wenn Sie die Lambda-Funktionskomponente bereitstellen, umfasst die Bereitstellung diese Abhängigkeiten, damit Ihre Funktion ausgeführt werden kann.

> **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. Geheimnisse, lokale Schatten und Stream-Manager. Definieren Sie diese Komponenten als feste Abhängigkeiten, sodass Ihre Lambda-Funktionskomponente neu gestartet wird, wenn sich der Status der Abhängigkeit ändert. Weitere Informationen finden Sie unter V1-Lambda-Funktionen importieren.

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 einzubeziehen.
   - Versionsanforderung — Die semantische Versionsbeschränkung im NPM-Stil, die die kompatiblen Versionen dieser Komponentenabhängigkeit identifiziert. Sie können eine einzelne Version oder einen Versionsbereich 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 Calculator.
   - Typ — Der Typ der Abhängigkeit. Wählen Sie aus den folgenden Optionen aus:
     - Hard — Die Lambda-Funktionskomponente wird neu gestartet, wenn sich der Status der Abhängigkeit ändert. Dies ist die Standardauswahl.

- Weich — Die Lambda-Funktionskomponente wird nicht neu gestartet, wenn sich der Status der Abhängigkeit ändert.

3. Um eine Komponentenabhängigkeit zu entfernen, wählen Sie neben der Komponentenabhängigkeit die Option Entfernen

Schritt 5: (Optional) Führen Sie die Lambda-Funktion in einem Container aus

Standardmäßig werden Lambda-Funktionen in einer isolierten Laufzeitumgebung innerhalb der AWS IoT Greengrass Core-Software ausgeführt. Sie können sich auch dafür entscheiden, die Lambda-Funktion als Prozess ohne Isolierung auszuführen (d. h. im Modus Kein Container).

Wählen Sie unter Linux-Prozesskonfiguration für den 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 jegliche 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, die dem Container zur Verfügung gestellt werden sollen.

   Gehen Sie unter Container-Parameter wie folgt vor.

   a. Geben Sie unter Speichergröße die Speichergröße ein, die Sie dem Container zuweisen möchten. Sie können die Speichergröße in MB oder kB angeben.
   b. Wählen Sie für den schreibgeschützten sys-Ordner aus, ob der Container Informationen aus dem Geräteordner lesen kann oder nicht. `/sys` Die Standardeinstellung 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 am Ziel innerhalb des Containers.

   a. Wählen Sie unter Volumes die Option Volume hinzufügen aus.
   b. Geben Sie für jedes Volumen, 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, vom Container aus auf den Quellordner zuzugreifen. Wählen Sie aus den folgenden Optionen aus:
  - Schreibgeschützt — Die Lambda-Funktion hat nur Lesezugriff auf den Quellordner. Dies ist die Standardauswahl.
  - Lesen/Schreiben — Die Lambda-Funktion hat Lese-/Schreibzugriff auf den Quellordner.
- Gruppenbesitzer hinzufügen — (Optional) Ob die Systemgruppe, die die Lambda-Funktionskomponente ausführt, als Besitzer des Quellordners hinzugefügt werden soll oder nicht. Die Standardeinstellung ist False.

c. Um ein Volume zu entfernen, wählen Sie neben dem Volume, das Sie entfernen möchten, die Option Entfernen aus.

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:

   - Bereitstellungspfad — Der Pfad zum Systemgerät auf dem Kerngerät.
   - Berechtigung — (Optional) Die Berechtigung, vom Container aus auf das Systemgerät zuzugreifen. Wählen Sie aus den folgenden Optionen aus:
     - Schreibgeschützt — Die Lambda-Funktion hat nur Lesezugriff auf das Systemgerät. Dies ist die Standardauswahl.
     - Lesen/Schreiben — Die Lambda-Funktion hat Lese-/Schreibzugriff auf den Quellordner.
   - Gruppenbesitzer hinzufügen — (Optional) Ob die Systemgruppe, die die Lambda-Funktionskomponente ausführt, als Besitzer des Systemgeräts hinzugefügt werden soll oder nicht. Die Standardeinstellung ist False.

Schritt 6: Erstellen Sie die Lambda-Funktionskomponente

Nachdem Sie die Einstellungen für Ihre Lambda-Funktionskomponente konfiguriert haben, wählen Sie Create, um die Erstellung der neuen Komponente abzuschließen.

Um die Lambda-Funktion auf Ihrem Kerngerät auszuführen, können Sie die neue Komponente dann auf Ihren Kerngeräten bereitstellen. Weitere Informationen finden Sie unter AWS IoT Greengrass Komponenten auf Geräten bereitstellen.

Eine Lambda-Funktion als Komponente importieren ()AWS CLI

Verwenden Sie die CreateComponentVersionOperation, um Komponenten aus Lambda-Funktionen zu erstellen. Wenn Sie diesen Vorgang aufrufen, geben Sie `lambdaFunction` an, dass eine Lambda-Funktion importiert werden soll.

Aufgaben

- Schritt 1: Definieren Sie die Lambda-Funktionskonfiguration
- Schritt 2: Erstellen Sie die Lambda-Funktionskomponente

Schritt 1: Definieren Sie die 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 das `lambdaArn` durch den ARN der Lambda-Funktion, die importiert werden soll.

   ```
   {
     "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 wird die Komponente mit dem Namen der Lambda-Funktion erstellt.

   ```
   {
     "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 wird die Komponente mit der Version der Lambda-Funktion als gültige semantische Version erstellt. 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 Hauptteil. unbedeutend. Patch-Nummernsystem. Version `1.0.0` steht beispielsweise für die erste Hauptversion einer Komponente. 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 Kerngeräte haben Attribute für Betriebssystem (`os`) und Architektur (`architecture`). Die AWS IoT Greengrass Core-Software kann weitere Plattformattribute hinzufügen. Sie können auch benutzerdefinierte Plattformattribute angeben, wenn Sie die Greengrass Nucleus-Komponente auf einem Core-Gerät bereitstellen. Gehen Sie wie folgt vor:

   a. Fügen Sie der Lambda-Funktion in `lambda-function-component.json` eine Liste von Plattformen (`componentPlatforms`) hinzu.

      ```
      {
      ```

```
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [

    ]
  }
}
```

b. Fügen Sie jede unterstützte Plattform zur Liste hinzu. Für jede Plattform gibt es eine `name` Benutzerschnittstelle zur Identifizierung 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"
  }
}
```

`lambda-function-component.json`Möglicherweise enthalten Sie ein Dokument, das dem folgenden Beispiel ähnelt.

```
{
  "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, umfasst die Bereitstellung diese Abhängigkeiten, damit Ihre Funktion ausgeführt werden kann.

   > **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. Geheimnisse, lokale Schatten und Stream-Manager. Definieren Sie diese Komponenten als feste Abhängigkeiten, sodass Ihre Lambda-Funktionskomponente neu gestartet wird, wenn sich der Status der Abhängigkeit ändert. Weitere Informationen finden Sie unter V1-Lambda-Funktionen importieren.

   Gehen Sie wie folgt vor:

   a. Fügen Sie der Lambda-Funktion in `lambda-function-component.json` eine Zuordnung von Komponentenabhängigkeiten (`componentDependencies`) 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": {

    }
  }
}
```

b. Fügen Sie der Map jede Komponentenabhängigkeit hinzu. Geben Sie den Komponentennamen als Schlüssel an und geben Sie ein Objekt mit den folgenden Parametern an:

- `versionRequirement`— Die semantische Versionsbeschränkung im NPM-Stil, die die kompatiblen Versionen der Komponentenabhängigkeit identifiziert. Sie können eine einzelne Version oder einen Versionsbereich angeben. Weitere Informationen zu semantischen Versionseinschränkungen finden Sie im npm Semver Calculator.
- `dependencyType`— (Optional) Der Typ der Abhängigkeit. Wählen Sie eine der folgenden Optionen aus:
  - `SOFT`— Die Lambda-Funktionskomponente wird nicht neu gestartet, wenn sich der Status der Abhängigkeit ändert.
  - `HARD`— Die Lambda-Funktionskomponente wird neu gestartet, wenn sich der Status der Abhängigkeit ändert.

  Der Standardwert ist `HARD`.

Das folgende Beispiel gibt an, dass diese Lambda-Funktion von einer beliebigen 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"
  }
}
```

`lambda-function-component.json`Möglicherweise enthalten Sie ein Dokument, das dem folgenden Beispiel ähnelt.

```
{
  "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 zur Ausführung 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-Parameter-Objekt (`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, die die Lambda-Funktion für geschäftliche Nachrichten abonniert. Sie können Ereignisquellen angeben, um diese Funktion für lokale Veröffentlichungs-/Abonnementnachrichten und MQTT-Nachrichten zu abonnieren. AWS IoT Core Die Lambda-Funktion wird aufgerufen, wenn sie eine Nachricht von einer Ereignisquelle empfängt.

> **Note**
>
> Um diese Funktion für Nachrichten von anderen Lambda-Funktionen oder -Komponenten zu abonnieren, stellen Sie die ältere Abonnement-Router-Komponente bereit, wenn Sie diese Lambda-Funktionskomponente bereitstellen. Wenn Sie die ältere Abonnement-Router-Komponente 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 jede Ereignisquelle zur Liste hinzu. Jede Ereignisquelle hat die folgenden Parameter:

- `topic`— Das Thema, zu dem Nachrichten abonniert werden sollen.
- `type`— Der Typ der Ereignisquelle. Wählen Sie aus den folgenden Optionen aus:
  - `PUB_SUB` — Abonnieren Sie lokale Veröffentlichen/Abonnement-Nachrichten.

    Wenn Sie Greengrass Nucleus v2.6.0 oder höher und Lambda Manager v2.2.5 oder höher verwenden, können Sie bei der Angabe dieses Typs Platzhalter (+und#) für MQTT-Themen verwenden. `topic`
  - `IOT_CORE` AWS IoT Core — Abonnieren Sie MQTT-Nachrichten.

    Sie können Platzhalter (+und#) für MQTT-Themen verwenden, `topic` wenn Sie diesen Typ angeben.

  Das folgende Beispiel abonniert AWS IoT Core MQTT für Themen, die dem Themenfilter entsprechen. `hello/world/+`

  ```
  {
    "topic": "hello/world/+",
    "type": "IOT_CORE"
  }
  ```

  Ihr `lambda-function-component.json` Beispiel 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 Lambda-Funktionsparameterobjekt an:

- `environmentVariables`— Die Karte der Umgebungsvariablen, die für die Lambda-Funktion verfügbar sind, wenn sie ausgeführt wird.
- `execArgs`— Die Liste der Argumente, die an die Lambda-Funktion übergeben werden sollen, wenn sie ausgeführt wird.
- `inputPayloadEncodingType`— Die Art der Nutzlast, die die Lambda-Funktion unterstützt. Wählen Sie aus den folgenden Optionen aus:

  - `json`
  - `binary`

  Standard: `json`

- `pinned`— Ob die Lambda-Funktion gepinnt ist oder nicht. Der Standardwert ist `true`.
  - Eine fixierte (oder langlebige) Lambda-Funktion AWS IoT Greengrass startet beim Start und läuft in ihrem eigenen Container weiter.
  - Eine Lambda-Funktion ohne Fixierung (oder On-Demand-Funktion) startet nur, wenn sie ein Arbeitselement empfängt, und wird beendet, wenn sie für eine festgelegte maximale Leerlaufzeit inaktiv bleibt. Wenn die Funktion mehrere Arbeitselemente hat, erstellt die AWS IoT Greengrass Core-Software mehrere Instanzen der Funktion.

  Verwenden Sie `maxIdleTimeInSeconds` diese Option, um die maximale Leerlaufzeit für Ihre Funktion festzulegen.
- `timeoutInSeconds`— Die maximale Zeit in Sekunden, die die Lambda-Funktion ausführen kann, bevor das Timeout eintritt. 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 fixierte Funktionen. Standardmäßig ist ein Zeitraum von 60 Sekunden festgelegt.
- `maxIdleTimeInSeconds`— Die maximale Zeit in Sekunden, während der eine Lambda-Funktion ohne Pin inaktiv sein kann, bevor die AWS IoT Greengrass Core-Software ihren Prozess stoppt. Standardmäßig ist ein Zeitraum von 60 Sekunden festgelegt.
- `maxInstancesCount`— Die maximale Anzahl von Instanzen, die eine nicht gepinnte Lambda-Funktion gleichzeitig ausführen kann. Die Standardeinstellung ist 100 Instanzen.
- `maxQueueSize`— Die maximale Größe der Nachrichtenwarteschlange für die Lambda-Funktionskomponente. Die AWS IoT Greengrass Core-Software speichert Nachrichten in einer FIFO (first-in-first-out) -Warteschlange, bis sie die Lambda-Funktion ausführen kann, um jede Nachricht zu verarbeiten. Die Standardeinstellung ist 1.000 Nachrichten.

`lambda-function-component.json`Möglicherweise enthält Ihr Dokument ein Dokument, das dem folgenden Beispiel ähnelt.

```
{
  "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 sich auch dafür entscheiden, die Lambda-Funktion als Prozess ohne jegliche Isolierung auszuführen. Wenn Sie die Lambda-Funktion in einem Container ausführen, konfigurieren Sie die Speichergröße des Containers und welche Systemressourcen für die Lambda-Funktion verfügbar sind. Gehen Sie wie folgt vor:

i. Fügen Sie das Linux-Prozessparameter-Objekt (`linuxProcessParams`) zum Lambda-Parameter-Objekt in `lambda-function-component.json` 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": {

      }
    }
  }
}
```

ii. (Optional) Geben Sie an, ob die Lambda-Funktion in einem Container ausgeführt wird. Fügen Sie den `isolationMode` Parameter dem Prozessparameter-Objekt hinzu und wählen Sie aus den folgenden Optionen:

- `GreengrassContainer`— Die Lambda-Funktion läuft in einem Container.
- `NoContainer`— Die Lambda-Funktion läuft als Prozess ohne jegliche Isolierung.

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, konfigurieren, die dem Container zur Verfügung gestellt werden sollen. Gehen Sie wie folgt vor:

A. Fügen Sie das Container-Parameter-Objekt (`containerParams`) zum Linux-Prozessparameter-Objekt in `lambda-function-component.json` 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": {

        }
      }
    }
  }
}
```

B. (Optional) Fügen Sie den `memorySizeInKB` Parameter hinzu, um die Speichergröße des Containers anzugeben. Die Standardeinstellung 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. 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 am Ziel innerhalb des Containers. Gehen Sie wie folgt vor:

I. Fügen Sie die Liste der Volumes (`volumes`) zu den Container-Parametern 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 Volumen hat die folgenden Parameter:

- `sourcePath`— Der Pfad zum Quellordner auf dem Core-Gerät.
- `destinationPath`— Der Pfad zum Zielordner im Container.
- `permission`— (Optional) Die Berechtigung, vom Container aus auf den Quellordner zuzugreifen. Wählen Sie aus den folgenden Optionen aus:
  - `ro`— Die Lambda-Funktion hat nur Lesezugriff auf den Quellordner.
  - `rw`— Die Lambda-Funktion hat Lese- und Schreibzugriff auf den Quellordner.

  Der Standardwert ist `ro`.
- `addGroupOwner`— (Optional) Ob die Systemgruppe, die die Lambda-Funktionskomponente ausführt, als Besitzer des Quellordners hinzugefügt werden soll oder nicht. Der Standardwert ist `false`.

`lambda-function-component.json`Möglicherweise enthalten Sie ein Dokument, das dem folgenden Beispiel ähnelt.

```
{
  "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 Container-Parametern 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 zur Liste hinzu. Jedes Systemgerät hat die folgenden Parameter:

- `path`— Der Pfad zum Systemgerät auf dem Kerngerät.
- `permission`— (Optional) Die Erlaubnis, vom Container aus auf das Systemgerät zuzugreifen. Wählen Sie aus den folgenden Optionen aus:
  - `ro`— Die Lambda-Funktion hat nur Lesezugriff auf das Systemgerät.
  - `rw`— Die Lambda-Funktion hat Lese- und Schreibzugriff auf das Systemgerät.

  Der Standardwert ist `ro`.
- `addGroupOwner`— (Optional) Ob die Systemgruppe, die die Lambda-Funktionskomponente ausführt, als Besitzer des Systemgeräts hinzugefügt werden soll oder nicht. Der Standardwert ist `false`.

`lambda-function-component.json`Möglicherweise enthalten Sie ein Dokument, das dem folgenden Beispiel ähnelt.

```
{
  "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 Kennzeichnen Sie Ihre AWS IoT Greengrass Version 2 Ressourcen.

Schritt 2: Erstellen Sie die Lambda-Funktionskomponente

1. Führen Sie den folgenden Befehl aus, um die Lambda-Funktionskomponente von `lambda-function-component.json` zu erstellen.

   ```
   aws greengrassv2 create-component-version --cli-input-json file://lambda-function-component.json
   ```

   Die Antwort ähnelt dem folgenden Beispiel, wenn die Anfrage 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 das `arn` aus der Ausgabe, um im nächsten Schritt den Status der Komponente zu überprüfen.

2. 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:account-
id:components:com.example.HelloWorldLambda:versions:1.0.0"
```

Wenn die Komponente validiert wird, gibt die Antwort an, dass der Status der Komponente lautetDEPLOYABLE.

```
{
  "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 fertig istDEPLOYABLE, können Sie die Lambda-Funktion auf Ihren Kerngeräten bereitstellen. Weitere Informationen finden Sie unter AWS IoT Greengrass Komponenten auf Geräten bereitstellen.

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 Vorgängen:

- 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

- IPC-Client-Versionen
- Wird SDKs für die Kommunikation zwischen Prozessen unterstützt
- Connect zum AWS IoT Greengrass Core IPC-Dienst her
- Autorisieren Sie Komponenten zur Ausführung von IPC-Vorgängen
- Abonnieren Sie IPC-Event-Streams
- Bewährte IPC-Praktiken
- Lokale Nachrichten veröffentlichen/abonnieren
- MQTT-Nachrichten veröffentlichen/abonnieren AWS IoT Core

- Interagieren Sie mit dem Lebenszyklus von Komponenten
- Interagieren Sie mit der Komponentenkonfiguration
- Geheime Werte abrufen
- Interagiere mit lokalen Schatten
- Lokale Bereitstellungen und Komponenten verwalten
- Client-Geräte authentifizieren und autorisieren

IPC-Client-Versionen

In späteren Versionen von 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, den IPC-Client V2 zu 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 Sie eine Greengrass-Komponente, die Komponenten-Updates verzögert
- Tutorial: Interagieren Sie mit lokalen IoT-Geräten über MQTT

Derzeit unterstützt der AWS IoT Device SDK für C++ v2 nur den IPC-Client V1.

Wird SDKs für die Kommunikation zwischen Prozessen unterstützt

Die AWS IoT Greengrass Core-IPC-Bibliotheken sind in den folgenden AWS IoT Device SDK Versionen enthalten.

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 später
- 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, mit dem Sie bauen. CMake

```
cmake_minimum_required(VERSION 3.1)
project (greengrassv2_pubsub_subscriber)

file(GLOB MAIN_SRC
        "*.h"
        "*.cpp"
        )
add_executable(${PROJECT_NAME} ${MAIN_SRC})

set_target_properties(${PROJECT_NAME} PROPERTIES
        LINKER_LANGUAGE CXX
        CXX_STANDARD 11)
find_package(aws-crt-cpp PATHS ~/sdk-cpp-workspace/build)
find_package(EventstreamRpc-cpp PATHS ~/sdk-cpp-workspace/build)
find_package(GreengrassIpc-cpp PATHS ~/sdk-cpp-workspace/build)
target_link_libraries(${PROJECT_NAME} AWS::GreengrassIpc-cpp)
```

4. 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://amzn-s3-demo-bucket/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pubsub_subscriber",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

YAML

```
...
Manifests:
  - Lifecycle:
      Run: {artifacts:path}/greengrassv2_pubsub_subscriber
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pubsub_subscriber
        Permission:
          Execute: OWNER
```

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 den AWS IoT Device SDK for 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 eine eindeutige Richtlinie zu erstellen IDs, 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 definieren, die wie folgt lauten: IDs
>
> - `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

```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 eine Ressource zu finden, 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 Thema

Wörtliches Zeichen	Escape-Sequenz	Hinweise
		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
- Beispiel für Abonnement-Handler

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 von Java und Python verfügbar ist, können Sie diese Funktionen definieren SDKs, ohne eine Abonnement-Handler-Klasse zu erstellen.

Java

Wenn Sie den IPC-Client V1 verwenden, müssen Sie die generische `software.amazon.awssdk.eventstreamrpc.StreamResponseHandler<`*`StreamEventType`*`>` Schnittstelle implementieren. *`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
- Autorisierung
- PublishToTopic
- SubscribeToTopic
- Beispiele

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 Literalze ichen. 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://amzn-s3-demo-bucket/artifacts/
com.example.PubSubPublisherCpp/1.0.0/greengrassv2_pubsub_publisher",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubPublisherCpp
ComponentVersion: 1.0.0
ComponentDescription: A component that publishes messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        com.example.PubSubPublisherCpp:pubsub:1:
          policyDescription: Allows access to publish to all topics.
          operations:
            - aws.greengrass#PublishToTopic
          resources:
            - "*"
Manifests:
  - Lifecycle:
      Run: "{artifacts:path}/greengrassv2_pubsub_publisher"
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.PubSubPublisherCpp/1.0.0/greengrassv2_pubsub_publisher
        Permission:
          Execute: OWNER
```

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://amzn-s3-demo-bucket/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pub_sub_subscriber",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
```

```
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubSubscriberCpp
ComponentVersion: 1.0.0
ComponentDescription: A component that subscribes to messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        com.example.PubSubSubscriberCpp:pubsub:1:
          policyDescription: Allows access to subscribe to all topics.
          operations:
            - aws.greengrass#SubscribeToTopic
          resources:
            - "*"
Manifests:
  - Lifecycle:
      Run: "{artifacts:path}/greengrassv2_pub_sub_subscriber"
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pub_sub_subscriber
        Permission:
          Execute: OWNER
```

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
- Autorisierung
- PublishToIoTCore
- SubscribeToIoTCore
- Beispiele

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 erhalten 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 Datenströme auf Greengrass-Kerngeräten verwalten.

Anforderung

Die Anforderung dieser Operation 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 `true` beim Veröffentlichen auf gesetzt werden soll.

`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 AWS IoT Core über ein Thema oder einen Themenfilter. Die AWS IoT Greengrass Core-Software entfernt Abonnements, wenn die Komponente das Ende ihres Lebenszyklus erreicht.

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 dieser Operation 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 `true` beim Veröffentlichen auf gesetzt werden soll.

`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://amzn-s3-demo-bucket/artifacts/
com.example.IoTCorePublisherCpp/1.0.0/greengrassv2_iotcore_publisher",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.IoTCorePublisherCpp
ComponentVersion: 1.0.0
ComponentDescription: A component that publishes MQTT messages to IoT Core.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.mqttproxy:
        com.example.IoTCorePublisherCpp:mqttproxy:1:
          policyDescription: Allows access to publish to all topics.
          operations:
```

```
            - aws.greengrass#PublishToIoTCore
          resources:
            - "*"
Manifests:
  - Lifecycle:
      Run: "{artifacts:path}/greengrassv2_iotcore_publisher"
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.IoTCorePublisherCpp/1.0.0/greengrassv2_iotcore_publisher
        Permission:
          Execute: OWNER
```

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://amzn-s3-demo-bucket/artifacts/
com.example.IoTCoreSubscriberCpp/1.0.0/greengrassv2_iotcore_subscriber",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.IoTCoreSubscriberCpp
ComponentVersion: 1.0.0
ComponentDescription: A component that subscribes to MQTT messages from IoT Core.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.mqttproxy:
        com.example.IoTCoreSubscriberCpp:mqttproxy:1:
          policyDescription: Allows access to subscribe to all topics.
          operations:
            - aws.greengrass#SubscribeToIoTCore
          resources:
            - "*"
Manifests:
  - Lifecycle:
      Run: "{artifacts:path}/greengrassv2_iotcore_subscriber"
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.IoTCoreSubscriberCpp/1.0.0/greengrassv2_iotcore_subscriber
        Permission:
```

```
            Execute: OWNER
```

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 Sie mit dem Lebenszyklus von Komponenten

Verwenden Sie den IPC-Service für den Komponentenlebenszyklus, um:

- Aktualisieren Sie den Komponentenstatus auf dem Kerngerät.

- Abonnieren Sie Updates zum Komponentenstatus.
- Verhindern Sie, dass der Nucleus die Komponente während einer Bereitstellung unterbricht, um ein Update anzuwenden.
- Unterbrechen Sie die Komponentenprozesse und setzen Sie sie fort.

Themen

- Minimale SDK-Versionen
- Autorisierung
- UpdateState
- SubscribeToComponentUpdates
- DeferComponentUpdate
- PauseComponent
- ResumeComponent

Minimale SDK-Versionen

In der folgenden Tabelle sind die Mindestversionen von aufgeführt AWS 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 v2 JavaScript	v1.12.0	

Autorisierung

Um andere Komponenten einer benutzerdefinierten Komponente anzuhalten oder wieder aufzunehmen, müssen Sie Autorisierungsrichtlinien definieren, die es Ihrer Komponente ermöglichen, andere Komponenten zu verwalten. Informationen zur Definition von Autorisierungsrichtlinien finden Sie unterAutorisieren Sie Komponenten zur Ausführung von IPC-Vorgängen.

Autorisierungsrichtlinien für das Lebenszyklusmanagement von Komponenten haben die folgenden Eigenschaften.

IPC-Dienst-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 wieder aufzunehmen.	Ein Komponentenname oder * um den Zugriff auf alle Komponenten zu ermöglichen.
*	Ermöglicht einer Komponente, die von Ihnen angegebenen Komponenten anzuhalten und wieder aufzunehmen.	Ein Komponentenname oder * um den Zugriff auf alle Komponenten zu ermöglichen.

Beispiele für Autorisierungsrichtlinien

Anhand des folgenden Beispiels für eine Autorisierungsrichtlinie können Sie Autorisierungsrichtlinien für Ihre Komponenten konfigurieren.

Example Beispiel für eine Autorisierungsrichtlinie

Das folgende Beispiel für eine Autorisierungsrichtlinie ermöglicht es einer Komponente, alle Komponenten anzuhalten und wieder aufzunehmen.

```
{
```

```
  "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 Kerngerät.

Anforderung

Die Anfrage dieses Vorgangs hat die folgenden Parameter:

`state`

Der festzulegende Status. Diese Aufzählung,`LifecycleState`, hat die folgenden Werte:

- `RUNNING`
- `ERRORED`

Antwort

Dieser Vorgang liefert in seiner Antwort keine Informationen.

SubscribeToComponentUpdates

Abonnieren Sie, um Benachrichtigungen zu erhalten, bevor die AWS IoT Greengrass Core-Software eine Komponente aktualisiert. Die Benachrichtigung gibt an, ob der Nucleus im Rahmen des Updates neu gestartet wird oder nicht.

Der Nucleus sendet nur dann Aktualisierungsbenachrichtigungen, wenn die Komponenten-Update-Richtlinie der Bereitstellung vorsieht, dass Komponenten benachrichtigt werden sollen. Das Standardverhalten besteht darin, Komponenten zu benachrichtigen. Weitere Informationen finden Sie unter Erstellen von Bereitstellungen und unter dem DeploymentComponentUpdatePolicyObjekt, das Sie angeben können, wenn Sie den CreateDeploymentVorgang aufrufen.

> **Important**
>
> Bei lokalen Bereitstellungen werden Komponenten vor Updates nicht benachrichtigt.

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: `ComponentUpdatePolicyEvents`

> **Tip**
>
> In einem Tutorial erfahren Sie, wie Sie eine Komponente entwickeln, die Komponenten-Updates bedingt verzögert. Weitere Informationen finden Sie unter Tutorial: Entwickeln Sie eine Greengrass-Komponente, die Komponenten-Updates verzögert.

Anforderung

Die Anfrage dieses Vorgangs hat keine Parameter.

Antwort

Die Antwort dieser Operation enthält die folgenden Informationen:

`messages`

Der Stream von Benachrichtigungsnachrichten. Dieses Objekt,`ComponentUpdatePolicyEvents`, enthält die folgenden Informationen:

`preUpdateEvent`(Python:`pre_update_event`)

(Optional) Ein Ereignis, das anzeigt, dass der Nucleus eine Komponente aktualisieren möchte. Sie können darauf antworten, indem DeferComponentUpdate Sie die Aktualisierung bestätigen oder verschieben, bis Ihre Komponente für den 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`)

Ob der Nucleus neu gestartet werden muss, um das Update anzuwenden.

`postUpdateEvent`(Python:`post_update_event`)

(Optional) Ein Ereignis, das darauf hinweist, dass der Nucleus 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**
>
> Für diese Funktion ist Version 2.7.0 oder höher der Greengrass Nucleus-Komponente erforderlich.

DeferComponentUpdate

Bestätigen oder verschieben Sie ein Komponenten-Update, das Sie mit entdecken. SubscribeToComponentUpdates Sie geben an, wie lange es dauern soll, bis der Nucleus erneut überprüft, ob Ihre Komponente bereit ist, mit dem Komponenten-Update fortzufahren. Sie können diesen Vorgang auch verwenden, um dem Nucleus mitzuteilen, dass Ihre Komponente für das Update bereit ist.

Wenn eine Komponente nicht auf die Benachrichtigung über das Komponenten-Update reagiert, wartet der Nucleus so lange, wie Sie in der Komponenten-Update-Richtlinie für die Installation angegeben haben. Nach diesem Timeout fährt der Nucleus mit der Bereitstellung fort. Das Standard-Timeout für Komponenten-Updates beträgt 60 Sekunden. Weitere Informationen finden Sie unter

Erstellen von Bereitstellungen und unter dem DeploymentComponentUpdatePolicyObjekt, das Sie angeben können, wenn Sie den CreateDeploymentVorgang aufrufen.

> **Tip**
>
> Sie können einem Tutorial folgen, um zu erfahren, wie Sie eine Komponente entwickeln, die Komponenten-Updates bedingt verzögert. Weitere Informationen finden Sie unter Tutorial: Entwickeln Sie eine Greengrass-Komponente, die Komponenten-Updates verzögert.

Anforderung

Die Anforderung dieses Vorgangs hat die folgenden Parameter:

`deploymentId`(Python:`deployment_id`)

Die ID der AWS IoT Greengrass Bereitstellung, die verschoben werden soll.

`message`

(Optional) Der Name der Komponente, für die Updates verschoben werden sollen.

Standardmäßig wird der Name der Komponente verwendet, die die Anfrage stellt.

`recheckAfterMs`(Python:`recheck_after_ms`)

Die Zeitspanne in Millisekunden, um die Aktualisierung verschoben werden soll. Der Nukleus wartet auf diese Zeitspanne und sendet dann eine weitere`PreComponentUpdateEvent`, mit der Sie etwas entdecken können. SubscribeToComponentUpdates

Geben Sie `0` an, ob Sie das Update bestätigen möchten. Dadurch wird dem Nucleus mitgeteilt, dass Ihre Komponente für das Update bereit ist.

Der Standardwert ist null Millisekunden, was bedeutet, dass das Update bestätigt wird.

Antwort

Dieser Vorgang liefert in seiner Antwort keine Informationen.

PauseComponent

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.

Unterbricht die Prozesse einer Komponente auf dem Kerngerät. Verwenden Sie den ResumeComponentVorgang, um eine Komponente wieder aufzunehmen.

Sie können nur generische Komponenten anhalten. Wenn Sie versuchen, einen anderen Komponententyp anzuhalten, löst dieser Vorgang eine `InvalidRequestError` aus.

> **Note**
>
> Durch diesen Vorgang können containerisierte Prozesse wie Docker-Container nicht angehalten werden. Um einen Docker-Container anzuhalten und wieder aufzunehmen, können Sie die Befehle docker pause und docker unpause verwenden.

Dieser Vorgang unterbricht keine Komponentenabhängigkeiten oder Komponenten, die von der Komponente abhängen, die Sie anhalten. Beachten Sie dieses Verhalten, wenn Sie eine Komponente anhalten, die von einer anderen Komponente abhängig ist, da bei der abhängigen Komponente möglicherweise Probleme auftreten, wenn ihre Abhängigkeit unterbrochen wird.

Wenn Sie eine angehaltene Komponente neu starten oder herunterfahren, z. B. durch eine Bereitstellung, nimmt der Greengrass-Nucleus die Komponente wieder auf und führt ihren Shutdown-Lebenszyklus durch. Weitere Informationen zum Neustarten einer Komponente finden Sie unter. RestartComponent

> **Important**
>
> Um diesen Vorgang verwenden zu können, müssen Sie eine Autorisierungsrichtlinie definieren, die die Erlaubnis zur Verwendung dieses Vorgangs erteilt. Weitere Informationen finden Sie unter Autorisierung.

Minimale SDK-Versionen

In der folgenden Tabelle sind die Mindestversionen von aufgeführt AWS IoT Device SDK , die Sie zum Anhalten und Wiederaufnehmen 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 Anfrage 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 in seiner Antwort keine Informationen.

ResumeComponent

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.

Setzt die Prozesse einer Komponente auf dem Kerngerät fort. Verwenden Sie den PauseComponentVorgang, um eine Komponente anzuhalten.

Sie können nur angehaltene Komponenten wieder aufnehmen. Wenn Sie versuchen, eine Komponente wieder aufzunehmen, 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.

Minimale SDK-Versionen

In der folgenden Tabelle sind die Mindestversionen von aufgeführt AWS IoT Device SDK , die Sie zum Anhalten und Wiederaufnehmen 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 Anfrage dieser Operation hat die folgenden Parameter:

`componentName`(Python:`component_name`)

Der Name der Komponente, die fortgesetzt werden soll.

Antwort

Dieser Vorgang liefert in seiner Antwort keine Informationen.

Interagieren Sie mit der Komponentenkonfiguration

Mit dem IPC-Dienst für die Komponentenkonfiguration können Sie Folgendes tun:

- Abrufen und Einstellen der Konfigurationsparameter für Komponenten.
- Abonnieren Sie Updates zur Komponentenkonfiguration.
- Validieren Sie Aktualisierungen der Komponentenkonfiguration, bevor der Nucleus sie einführt.

Themen

- Minimale SDK-Versionen
- GetConfiguration
- UpdateConfiguration
- SubscribeToConfigurationUpdate
- SubscribeToValidateConfigurationUpdates
- SendConfigurationValidityReport

Minimale SDK-Versionen

In der folgenden Tabelle sind die Mindestversionen von aufgeführt AWS 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 v2 JavaScript	v1.12.0	

GetConfiguration

Ruft einen Konfigurationswert für eine Komponente auf dem Kerngerät ab. Sie geben den Schlüsselpfad an, für den ein Konfigurationswert abgerufen werden soll.

Anforderung

Die Anforderung dieses Vorgangs hat die folgenden Parameter:

`componentName`(Python:`component_name`)

(Optional) Der Name der Komponente.

Standardmäßig wird der Name der Komponente verwendet, 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 `["mqtt", "port"]` an, dass der Wert von `port` in der folgenden Konfiguration abgerufen werden soll.

```
{
  "mqtt": {
    "port": 443
  }
}
```

Um die vollständige Konfiguration der Komponente abzurufen, geben Sie eine leere Liste an.

Antwort

Die Antwort dieses Vorgangs 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 Kerngerät.

Anforderung

Die Anforderung dieses Vorgangs hat die folgenden Parameter:

`keyPath`(Python:`key_path`)

(Optional) Der Schlüsselpfad zum Containerknoten (dem Objekt), der aktualisiert werden soll. 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 `{ "port": 443 }` an, für den der Wert `port` in der folgenden Konfiguration festgelegt werden soll.

```
{
  "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, erstellt dieser Vorgang ihn und setzt seinen Wert auf das Objekt in`valueToMerge`.

Standardmäßig wird das Stammverzeichnis des Konfigurationsobjekts verwendet.

`timestamp`

Die aktuelle Zeit der Unix-Epoche 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 der von Ihnen angegebenen Position zusammengeführt werden sollkeyPath. Weitere Informationen finden Sie unter Komponentenkonfigurationen aktualisieren.

Antwort

Dieser Vorgang liefert in seiner Antwort keine Informationen.

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 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: `ConfigurationUpdateEvents`

Anforderung

Die Anforderung dieses Vorgangs hat die folgenden Parameter:

`componentName`(Python:`component_name`)

(Optional) Der Name der Komponente.

Standardmäßig wird der Name der Komponente verwendet, die die Anforderung stellt.

`keyPath`(Python:`key_path`)

Der Schlüsselpfad zu dem 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 `["mqtt", "port"]` an, dass der Wert von `port` in der folgenden Konfiguration abgerufen werden soll.

```
{
  "mqtt": {
    "port": 443
  }
}
```

Um Updates für alle Werte in der Konfiguration der Komponente zu abonnieren, geben Sie eine leere Liste an.

Antwort

Die Antwort dieses Vorgangs enthält die folgenden Informationen:

`messages`

Der Stream von Benachrichtigungsnachrichten. Dieses Objekt,`ConfigurationUpdateEvents`, enthält die folgenden Informationen:

`configurationUpdateEvent`(Python:`configuration_update_event`)

Das Ereignis zur Aktualisierung der Konfiguration. Dieses Objekt,`ConfigurationUpdateEvent`, enthält die folgenden Informationen:

`componentName`(Python:`component_name`)

Der Name der Komponente.

`keyPath`(Python:`key_path`)

Der Schlüsselpfad zu dem Konfigurationswert, der aktualisiert wurde.

SubscribeToValidateConfigurationUpdates

Abonnieren Sie, um Benachrichtigungen vor den Konfigurationsupdates dieser Komponente zu erhalten. Auf diese Weise können Komponenten Aktualisierungen ihrer eigenen Konfiguration validieren. Verwenden Sie die SendConfigurationValidityReport Operation, um dem Nucleus mitzuteilen, ob die Konfiguration gültig ist oder nicht.

> **Important**
>
> Lokale Bereitstellungen benachrichtigen die Komponenten nicht über Updates.

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: `ValidateConfigurationUpdateEvents`

Anforderung

Die Anfrage dieses Vorgangs hat keine Parameter.

Antwort

Die Antwort dieser Operation enthält die folgenden Informationen:

`messages`

Der Stream von Benachrichtigungsnachrichten. Dieses Objekt,`ValidateConfigurationUpdateEvents`, enthält die folgenden Informationen:

`validateConfigurationUpdateEvent`(Python:`validate_configuration_update_event`)

Das Ereignis zur Aktualisierung der Konfiguration. 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 Nucleus mit, ob ein Konfigurationsupdate für diese Komponente gültig ist oder nicht. Die Bereitstellung schlägt fehl, wenn Sie dem Nucleus mitteilen, dass die neue Konfiguration nicht gültig ist. Verwenden Sie den SubscribeToValidateConfigurationUpdates Vorgang, um die Konfigurationsupdates zu abonnieren und zu validieren.

Wenn eine Komponente nicht auf eine Benachrichtigung zur Überprüfung des Konfigurationsupdates reagiert, wartet der Nucleus so lange, wie Sie in der Konfigurationsvalidierungsrichtlinie für die Bereitstellung angegeben haben. Nach diesem Timeout fährt der Nucleus mit der Bereitstellung fort. Das Standard-Timeout für die Komponentenvalidierung beträgt 20 Sekunden. Weitere Informationen finden Sie unter Erstellen von Bereitstellungen und unter dem DeploymentConfigurationValidationPolicyObjekt, das Sie angeben können, wenn Sie den CreateDeploymentVorgang aufrufen.

Anforderung

Die Anforderung dieses Vorgangs hat die folgenden Parameter:

`configurationValidityReport`(Python:`configuration_validity_report`)

Der Bericht, der dem Nucleus mitteilt, ob das Konfigurationsupdate 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 Nucleus kann sie auf diese Komponente anwenden.
- `REJECTED`— Die Konfiguration ist nicht gültig 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 angibt, warum die Konfiguration nicht gültig ist.

Antwort

Dieser Vorgang liefert in seiner Antwort keine Informationen.

Geheime Werte abrufen

Verwenden Sie den Secret Manager IPC-Dienst, um geheime Werte aus Geheimnissen auf dem Kerngerät abzurufen. Sie verwenden die Secret Manager-Komponente, um verschlüsselte Geheimnisse auf Kerngeräten bereitzustellen. Anschließend können Sie eine IPC-Operation verwenden, um das Geheimnis zu entschlüsseln und seinen Wert in Ihren benutzerdefinierten Komponenten zu verwenden.

Themen

- Minimale SDK-Versionen
- Autorisierung
- GetSecretValue

- Beispiele

Minimale SDK-Versionen

In der folgenden Tabelle sind die Mindestversionen von aufgeführt AWS IoT Device SDK , die Sie verwenden müssen, um geheime Werte aus Geheimnissen auf dem Kerngerät 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 v2 JavaScript	v1.12.0	

Autorisierung

Um den Secret Manager in einer benutzerdefinierten Komponente zu verwenden, müssen Sie Autorisierungsrichtlinien definieren, die es Ihrer Komponente ermöglichen, den Wert von Geheimnissen abzurufen, die Sie auf dem Kerngerät speichern. Informationen zur Definition von Autorisierungsrichtlinien finden Sie unterAutorisieren Sie Komponenten zur Ausführung von IPC-Vorgängen.

Autorisierungsrichtlinien für Secret Manager haben die folgenden Eigenschaften.

IPC-Dienst-ID: `aws.greengrass.SecretManager`

Operation	Beschreibung	Ressourcen
`aws.greengrass#GetSecretValue` oder *	Ermöglicht einer Komponente, den Wert von Geheimnis	Ein geheimer ARN für Secrets Manager oder * um den

Operation	Beschreibung	Ressourcen
	sen abzurufen, die auf dem Kerngerät verschlüsselt sind.	Zugriff auf alle Geheimnisse zu ermöglichen.

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, den Wert eines beliebigen Geheimnisses auf dem Kerngerät abzurufen.

> **Note**
>
> Wir empfehlen, in einer Produktionsumgebung den Geltungsbereich der Autorisierungsrichtlinie zu reduzieren, sodass die Komponente nur die Geheimnisse abruft, die sie verwendet. Sie können den * Platzhalter ARNs bei der Bereitstellung der Komponente in eine geheime Liste ändern.

```
{
  "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 Geheimnisses ab, das Sie auf dem Kerngerät speichern.

Dieser Vorgang ähnelt dem Secrets Manager Manager-Vorgang, mit dem Sie den Wert eines Secrets in der abrufen können AWS Cloud. Weitere Informationen finden Sie unter GetSecretValue in der AWS Secrets Manager -API-Referenz.

Anforderung

Die Anfrage dieses Vorgangs hat die folgenden Parameter:

`refresh`(Python:`refresh`)

(optional): Ob das angeforderte Geheimnis mit seinem neuesten Wert vom AWS Secrets Manager Dienst synchronisiert werden soll.

Wenn dieser Wert auf true gesetzt ist, fordert Secret Manager den AWS Secrets Manager Dienst nach dem neuesten Wert des angegebenen geheimen Labels an und gibt diesen Wert als Antwort zurück. Andernfalls wird der geheime Wert zurückgegeben, der lokal gespeichert wurde.

Dieser Parameter funktioniert nicht in Verbindung mit dem `versionId` Parameter in der Anfrage. Dieser Parameter funktioniert, wenn er in Verbindung mit Nucleus 2.13.0 und höher verwendet wird.

`secretId`(Python:`secret_id`)

Der Name des Geheimnisses, das abgerufen werden soll. Sie können entweder den Amazon-Ressourcennamen (ARN) oder den benutzerfreundlichen Namen des Geheimnisses angeben.

`versionId`(Python:`version_id`)

(Optional) Die ID der Version, die abgerufen werden soll.

Sie können entweder `versionId` oder `versionStage` angeben.

Wenn Sie `versionId` oder nicht angeben`versionStage`, verwendet dieser Vorgang standardmäßig die Version mit der `AWSCURRENT` Bezeichnung.

`versionStage`(Python:`version_stage`)

(Optional) Das Staging-Label der abzurufenden Version.

Sie können entweder `versionId` oder `versionStage` angeben.

Wenn Sie `versionId` oder nicht angeben`versionStage`, verwendet dieser Vorgang standardmäßig die Version mit dem `AWSCURRENT` Label.

Antwort

Die Antwort dieses Vorgangs enthält die folgenden Informationen:

`secretId`(Python:`secret_id`)

Die ID des Geheimnisses.

`versionId`(Python:`version_id`)

Die ID dieser Version des Geheimnisses.

`versionStage`(Python:`version_stage`)

Die Liste der Staging-Labels, die dieser Version des Secrets beigefügt sind.

`secretValue`(Python:`secret_value`)

Der Wert dieser Version des Geheimnisses. Dieses Objekt,`SecretValue`, enthält die folgenden Informationen.

`secretString`(Python:`secret_string`)

Der entschlüsselte Teil der geschützten geheimen Information, die Sie Secrets Manager als Zeichenfolge zur Verfügung gestellt haben.

`secretBinary`(Python:`secret_binary`)

(Optional) Der entschlüsselte Teil der geschützten geheimen Information, die Sie Secrets Manager als Binärdaten in Form eines Byte-Arrays zur Verfügung gestellt haben. Diese Eigenschaft enthält die Binärdaten als Base64-kodierte Zeichenfolge.

Diese Eigenschaft wird nicht verwendet, wenn Sie das Geheimnis in der Secrets Manager-Konsole erstellt haben.

Beispiele

Die folgenden Beispiele zeigen, wie dieser Vorgang in benutzerdefiniertem Komponentencode aufgerufen wird.

Java (IPC client V1)

Example Beispiel: Holen Sie sich einen geheimen Wert

> **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.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: Holen Sie sich einen geheimen Wert

> **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: Holen Sie sich einen geheimen Wert

```
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-Dienst in Ihren Komponenten verwenden können.

Beispiel: Geheimnis drucken (Python, IPC-Client V1)

Diese Beispielkomponente gibt den Wert eines Secrets aus, das Sie auf dem Kerngerät bereitstellen.

> **Important**
>
> Diese Beispielkomponente gibt den Wert eines Geheimnisses aus. Verwenden Sie sie daher nur mit Geheimnissen, die Testdaten speichern. Verwenden Sie diese Komponente nicht, um den Wert eines Geheimnisses zu drucken, in dem wichtige Informationen gespeichert sind.

Themen

- Rezept
- -Artefakte
- Verwendung

Rezept

Das folgende Beispielrezept definiert einen geheimen ARN-Konfigurationsparameter und ermöglicht es der Komponente, den Wert eines beliebigen Geheimnisses auf dem Kerngerät abzurufen.

> **Note**
>
> Wir empfehlen, in einer Produktionsumgebung den Geltungsbereich der Autorisierungsrichtlinie zu reduzieren, sodass die Komponente nur die Geheimnisse abruft,

die sie verwendet. Sie können den * Platzhalter ARNs bei der Bereitstellung der Komponente in eine geheime Liste ändern.

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-Dienst verwendet wird, um den Wert eines Secrets auf dem Kerngerät 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 zusammen mit der Secret Manager-Komponente verwenden, um den Wert eines Secrets auf Ihrem Kerngerät bereitzustellen und auszudrucken.

Um ein Testgeheimnis zu erstellen, bereitzustellen und auszudrucken

1. Erstellen Sie ein Secrets Manager Manager-Geheimnis 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, um ihn in den folgenden Schritten zu verwenden.

Weitere Informationen finden Sie im AWS Secrets Manager Benutzerhandbuch unter Creating a Secret.

2. Stellen Sie die Secret Manager-Komponente (`aws.greengrass.SecretManager`) mit dem folgenden Update zur Zusammenführung der Konfiguration bereit. Geben Sie den ARN des Geheimnisses an, das Sie zuvor erstellt haben.

```
{
  "cloudSecrets": [
    {
      "arn": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyTestGreengrassSecret-abcdef"
    }
  ]
}
```

Weitere Informationen finden Sie unter AWS IoT Greengrass Komponenten auf Geräten bereitstellen oder im Greengrass CLI-Bereitstellungsbefehl.

3. Erstellen und implementieren Sie die Beispielkomponente in diesem Abschnitt mit dem folgenden Update zur Zusammenführung der Konfiguration. Geben Sie den ARN des Geheimnisses 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 AWS IoT Greengrass Komponenten erstellen

4. Sehen Sie sich die AWS IoT Greengrass Core-Softwareprotokolle an, um zu überprüfen, ob die Bereitstellungen erfolgreich waren, und sehen Sie sich das `com.example.PrintSecret` Komponentenprotokoll an, um den geheimen Wert gedruckt zu sehen. Weitere Informationen finden Sie unter AWS IoT Greengrass Protokolle überwachen.

Interagiere mit lokalen Schatten

Verwenden Sie den Shadow-IPC-Dienst, um mit lokalen Schatten auf einem Gerät zu interagieren. Das Gerät, mit dem Sie interagieren möchten, kann Ihr Kerngerät 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. Sie können dann IPC-Operationen in Ihren benutzerdefinierten Komponenten verwenden, um über den Shadow Manager mit lokalen Schatten auf Ihrem Gerät zu interagieren. Um es benutzerdefinierten Komponenten zu ermöglichen, auf Änderungen in lokalen Shadow-Zuständen zu reagieren, können Sie den publish/subscribe IPC service to subscribe to shadow events. For more information about using the publish/subscribe Dienst auch verwenden. Weitere Informationen finden Sie unter. Lokale Nachrichten veröffentlichen/abonnieren

> **Note**
>
> Damit ein Kerngerät mit den Schatten von Client-Geräten interagieren kann, müssen Sie auch die MQTT-Bridge-Komponente konfigurieren und bereitstellen. Weitere Informationen finden Sie unter Aktivieren des Shadow-Managers für die Kommunikation mit Client-Geräten.

Themen

- Minimale SDK-Versionen
- Autorisierung
- GetThingShadow
- UpdateThingShadow
- DeleteThingShadow
- ListNamedShadowsForThing

Minimale SDK-Versionen

In der folgenden Tabelle sind die Mindestversionen von aufgeführt AWS 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	
AWS IoT Device SDK für C++ v2	v1.17.0	
AWS IoT Device SDK für v2 JavaScript	v1.12.0	

Autorisierung

Um den Shadow-IPC-Dienst in einer benutzerdefinierten Komponente zu verwenden, müssen Sie Autorisierungsrichtlinien definieren, die es Ihrer Komponente ermöglichen, mit Shadows zu interagieren. Informationen zur Definition von Autorisierungsrichtlinien finden Sie unterAutorisieren Sie Komponenten zur Ausführung von IPC-Vorgängen.

Autorisierungsrichtlinien für Shadow-Interaktionen haben die folgenden Eigenschaften.

IPC-Dienst-ID: `aws.greengrass.ShadowManager`

Operation	Beschreibung	Ressourcen
`aws.greengrass#GetThingShadow`	Ermöglicht einer Komponente, den Schatten eines Objekts abzurufen.	Eine der folgenden Zeichenketten: • `$aws/things/` *thingName* `/shadow/`, um den Zugriff auf den klassischen Device Shadow zu ermöglichen. • `$aws/things/` *thingName* `/shadow/name/` *shadowName*, um den Zugriff auf einen benannten Schatten zu ermöglichen. • `*` um den Zugriff auf alle Schatten zu ermöglichen.
`aws.greengrass#UpdateThingShadow`	Ermöglicht einer Komponente, den Schatten eines Objekts zu aktualisieren.	Eine der folgenden Zeichenketten: • `$aws/things/` *thingName* `/shadow/`, um den Zugriff auf den klassischen Device Shadow zu ermöglichen. • `$aws/things/` *thingName* `/shadow/name/` *shadowName*, um den Zugriff auf einen benannten Schatten zu ermöglichen.

Operation	Beschreibung	Ressourcen
		• *um den Zugriff auf alle Schatten zu ermöglichen.
`aws.greengrass#DeleteThingShadow`	Ermöglicht einer Komponente, den Schatten eines Objekts zu löschen.	Eine der folgenden Zeichenketten: • `$aws/things/` *thingName* `/shadow/`, um den Zugriff auf den klassischen Device Shadow zu ermöglichen • `$aws/things/` *thingName* `/shadow/name/` *shadowName*, um den Zugriff auf einen benannten Schatten zu ermöglichen • *, um den Zugriff auf alle Schatten zu ermöglichen.
`aws.greengrass#ListNamedShadowsForThing`	Ermöglicht einer Komponente, die Liste der benannten Schatten für ein Ding abzurufen.	Eine Zeichenfolge mit dem Namen eines Dings, die den Zugriff auf das Ding ermöglicht, um seine Schatten aufzulisten. Wird verwendet*, um den Zugriff auf alle Dinge zu ermöglichen.

IPC-Dienst-ID: `aws.greengrass.ipc.pubsub`

Operation	Beschreibung	Ressourcen
`aws.greengrass#SubscribeToTopic`	Ermöglicht einer Komponente, Nachrichten zu den von Ihnen angegebenen Themen zu abonnieren.	Eine der folgenden Themenzeichenfolgen: • *`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 *`shadowTopicPrefix`* hängt von der Art des Schattens ab: • Klassischer Schatten: `$aws/things/` *`thingName`* `/shadow` • Benannter Schatten: `$aws/things/` *`thingName`* `/shadow/name/` *`shadowName`* Wird verwendet*, um den Zugriff auf alle Themen zu ermöglichen.

Operation	Beschreibung	Ressourcen
		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 Literalze ichen. Wenn beispielsweise die Autorisierungsrichtlinie einer Komponente Zugriff auf gewährt`test/topic/#` , kann die Komponente zwar etwas abonnieren`test/topic/#` , aber nicht abonnieren. `test/topic/filter`

Rezeptvariablen in lokalen Shadow-Autorisierungsrichtlinien

Wenn Sie Version 2.6.0 oder höher von Greengrass Nucleus verwenden und die interpolateComponentConfigurationKonfigurationsoption von Greengrass Nucleus auf einstellen`true`, können Sie die `{iot:thingName}` Rezeptvariable in Autorisierungsrichtlinien verwenden. Mit dieser Funktion können Sie eine einzige Autorisierungsrichtlinie für eine Gruppe von Kerngeräten konfigurieren, wobei jedes Kerngerät nur auf seinen eigenen Schatten zugreifen kann. Beispielsweise können Sie einer Komponente den Zugriff auf die folgende Ressource für Shadow-IPC-Operationen gewähren.

```
$aws/things/{iot:thingName}/shadow/
```

Beispiele für Autorisierungsrichtlinien

Anhand der folgenden Beispiele für Autorisierungsrichtlinien können Sie Autorisierungsrichtlinien für Ihre Komponenten konfigurieren.

Example Beispiel: Erlauben Sie einer Gruppe von Kerngeräten, mit lokalen Shadows zu interagieren

> **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. Um diese Funktion zu aktivieren, setzen Sie die interpolateComponentConfigurationKonfigurationsoption von Greengrass Nucleus auf. `true` Ein Beispiel, das für alle Versionen von Greengrass Nucleus funktioniert, finden Sie in der Beispielautorisierungsrichtlinie für ein Single-Core-Gerät.

Die folgende Beispielautorisierungsrichtlinie ermöglicht es der Komponente`com.example.MyShadowInteractionComponent`, mit dem klassischen Geräteshadow und dem benannten Shadow `myNamedShadow` für das Kerngerät zu interagieren, auf dem die Komponente ausgeführt wird. Diese Richtlinie ermöglicht es dieser Komponente auch, Nachrichten zu lokalen Themen für diese Shadows 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 Kerngeräten, mit Client-Geräteschatten zu interagieren

> **Important**
>
> Diese Funktion erfordert Greengrass Nucleus v2.6.0 oder höher, Shadow Manager v2.2.0 oder höher und MQTT Bridge v2.2.0 oder höher. Sie müssen MQTT Bridge so konfigurieren, dass 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 `MyClientDevice` beginnen.

> **Note**
>
> Damit ein Kerngerät mit Client-Geräteschatten interagieren kann, müssen Sie auch die MQTT-Bridge-Komponente konfigurieren und bereitstellen. Weitere Informationen finden Sie unter Aktivieren des Shadow-Managers für die Kommunikation mit Client-Geräten.

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: Erlauben Sie einem Single-Core-Gerät, mit lokalen Shadows zu interagieren

Das folgende Beispiel für eine Autorisierungsrichtlinie ermöglicht es der Komponente`com.example.MyShadowInteractionComponent`, mit dem klassischen Geräteshadow und dem benannten Shadow `myNamedShadow` für das Gerät zu

interagierenMyThingName. Diese Richtlinie ermöglicht es dieser Komponente auch, Nachrichten zu lokalen Themen für diese Shadows 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 Shadow-Status-Änderungen zu reagieren

> **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. Um diese Funktion zu aktivieren, setzen Sie die interpolateComponentConfigurationKonfigurationsoption von Greengrass Nucleus auf. `true`

Ein Beispiel, das für alle Versionen von Greengrass Nucleus funktioniert, finden Sie in der Beispielautorisierungsrichtlinie für ein Single-Core-Gerät.

Das folgende Beispiel für eine Zugriffskontrollrichtlinie ermöglicht es dem Benutzer`com.example.MyShadowReactiveComponent`, Nachrichten `/update/delta` zum Thema für den klassischen Geräteshadow und den benannten Shadow `myNamedShadow` auf jedem Kerngerät zu empfangen, auf dem die Komponente ausgeführt wird.

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: Erlaubt einem Single-Core-Gerät, auf lokale Shadow-Status-Änderungen zu reagieren

Das folgende Beispiel für eine Zugriffskontrollrichtlinie ermöglicht es dem `com.example.MyShadowReactiveComponent` Benutzer, Nachrichten `/update/delta` zum Thema für den klassischen Geräteshadow und zum benannten Shadow `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

Ruft den Schatten für ein bestimmtes Objekt ab.

Anforderung

Die Anfrage 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 Dings anzugeben, setzen Sie diesen Parameter auf eine leere Zeichenfolge (`""`).

> **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.

Typ: `string`

Antwort

Die Antwort dieses Vorgangs enthält die folgenden Informationen:

`payload`

Das Dokument mit dem Antwortstatus als Blob.

Typ: der `object` die folgenden Informationen enthält:

`state`

Die Informationen zum Bundesstaat.

Dieses Objekt enthält die folgenden Informationen.

`desired`

Die Statuseigenschaften und Werte, deren Aktualisierung im Gerät angefordert wurde.

Typ: `map` von Schlüssel-Wert-Paaren

`reported`

Die vom Gerät gemeldeten Statuseigenschaften und Werte.

Typ: `map` von Schlüssel-Wert-Paaren

`delta`

Der Unterschied zwischen den gewünschten und den gemeldeten Zustandseigenschaften und -werten. Diese Eigenschaft ist nur vorhanden, wenn sich der `desired` `reported` Status und der Status unterscheiden.

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`

Datum und Uhrzeit der Generierung der Antwort aus der Epoche.

Typ: `integer`

`clientToken`(Python:`clientToken`)

Das Token, das verwendet wird, um die Anfrage 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 Shadow-Dienst kann die Anforderungsparameter nicht validieren. Dies kann vorkommen, wenn die Anfrage falsch formatiertes JSON oder nicht unterstützte Zeichen enthält.

`ResourceNotFoundError`

Das angeforderte lokale Shadow-Dokument konnte nicht gefunden werden.

`ServiceError`

Ein interner Dienstfehler ist aufgetreten, oder die Anzahl der Anfragen an den IPC-Dienst 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 dieser Vorgang im benutzerdefinierten Komponentencode aufgerufen wird.

Java (IPC client V1)

Example Beispiel: Holen Sie sich ein Ding Shadow

> **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.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: Holen Sie sich ein Ding Shadow

```
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: Holen Sie sich einen Dingschatten

```
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

Aktualisiere den Schatten für das angegebene Ding. Wenn kein Schatten existiert, wird einer erstellt.

Anforderung

Die Anfrage 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 Dings anzugeben, setzen Sie diesen Parameter auf eine leere Zeichenfolge (`""`).

> **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.

Typ: `string`

`payload`

Das Dokument mit dem Status der Anfrage wird als Blob angezeigt.

Typ: der `object` die folgenden Informationen enthält:

`state`

Die zu aktualisierenden Statusinformationen. Dieser IPC-Vorgang 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 beide in derselben Anfrage.

`desired`

Die Statuseigenschaften und Werte, deren Aktualisierung im Gerät angefordert wurde.

Typ: `map` von Schlüssel-Wert-Paaren

`reported`

Die vom Gerät gemeldeten Statuseigenschaften und Werte.

Typ: `map` von Schlüssel-Wert-Paaren

`clientToken`(Python:`client_token`)

(Optional) Das Token, das verwendet wird, um die Anfrage und die entsprechende Antwort des Client-Tokens abzugleichen.

Typ: `string`

`version`

(Optional) Die Version des lokalen Shadow-Dokuments, das aktualisiert werden soll. Der Shadow-Dienst verarbeitet das Update nur, wenn die angegebene Version mit der neuesten Version übereinstimmt.

Typ: `integer`

Antwort

Die Antwort dieses Vorgangs enthält die folgenden Informationen:

`payload`

Das Dokument mit dem Antwortstatus als Blob.

Typ: der `object` die folgenden Informationen enthält:

`state`

Die Informationen zum Bundesstaat.

Dieses Objekt enthält die folgenden Informationen.

`desired`

Die Statuseigenschaften und Werte, deren Aktualisierung im Gerät angefordert wurde.

Typ: `map` von Schlüssel-Wert-Paaren

`reported`

Die vom Gerät gemeldeten Statuseigenschaften und Werte.

Typ: `map` von Schlüssel-Wert-Paaren

`delta`

Die vom Gerät gemeldeten Statuseigenschaften 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`

Datum und Uhrzeit der Generierung der Antwort aus der Epoche.

Typ: `integer`

`clientToken`(Python:`client_token`)

Das Token, das verwendet wird, um die Anfrage und die entsprechende Antwort abzugleichen.

Typ: `string`

`version`

Die Version des lokalen Shadow-Dokuments nach Abschluss der Aktualisierung.

Typ: `integer`

Fehler

Bei diesem Vorgang können die folgenden Fehler zurückgegeben werden.

`ConflictError`

Der lokale Shadow-Dienst ist während des Aktualisierungsvorgangs auf einen Versionskonflikt gestoßen. Dieser Fehler tritt auf, wenn die Version in der Payload der Anforderung nicht mit der Version im neuesten verfügbaren lokalen Shadow-Dokument übereinstimmt.

`InvalidArgumentsError`

Der lokale Shadow-Dienst kann die Anforderungsparameter nicht überprüfen. Dies kann vorkommen, wenn die Anfrage falsch formatiertes JSON oder nicht unterstützte Zeichen enthält.

Ein gültiger Wert `payload` hat die folgenden Eigenschaften:

- Der `state` Knoten ist vorhanden und ist ein Objekt, das die `desired` `reported` Zustandsinformationen 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` Werts darf 64 Zeichen nicht überschreiten.
- Der `version` Wert muss 1 mindestens so hoch sein.

`ServiceError`

Ein interner Dienstfehler ist aufgetreten, oder die Anzahl der Anfragen an den IPC-Dienst 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 dieser Vorgang im benutzerdefinierten Komponentencode aufgerufen wird.

Java (IPC client V1)

Example Beispiel: Aktualisieren Sie einen Ding-Shadow

> **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.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: Aktualisiere einen Ding-Shadow

```
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: Aktualisiere einen Dingschatten

```
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 Shadows 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 dieses Vorgangs 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 Dings anzugeben, setzen Sie diesen Parameter auf eine leere Zeichenfolge (`""`).

> **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.

Typ: `string`

Antwort

Die Antwort dieses Vorgangs enthält die folgenden Informationen:

`payload`

Ein leeres Dokument mit dem Antwortstatus.

Fehler

Dieser Vorgang kann die folgenden Fehler zurückgeben.

`InvalidArgumentsError`

Der lokale Shadow-Dienst kann die Anforderungsparameter nicht validieren. Dies kann vorkommen, wenn die Anfrage falsch formatiertes JSON oder nicht unterstützte Zeichen enthält.

`ResourceNotFoundError`

Das angeforderte lokale Shadow-Dokument konnte nicht gefunden werden.

`ServiceError`

Ein interner Dienstfehler ist aufgetreten, oder die Anzahl der Anfragen an den IPC-Dienst 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 dieser Vorgang im benutzerdefinierten Komponentencode aufgerufen wird.

Java (IPC client V1)

Example Beispiel: Löschen Sie einen Ding-Shadow

> **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.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: Lösche einen Dingschatten

```
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: Lösche einen Dingschatten

```
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

Listet die benannten Schatten für das angegebene Ding auf.

Anforderung

Die Anfrage 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 zum Abrufen der nächsten Ergebnismenge. Dieser Wert wird für nach Seiten organisierte Ergebnisse zurückgegeben und in dem Aufruf verwendet, der die nächste Seite zurückgibt.

Typ: `string`

Antwort

Die Antwort dieses Vorgangs enthält die folgenden Informationen:

`results`

Die Liste der Schattennamen.

Typ: `array`

`timestamp`

(Optional) Datum und Uhrzeit der Generierung der Antwort.

Typ: `integer`

`nextToken`(Python:`next_token`)

(Optional) Der Tokenwert, der in Seitenanfragen verwendet werden soll, um die nächste Seite in der Sequenz abzurufen. Dieses Token ist nicht vorhanden, wenn keine Shadow-Namen mehr zurückgegeben werden können.

Typ: `string`

> **Note**
>
> Wenn die angeforderte Seitengröße genau der Anzahl der Schattennamen in der Antwort entspricht, ist dieses Token vorhanden. Wenn es verwendet wird, gibt es jedoch eine leere Liste zurück.

Fehler

Dieser Vorgang kann die folgenden Fehler zurückgeben.

`InvalidArgumentsError`

Der lokale Shadow-Dienst kann die Anforderungsparameter nicht validieren. Dies kann vorkommen, wenn die Anfrage falsch formatiertes JSON oder nicht unterstützte Zeichen enthält.

`ResourceNotFoundError`

Das angeforderte lokale Shadow-Dokument konnte nicht gefunden werden.

`ServiceError`

Ein interner Dienstfehler ist aufgetreten, oder die Anzahl der Anfragen an den IPC-Dienst 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 dieser Vorgang im benutzerdefinierten Komponentencode aufgerufen wird.

Java (IPC client V1)

Example Beispiel: Listet ein Ding auf, das Schatten heißt

> **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.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: Listet ein Ding auf, das Schatten heißt

```
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: Listet die Namen Schatten eines Dings auf

```
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();
```

Lokale Bereitstellungen und Komponenten verwalten

> **Note**
>
> Diese Funktion ist für Version 2.6.0 und höher der Greengrass Nucleus-Komponente verfügbar.

Verwenden Sie den Greengrass CLI IPC-Dienst, um lokale Bereitstellungen und Greengrass-Komponenten auf dem Kerngerät 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 Kerngerät zu ändern und zu konfigurieren.
- Starten Sie die Greengrass-Komponenten auf dem Kerngerät neu und beenden Sie sie.
- Generieren Sie ein Passwort, mit dem Sie sich an der lokalen Debug-Konsole anmelden können.

Themen

- Minimale SDK-Versionen
- Autorisierung
- CreateLocalDeployment
- ListLocalDeployments
- GetLocalDeploymentStatus
- ListComponents
- GetComponentDetails
- RestartComponent
- StopComponent
- CreateDebugPassword

Minimale SDK-Versionen

In der folgenden Tabelle sind die Mindestversionen von aufgeführt AWS 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 v2 JavaScript	v1.12.0	

Autorisierung

Um den Greengrass CLI IPC-Dienst in einer benutzerdefinierten Komponente zu verwenden, müssen Sie Autorisierungsrichtlinien definieren, die es Ihrer Komponente ermöglichen, lokale Bereitstellungen und Komponenten zu verwalten. Informationen zur Definition von Autorisierungsrichtlinien finden Sie unter. Autorisieren Sie Komponenten zur Ausführung von IPC-Vorgängen

Autorisierungsrichtlinien für die Greengrass-CLI haben die folgenden Eigenschaften.

IPC-Servicekennung: `aws.greengrass.Cli`

Operation	Beschreibung	Ressourcen
`aws.greengrass#CreateLocalDeployment`	Ermöglicht einer Komponente, eine lokale Bereitstellung auf dem Kerngerät zu erstellen.	*

Operation	Beschreibung	Ressourcen
`aws.greengrass#ListLocalDeployments`	Ermöglicht einer Komponente, lokale Bereitstellungen auf dem Kerngerät aufzulisten.	*
`aws.greengrass#GetLocalDeploymentStatus`	Ermöglicht einer Komponente, den Status einer lokalen Bereitstellung auf dem Kerngerät abzurufen.	Eine lokale Bereitstellungs-ID oder * um den Zugriff auf alle lokalen Bereitstellungen zu ermöglichen.
`aws.greengrass#ListComponents`	Ermöglicht einer Komponente, Komponenten auf dem Kerngerät aufzulisten.	*
`aws.greengrass#GetComponentDetails`	Ermöglicht einer Komponente, Details zu einer Komponente auf dem Kerngerät abzurufen.	Ein Komponentenname, z. B. oder`com.example.HelloWorld` , * um den Zugriff auf alle Komponenten zu ermöglichen.
`aws.greengrass#RestartComponent`	Ermöglicht einer Komponente, eine Komponente auf dem Kerngerät neu zu starten.	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 Kerngerät zu stoppen.	Ein Komponentenname, z. B. oder`com.example.HelloWorld` , * um den Zugriff auf alle Komponenten zu ermöglichen.
`aws.greengrass#CreateDebugPassword`	Ermöglicht einer Komponente, ein Passwort zu generieren, mit dem sie sich bei der lokalen Debug-Konsolenkomponente anmelden kann.	*

Example Beispiel für eine Autorisierungsrichtlinie

Das folgende Beispiel für Autorisierungsrichtlinien ermöglicht es einer Komponente, lokale Bereitstellungen zu erstellen, alle lokalen Bereitstellungen und Komponenten anzuzeigen und eine Komponente mit dem Namen neu zu starten und zu beenden. `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 mithilfe bestimmter Komponentenrezepte, Artefakte und Laufzeitargumente.

Dieser Vorgang bietet dieselbe Funktionalität wie der Befehl deployment create in der Greengrass-CLI.

Anforderung

Die Anfrage dieses Vorgangs hat die folgenden Parameter:

`recipeDirectoryPath`(Python:`recipe_directory_path`)

(Optional) Der absolute Pfad zu dem Ordner, der die Rezeptdateien für Komponenten 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 Artifacts 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 Kerngerät installiert werden sollen. Dieses Objekt,`ComponentToVersionMap`, ist eine Map, 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 Kerngerät deinstalliert werden sollen. Geben Sie eine Liste an, in der jeder Eintrag dem Namen einer Komponente entspricht.

`componentToConfiguration`(Python:`component_to_configuration`)

(Optional) Die Konfigurationsaktualisierungen für jede Komponente in der Bereitstellung. Dieses Objekt,`ComponentToConfiguration`, ist eine Map, die die folgenden Schlüssel-Wert-Paare enthält:

`key`

Der Name der Komponente.

`value`

Das JSON-Objekt für das Konfigurationsupdate für die Komponente. Das JSON-Objekt muss das folgende Format haben.

```
{
  "MERGE": {
    "config-key": "config-value"
  },
  "RESET": [
    "path/to/reset/"
  ]
}
```

Weitere Informationen zu Konfigurationsupdates finden Sie unterKomponentenkonfigurationen aktualisieren.

`componentToRunWithInfo`(Python:`component_to_run_with_info`)

(Optional) Die Laufzeitkonfiguration für jede Komponente in der Bereitstellung. Diese Konfiguration umfasst den Systembenutzer, der für die Prozesse der einzelnen Komponenten verantwortlich ist, sowie die Systemlimits, die für jede Komponente gelten sollen. Dieses Objekt,`ComponentToRunWithInfo`, ist eine Map, 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 Greengrass Core-Software die Standardwerte, die Sie auf dem Greengrass-Nucleus konfigurieren. Dieses Objekt,`RunWithInfo`, enthält die folgenden Informationen:

`posixUser`(Python:`posix_user`)

(Optional) Der POSIX-Systembenutzer und optional die Gruppe, die verwendet werden sollen, um diese Komponente auf Linux-Kerngeräten auszuführen. 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 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.

`systemResourceLimits`(Python:`system_resource_limits`)

(Optional) Die Systemressourcenlimits, die für die Prozesse dieser Komponente gelten 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 Menge an CPU-Zeit, die die Prozesse dieser Komponente 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 dieser 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 dieser 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 Komponente nicht.

`memory`

(Optional) Die maximale Menge an RAM (in Kilobyte), die die Prozesse dieser Komponente auf dem Kerngerät verwenden können.

`groupName`(Python:`group_name`)

(Optional) Der Name der Dinggruppe, auf die diese Bereitstellung abzielen soll.

Antwort

Die Antwort dieses Vorgangs enthält die folgenden Informationen:

`deploymentId`(Python:`deployment_id`)

Die ID der lokalen Bereitstellung, die durch die Anfrage erstellt wurde.

ListLocalDeployments

Ruft den Status der letzten 10 lokalen Bereitstellungen ab.

Dieser Vorgang bietet dieselbe Funktionalität wie der Befehl deployment list in der Greengrass-CLI.

Anforderung

Die Anfrage dieses Vorgangs hat keine Parameter.

Antwort

Die Antwort dieses Vorgangs 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 dieselbe Funktionalität wie der Befehl Deployment Status in der Greengrass-CLI.

Anforderung

Die Anfrage dieses Vorgangs hat die folgenden Parameter:

`deploymentId`(Python:`deployment_id`)

Die ID der lokalen Bereitstellung, die abgerufen werden soll.

Antwort

Die Antwort dieses Vorgangs 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 Stammkomponente auf dem Kerngerät ab. Eine Stammkomponente ist eine Komponente, die Sie in einer Bereitstellung angeben. Diese Antwort beinhaltet keine Komponenten, die als Abhängigkeiten von anderen Komponenten installiert werden.

Dieser Vorgang bietet dieselbe Funktionalität wie der Befehl component list in der Greengrass-CLI.

Anforderung

Die Anfrage dieser Operation hat keine Parameter.

Antwort

Die Antwort dieses Vorgangs 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 Kerngerät ab.

Dieser Vorgang bietet dieselbe Funktionalität wie der Befehl Component Details in der Greengrass-CLI.

Anforderung

Die Anfrage dieses Vorgangs hat die folgenden Parameter:

`componentName`(Python:`component_name`)

Der Name der Komponente, die abgerufen werden soll.

Antwort

Die Antwort dieses Vorgangs 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 Kerngerät neu.

> **Note**
>
> Sie können zwar jede Komponente neu starten, wir empfehlen jedoch, nur generische Komponenten neu zu starten.

Dieser Vorgang bietet dieselbe Funktionalität wie der Befehl zum Neustarten der Komponente in der Greengrass-CLI.

Anforderung

Die Anfrage 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 Neustartanfrage. Der Anforderungsstatus kann einer der folgenden sein:

- `SUCCEEDED`
- `FAILED`

`message`

Eine Meldung darüber, warum die Komponente nicht neu gestartet werden konnte, falls die Anfrage fehlgeschlagen ist.

StopComponent

Stoppt die Prozesse einer Komponente auf dem Kerngerät.

> **Note**
>
> Sie können zwar jede Komponente beenden, wir empfehlen jedoch, nur generische Komponenten zu beenden.

Dieser Vorgang bietet dieselbe Funktionalität wie der Befehl zum Stoppen der Komponente in der Greengrass-CLI.

Anforderung

Die Anfrage 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-Anfrage. Der Anforderungsstatus kann einer der folgenden sein:

- `SUCCEEDED`
- `FAILED`

`message`

Eine Meldung darüber, warum die Komponente nicht gestoppt werden konnte, falls die Anfrage 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.

Dieser Vorgang bietet dieselbe Funktionalität wie der get-debug-password Befehl in der Greengrass-CLI.

Anforderung

Die Anfrage dieser Operation hat keine Parameter.

Antwort

Die Antwort dieses Vorgangs enthält die folgenden Informationen:

`username`

Der Benutzername, mit dem Sie sich anmelden möchten.

`password`

Das Passwort, das für die Anmeldung verwendet werden soll.

`passwordExpiration`(Python:`password_expiration`)

Die Zeit, zu der 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. Verwenden Sie diesen Fingerabdruck, wenn Sie die lokale Debug-Konsole öffnen, um zu überprüfen, ob das Zertifikat legitim 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. Verwenden Sie diesen Fingerabdruck, wenn Sie die lokale Debug-Konsole öffnen, um zu überprüfen, ob das Zertifikat legitim und die Verbindung sicher ist.

Client-Geräte authentifizieren und autorisieren

> **Note**
>
> Diese Funktion ist für Version 2.6.0 und höher der Greengrass Nucleus-Komponente verfügbar.

Verwenden Sie den IPC-Dienst für die Client-Geräteauthentifizierung, um eine benutzerdefinierte lokale Broker-Komponente zu entwickeln, mit der lokale IoT-Geräte, z. B. Client-Geräte, eine Verbindung herstellen können.

Um diese IPC-Operationen zu verwenden, fügen Sie Version 2.2.0 oder höher der Komponente für die Client-Geräteauthentifizierung 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 Kerngerät herstellen.
- Erstellen Sie eine Sitzung für ein Client-Gerät, um eine Verbindung zum Kerngerät herzustellen.
- Überprüfen Sie, ob ein Client-Gerät berechtigt ist, eine Aktion auszuführen.
- Sie erhalten eine Benachrichtigung, wenn das Serverzertifikat des Core-Geräts rotiert.

Themen

- Minimale SDK-Versionen
- Autorisierung
- VerifyClientDeviceIdentity
- GetClientDeviceAuthToken
- AuthorizeClientDeviceAction
- SubscribeToCertificateUpdates

Minimale SDK-Versionen

In der folgenden Tabelle sind die Mindestversionen von aufgeführt AWS IoT Device SDK , die Sie für die Interaktion mit dem IPC-Dienst für die Client-Geräteauthentifizierung 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 v2 JavaScript	v1.12.0	

Autorisierung

Um den IPC-Dienst für die Client-Geräteauthentifizierung in einer benutzerdefinierten Komponente zu verwenden, müssen Sie Autorisierungsrichtlinien definieren, die es Ihrer Komponente ermöglichen, diese Vorgänge auszuführen. Hinweise zur Definition 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-Dienst-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 überprüfen und eine Sitzung für dieses Client-Gerät zu erstellen.	*

Operation	Beschreibung	Ressourcen
`aws.greengrass#AuthorizeClientDeviceAction`	Ermöglicht einer Komponente, zu überprüfen, ob ein Client-Gerät berechtigt ist, eine Aktion auszuführen.	*
`aws.greengrass#SubscribeToCertificateUpdates`	Ermöglicht einer Komponente, Benachrichtigungen zu empfangen, wenn das Serverzertifikat des Kerngeräts rotiert.	*
*	Ermöglicht einer Komponente, alle IPC-Authentifizierungsvorgänge des Client-Geräts auszuführen.	*

Beispiele für Autorisierungsrichtlinien

Anhand des folgenden Beispiels für eine Autorisierungsrichtlinie können Sie Autorisierungsrichtlinien für Ihre Komponenten konfigurieren.

Example Beispiel für eine Autorisierungsrichtlinie

Das folgende Beispiel für eine Autorisierungsrichtlinie ermöglicht es einer Komponente, alle IPC-Authentifizierungsvorgänge auf Clientgeräten 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. Dieser Vorgang überprüft, ob es sich bei dem Client-Gerät um ein gültiges Gerät handelt AWS IoT .

Anforderung

Die Anfrage dieses Vorgangs 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 dieses Vorgangs enthält die folgenden Informationen:

`isValidClientDevice`(Python:`is_valid_client_device`)

Ob die Identität des Client-Geräts gültig ist.

GetClientDeviceAuthToken

Überprüft die Anmeldeinformationen eines Client-Geräts und erstellt eine Sitzung für das Client-Gerät. Dieser Vorgang gibt ein Sitzungstoken zurück, das Sie in nachfolgenden Anfragen verwenden können, um Aktionen auf dem Client-Gerät zu autorisieren.

Um ein Clientgerät erfolgreich zu verbinden, muss die Authentifizierungskomponente für das Clientgerät die `mqtt:connect` Genehmigung für die Client-ID erteilen, die das Clientgerät verwendet.

Anforderung

Die Anforderung dieses Vorgangs 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, das 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 dieses Vorgangs enthält die folgenden Informationen:

`clientDeviceAuthToken`(Python:`client_device_auth_token`)

Das Sitzungstoken für das Client-Gerät. Sie können dieses Sitzungstoken in nachfolgenden Anfragen verwenden, um die Aktionen dieses Client-Geräts zu autorisieren.

AuthorizeClientDeviceAction

Überprüfen Sie, ob ein Client-Gerät berechtigt ist, eine Aktion auf einer Ressource auszuführen. Autorisierungsrichtlinien für Client-Geräte spezifizieren die Berechtigungen, die Client-Geräte ausführen können, während sie mit einem Core-Gerät verbunden sind. Sie definieren Autorisierungsrichtlinien für Client-Geräte, wenn Sie die Authentifizierungskomponente für Client-Geräte konfigurieren.

Anforderung

Die Anfrage dieses Vorgangs hat die folgenden Parameter:

`clientDeviceAuthToken`(Python:`client_device_auth_token`)

Das Sitzungstoken für das Client-Gerät.

`operation`

Der zu autorisierende Vorgang.

`resource`

Die Ressource, auf der das Client-Gerät den Vorgang ausführt.

Antwort

Die Antwort dieses Vorgangs enthält die folgenden Informationen:

`isAuthorized`(Python:`is_authorized`)

Ob das Client-Gerät autorisiert ist, den Vorgang auf der Ressource auszuführen.

SubscribeToCertificateUpdates

Abonnieren Sie, um bei jedem Wechsel das neue Serverzertifikat des Core-Geräts zu erhalten. Wenn das Serverzertifikat rotiert, müssen die Broker das Serverzertifikat mit dem neuen Serverzertifikat neu laden.

Die Authentifizierungskomponente für Clientgeräte rotiert die Serverzertifikate standardmäßig alle 7 Tage. Sie können das Rotationsintervall auf 2 bis 10 Tage konfigurieren.

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: `CertificateUpdateEvent`

Anforderung

Die Anforderung dieses Vorgangs hat die folgenden Parameter:

`certificateOptions`(Python:`certificate_options`)

Die Arten von Zertifikatsupdates, die abonniert werden sollen. Dieses Objekt,`CertificateOptions`, enthält die folgenden Informationen:

`certificateType`(Python:`certificate_type`)

Die Art der Zertifikatsupdates, die abonniert werden sollen. Wählen Sie die folgende Option:

- `SERVER`

Antwort

Die Antwort dieses Vorgangs enthält die folgenden Informationen:

`messages`

Der Nachrichtenstrom. 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-Zertifikatskette des Zertifikats.

Interagieren Sie 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 Kerngeräten verbinden, um Folgendes zu tun:

- Interagieren Sie mit MQTT-Nachrichten in Greengrass-Komponenten.
- Nachrichten und Daten zwischen Client-Geräten weiterleiten und. AWS IoT Core
- Interagieren Sie mit Schatten von Client-Geräten in Greengrass-Komponenten.
- Synchronisieren Sie die Schatten von Client-Geräten mit AWS IoT Core.
- Wird IPv6 für lokales Messaging verwendet.

Um eine Verbindung zu einem Kerngerät herzustellen, können Client-Geräte Cloud Discovery verwenden. Client-Geräte stellen eine Verbindung zum AWS IoT Greengrass Cloud-Dienst her, um Informationen über Kerngeräte abzurufen, mit denen sie eine Verbindung herstellen können. Anschließend können sie eine Verbindung zu einem Kerngerät herstellen, um ihre Nachrichten zu verarbeiten und ihre Daten mit dem AWS IoT Core Cloud-Dienst zu synchronisieren.

Sie können einem Tutorial folgen, in dem erklärt wird, wie Sie ein Kerngerät so konfigurieren, dass es eine Verbindung herstellt und mit einem AWS IoT Objekt kommuniziert. In diesem Tutorial wird auch untersucht, wie Sie eine benutzerdefinierte Greengrass-Komponente entwickeln, die mit Client-Geräten interagiert. Weitere Informationen finden Sie unter Tutorial: Interagieren Sie mit lokalen IoT-Geräten über MQTT.

Themen

- AWS-bereitgestellte Client-Gerätekomponenten
- Client-Geräte mit Core-Geräten Connect
- MQTT-Nachrichten zwischen Client-Geräten weiterleiten und AWS IoT Core
- Interagieren Sie mit Client-Geräten in Komponenten
- Interagieren Sie mit Schatten auf Client-Geräten und synchronisieren Sie sie
- IPv6 Für lokales Messaging verwenden
- Problembehandlung bei Client-Geräten

AWS-bereitgestellte Client-Gerätekomponenten

AWS IoT Greengrass stellt die folgenden öffentlichen Komponenten bereit, die Sie auf Kerngeräten bereitstellen können. Diese Komponenten ermöglichen es Client-Geräten, eine Verbindung zu einem Kerngerät herzustellen und mit diesem zu kommunizieren.

> **Note**
>
> Einige der AWS bereitgestellten Komponenten hängen von bestimmten Nebenversionen des Greengrass-Kerns ab. Aufgrund dieser Abhängigkeit müssen Sie diese Komponenten aktualisieren, wenn Sie den Greengrass Nucleus auf eine neue Nebenversion aktualisieren. Informationen zu den spezifischen Versionen von Nucleus, von denen jede Komponente abhängt, finden Sie im entsprechenden Komponententhema. Weitere Informationen zur Aktualisierung des Nucleus finden Sie unterAktualisieren Sie die AWS IoT Greengrass Core-Software (OTA).

Wenn eine Komponente einen Komponententyp sowohl generisch als auch Lambda hat, ist die aktuelle Version der Komponente der generische Typ und eine frühere Version der Komponente ist der Lambda-Typ.

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
Authentifizierung auf Client-Geräten	Ermöglicht es lokalen IoT-Geräten, so genannten Client-Geräten, eine Verbindung zum Kerngerät herzustellen.	Plug-In	Linux, Windows	Ja

Komponente	Beschreibung	Komponententyp	Unterstützes Betriebssystem	Open-Source
IP-Detektor	Meldet Verbindungsinformationen an den MQTT-Broker AWS IoT Greengrass, damit Client-Geräte herausfinden können, wie sie eine Verbindung herstellen können.	Plug-In	Linux, Windows	Ja
MQTT-Brücke	Leitet MQTT-Nachrichten zwischen Client-Geräten weiter, AWS IoT Greengrass veröffentlicht sie lokal und abonniert sie. 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 Kerngerät. Es verwaltet die Speicherung von Shadow-Dokumenten und auch die Synchronisation lokaler Shadow-Zustände mit dem AWS IoT Device Shadow-Dienst.	Plug-In	Linux, Windows	Ja

Client-Geräte mit Core-Geräten Connect

Sie können Cloud Discovery so konfigurieren, dass Client-Geräte mit Kerngeräten verbunden werden. Wenn Sie Cloud Discovery konfigurieren, können Client-Geräte eine Verbindung zum AWS IoT Greengrass Cloud-Dienst herstellen, um Informationen über Kerngeräte abzurufen, mit denen sie eine Verbindung herstellen können. Anschließend können die Client-Geräte versuchen, eine Verbindung zu jedem Kerngerät herzustellen, bis sie erfolgreich verbunden sind.

Um Cloud Discovery zu verwenden, müssen Sie wie folgt vorgehen:

- Ordnen Sie Client-Geräte den Kerngeräten zu, mit denen sie eine Verbindung herstellen können.

- Geben Sie die MQTT-Broker-Endpunkte an, an denen Client-Geräte eine Verbindung zu den einzelnen Kerngeräten herstellen können.
- Stellen Sie Komponenten auf dem Kerngerät bereit, die die Unterstützung für Client-Geräte ermöglichen.

  Sie können auch optionale Komponenten bereitstellen, um Folgendes zu tun:

  - Leitet Nachrichten zwischen Client-Geräten, Greengrass-Komponenten und dem AWS IoT Core Cloud-Dienst weiter.
  - Verwalten Sie die MQTT-Broker-Endpunkte der Kerngeräte automatisch für Sie.
  - Verwalten Sie lokale Client-Geräte-Schatten und synchronisieren Sie Shadows mit dem AWS IoT Core Cloud-Dienst.

Sie müssen auch die AWS IoT Richtlinien für das Kerngerät 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 Cloud Discovery konfiguriert haben, können Sie die Kommunikation zwischen einem Client-Gerät und einem Core-Gerät testen. Weitere Informationen finden Sie unter Testen Sie die Kommunikation mit Client-Geräten.

Themen

- Voraussetzungen
- Greengrass-Komponenten für die Unterstützung von Client-Geräten
- Konfigurieren Sie Cloud Discovery (Konsole)
- Cloud Discovery konfigurieren ()AWS CLI
- Client-Geräte zuordnen
- Authentifizierung von Clients im Offlinemodus
- Endpunkte von Kerngeräten verwalten
- Wählen Sie einen MQTT-Broker
- Client-Geräte über einen AWS IoT Greengrass MQTT-Broker mit einem Core-Gerät verbinden
- Testen Sie die Kommunikation mit Client-Geräten
- Greengrass-Entdeckungs-API RESTful

Voraussetzungen

Um Client-Geräte mit einem Core-Gerät zu verbinden, müssen Sie über Folgendes verfügen:

- Auf dem Kerngerät muss Greengrass Nucleus v2.2.0 oder höher ausgeführt werden.
- Die Greengrass-Servicerolle, die AWS IoT Greengrass für Sie AWS-Konto in der AWS Region gilt, in der das Kerngerät betrieben wird. Weitere Informationen finden Sie unter Konfigurieren Sie die Greengrass-Servicerolle.
- Die AWS IoT Richtlinie des Kerngeräts 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 Netzwerkverbindungsinformationen des Kerngeräts an den AWS IoT Greengrass Cloud-Dienst meldet.
  - `iot:GetThingShadow`iot:UpdateThingShadow, und `iot:DeleteThingShadow` — (Optional) Diese Berechtigungen sind erforderlich, um die Shadow-Manager-Komponente zum Synchronisieren von Schatten auf Client-Geräten zu verwenden AWS IoT Core. Diese Funktion erfordert Greengrass Nucleus v2.6.0 oder höher, Shadow Manager v2.2.0 oder höher und MQTT Bridge v2.2.0 oder höher.

  Weitere Informationen finden Sie unter Konfigurieren Sie die AWS IoT Ding-Richtlinie.

  > **Note**
  >
  > Wenn Sie bei der Installation der AWS IoT Greengrass Core-Software die AWS IoT Standardrichtlinie verwendet haben, verfügt das Core-Gerät über eine AWS IoT Richtlinie, die den Zugriff auf alle Aktionen ermöglicht (). AWS IoT Greengrass `greengrass:*`

- AWS IoT Dinge, die Sie als Client-Geräte verbinden können. Weitere Informationen finden Sie im AWS IoT Core Entwicklerhandbuch unter AWS IoT Ressourcen erstellen.
- Das Client-Gerät muss über eine Client-ID eine Verbindung herstellen. Eine Client-ID ist ein Dingname. Keine andere Client-ID wird akzeptiert.
- Die AWS IoT Richtlinie jedes Client-Geräts muss die `greengrass:Discover` Genehmigung zulassen. Weitere Informationen finden Sie unter Minimale AWS IoT Richtlinie für Client-Geräte.

Themen

- Konfigurieren Sie die Greengrass-Servicerolle
- Konfigurieren Sie die AWS IoT Ding-Richtlinie

Konfigurieren Sie die Greengrass-Servicerolle

Die Greengrass-Servicerolle ist eine AWS Identity and Access Management (IAM) -Servicerolle, die den AWS IoT Greengrass Zugriff auf Ressourcen von AWS Diensten in Ihrem Namen autorisiert. Diese Rolle ermöglicht es, die Identität von Client-Geräten AWS IoT Greengrass zu überprüfen und die Verbindungsinformationen der Kerngeräte zu verwalten.

Wenn Sie die Greengrass-Servicerolle in dieser Region noch nicht eingerichtet haben, müssen Sie eine Greengrass-Servicerolle AWS IoT Greengrass für Sie AWS-Konto in dieser Region zuordnen.

Wenn Sie die Seite „Core-Geräteerkennung konfigurieren“ in der AWS IoT Greengrass Konsole verwenden, AWS IoT Greengrass wird die Greengrass-Servicerolle für Sie eingerichtet. Andernfalls können Sie sie manuell über die AWS IoT Konsole oder die AWS IoT Greengrass API einrichten.

In diesem Abschnitt überprüfen Sie, ob die Greengrass-Servicerolle eingerichtet ist. Wenn sie nicht eingerichtet ist, erstellen Sie eine neue Greengrass-Servicerolle, die Sie AWS IoT Greengrass für Sie AWS-Konto in dieser Region zuordnen können.

Konfiguration der Greengrass-Servicerolle (Konsole)

1. Prüfen Sie, ob die Greengrass-Servicerolle AWS IoT Greengrass für Sie 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 zugewiesen ist, erfüllen Sie diese Anforderung, um die IP-Detektorkomponente verwenden zu können. Fahren Sie mit Konfigurieren Sie die AWS IoT Ding-Richtlinie fort.
2. Wenn die Greengrass-Servicerolle AWS IoT Greengrass für Sie AWS-Konto in dieser Region nicht zugeordnet ist, erstellen Sie eine Greengrass-Servicerolle und ordnen Sie sie zu. 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 die Option aus AWS-Service.

   ii. Wählen Sie unter Anwendungsfall, Anwendungsfälle für andere AWS-Services die Option Greengrass und anschließend Greengrass aus. Diese Option gibt an, dass eine vertrauenswürdige Entität hinzugefügt AWS IoT Greengrass werden soll, die diese Rolle übernehmen kann.

   iii. Wählen Sie Weiter aus.

   iv. Wählen Sie unter Berechtigungsrichtlinien die aus, AWSGreengrassResourceAccessRolePolicydie der Rolle zugewiesen werden sollen.

   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 anhängen aus.

h. Wählen Sie im Modal Update Greengrass Service Role die IAM-Rolle aus, die Sie erstellt haben, und wählen Sie dann Rolle anhängen aus.

Konfigurieren Sie die Greengrass-Servicerolle ()AWS CLI

1. Prüfen Sie, ob die Greengrass-Servicerolle AWS IoT Greengrass für Sie AWS-Konto in dieser Region zugeordnet ist.

   ```
   aws greengrassv2 get-service-role-for-account
   ```

   Wenn die Greengrass-Servicerolle zugeordnet ist, gibt der Vorgang eine Antwort zurück, die Informationen über die Rolle enthält.

Wenn Ihnen eine Greengrass-Servicerolle zugewiesen ist, erfüllen Sie diese Anforderung, um die IP-Detektorkomponente verwenden zu können. Fahren Sie mit Konfigurieren Sie die AWS IoT Ding-Richtlinie fort.

2. Wenn die Greengrass-Servicerolle AWS IoT Greengrass für Sie AWS-Konto in dieser Region nicht zugeordnet ist, erstellen Sie eine Greengrass-Servicerolle und ordnen Sie sie zu. Gehen Sie wie folgt vor:

   a. Erstellen Sie eine Rolle mit einer Vertrauensrichtlinie, die es Ihnen ermöglicht, die Rolle AWS IoT Greengrass zu übernehmen. In diesem Beispiel wird eine Rolle namens `Greengrass_ServiceRole` erstellt, aber Sie können einen anderen Namen verwenden. Wir empfehlen, dass Sie auch die Kontextschlüssel `aws:SourceArn` und die `aws:SourceAccount` globalen Bedingungsschlüssel in Ihre Vertrauensrichtlinie aufnehmen, um das Sicherheitsproblem „Confused Deputy“ zu vermeiden. Die Bedingungskontextschlüssel beschränken den Zugriff, sodass nur Anfragen zugelassen werden, die vom angegebenen Konto und Greengrass-Workspace stammen. Weitere Informationen zum Confused-Deputy-Problem finden Sie Serviceübergreifende Confused-Deputy-Prävention.

      Linux 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. Verknüpfen Sie die Greengrass-Servicerolle mit AWS IoT Greengrass für Sie AWS-Konto. *role-arn*Ersetzen Sie es durch den ARN der Servicerolle.

```
aws greengrassv2 associate-service-role-to-account --role-arn role-arn
```

Der Vorgang gibt die folgende Antwort zurück, wenn er erfolgreich ist.

```
{
  "associatedAt": "timestamp"
}
```

Konfigurieren Sie die AWS IoT Ding-Richtlinie

Core-Geräte verwenden X.509-Gerätezertifikate, um Verbindungen zu zu autorisieren. AWS Sie fügen Gerätezertifikaten AWS IoT Richtlinien hinzu, um die Berechtigungen für ein Kerngerät zu definieren. Weitere Informationen erhalten Sie unter AWS IoT Richtlinien für den Betrieb 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 Netzwerkverbindungsinformationen des Kerngeräts an den AWS IoT Greengrass Cloud-Dienst meldet.
- `iot:GetThingShadow`, `iot:UpdateThingShadow`, und `iot:DeleteThingShadow` — (Optional) Diese Berechtigungen sind erforderlich, um die Shadow-Manager-Komponente zum Synchronisieren von Schatten auf Client-Geräten zu verwenden AWS IoT Core. Diese Funktion erfordert Greengrass Nucleus v2.6.0 oder höher, Shadow Manager v2.2.0 oder höher und MQTT Bridge v2.2.0 oder höher.

In diesem Abschnitt überprüfen Sie die AWS IoT Richtlinien für Ihr Kerngerät und fügen alle erforderlichen Berechtigungen hinzu, die noch fehlen. Wenn Sie das AWS IoT Greengrass Core-

Softwareinstallationsprogramm zur Bereitstellung von Ressourcen verwendet haben, verfügt Ihr Kerngerät über eine AWS IoT Richtlinie, die den Zugriff auf alle AWS IoT Greengrass Aktionen ermöglicht (`greengrass:*`). In diesem Fall müssen Sie die AWS IoT Richtlinie nur aktualisieren, wenn Sie die Shadow-Manager-Komponente zur Synchronisierung von Geräteschatten bereitstellen möchten AWS IoT Core. Andernfalls können Sie diesen Abschnitt überspringen.

Konfigurieren Sie die AWS IoT Ding-Richtlinie (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 Kerngerät aus.
3. Wählen Sie auf der Seite mit den Details zum Kerngerät den Link zum Ding des Kerngeräts aus. Dieser Link öffnet die Seite mit den Ding-Details in der AWS IoT Konsole.
4. Wählen Sie auf der Seite mit den Ding-Details die Option Zertifikate aus.
5. Wählen Sie auf der Registerkarte Zertifikate das aktive Zertifikat der Sache aus.
6. Wählen Sie auf der Seite mit den Zertifikatsdetails die Option Richtlinien aus.
7. Wählen Sie auf der Registerkarte Richtlinien die AWS IoT Richtlinie aus, die Sie überprüfen und aktualisieren möchten. Sie können jeder Richtlinie, die mit dem aktiven Zertifikat des Kerngeräts verknüpft ist, die erforderlichen Berechtigungen hinzufügen.

   > **Note**
   >
   > Wenn Sie das AWS IoT Greengrass Core-Softwareinstallationsprogramm zur Bereitstellung von Ressourcen verwendet haben, gibt es zwei AWS IoT Richtlinien. Wir empfehlen, dass Sie die genannte Richtlinie wählen GreengrassV2IoTThingPolicy, falls sie existiert. Kerngeräte, die Sie mit dem Schnellinstallationsprogramm erstellen, verwenden standardmäßig diesen Richtliniennamen. Wenn Sie dieser Richtlinie Berechtigungen hinzufügen, gewähren Sie diese Berechtigungen auch anderen Kerngeräten, 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 fehlenden erforderlichen Berechtigungen hinzu.

   - `greengrass:PutCertificateAuthorities`
   - `greengrass:VerifyClientDeviceIdentity`
   - `greengrass:VerifyClientDeviceIoTCertificateAssociation`

- `greengrass:GetConnectivityInfo`
- `greengrass:UpdateConnectivityInfo`— (Optional) Diese Berechtigung ist erforderlich, um die IP-Detektorkomponente zu verwenden, die die Netzwerkverbindungsinformationen des Kerngeräts an den AWS IoT Greengrass Cloud-Dienst meldet.
- `iot:GetThingShadow`iot:UpdateThingShadow, und `iot:DeleteThingShadow` — (Optional) Diese Berechtigungen sind erforderlich, um die Shadow-Manager-Komponente zum Synchronisieren von Schatten auf Client-Geräten zu verwenden AWS IoT Core. Diese Funktion erfordert Greengrass Nucleus v2.6.0 oder höher, Shadow Manager v2.2.0 oder höher und MQTT Bridge v2.2.0 oder höher.

10. (Optional) Damit das Core-Gerät Schatten mit synchronisieren kann, fügen Sie der Richtlinie die folgende Anweisung AWS IoT Core hinzu. Wenn Sie mit Shadows auf Client-Geräten interagieren, diese aber nicht synchronisieren möchten AWS IoT Core, überspringen Sie diesen Schritt. Ersetzen Sie *region* und *account-id* durch die Region, die Sie verwenden, und Ihre AWS-Konto Nummer.

    - Diese Beispielanweisung ermöglicht den Zugriff auf die Geräteschatten aller Dinge. Um den bewährten Sicherheitspraktiken zu folgen, können Sie den Zugriff nur auf das Kerngerät und die Client-Geräte beschränken, die Sie mit dem Kerngerät 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 Aussage hinzugefügt haben, könnte das Richtliniendokument dem folgenden Beispiel ähneln.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect",
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive",
        "greengrass:*"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:DeleteThingShadow"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:thing/*"
      ]
    }
  ]
}
```

11. Um eine neue Richtlinienversion als aktive Version festzulegen, wählen Sie unter Status der Richtlinienversion die Option Die bearbeitete Version als aktive Version für diese Richtlinie festlegen aus.
12. Wählen Sie Als neue Version speichern aus.

Konfigurieren Sie die AWS IoT Ding-Richtlinie (AWS CLI)

1. Listet die Principals für das AWS IoT Ding des Kerngeräts auf. Bei den Ding-Prinzipalen kann es sich um X.509-Gerätezertifikate oder andere Identifikationen handeln. Führen Sie den folgenden Befehl aus und *MyGreengrassCore* ersetzen Sie ihn durch den Namen des Kerngeräts.

```
aws iot list-thing-principals --thing-name MyGreengrassCore
```

Der Vorgang gibt eine Antwort zurück, in der die Ding-Prinzipale des Kerngeräts aufgeführt sind.

```
{
    "principals": [
        "arn:aws:iot:us-west-2:123456789012:cert/certificateId"
    ]
}
```

2. Identifizieren Sie das aktive Zertifikat des Kerngeräts. Führen Sie den folgenden Befehl aus und *certificateId* ersetzen Sie ihn durch die ID jedes Zertifikats aus dem vorherigen Schritt, bis Sie das aktive Zertifikat gefunden haben. Die Zertifikat-ID ist die hexadezimale Zeichenfolge am Ende des Zertifikat-ARN. Das `--query` Argument gibt an, dass nur der Status des Zertifikats ausgegeben werden soll.

```
aws iot describe-certificate --certificate-id certificateId --query
 'certificateDescription.status'
```

Die Operation gibt den Status des Zertifikats als Zeichenfolge zurück. Wenn das Zertifikat beispielsweise aktiv ist, wird bei diesem Vorgang eine Ausgabe ausgegeben`"ACTIVE"`.

3. Listet die AWS IoT Richtlinien auf, die mit dem Zertifikat verknüpft 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 an das Zertifikat angehängt 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 Richtlinie aus, die Sie anzeigen und aktualisieren möchten.

> **Note**
>
> Wenn Sie das AWS IoT Greengrass Core-Softwareinstallationsprogramm zur Bereitstellung von Ressourcen verwendet haben, gelten zwei AWS IoT Richtlinien. Wir empfehlen, dass Sie die genannte Richtlinie wählen GreengrassV2IoTThingPolicy, falls sie existiert. Kerngeräte, die Sie mit dem Schnellinstallationsprogramm erstellen, verwenden standardmäßig diesen Richtliniennamen. Wenn Sie dieser Richtlinie Berechtigungen hinzufügen, gewähren Sie diese Berechtigungen auch anderen Kerngeräten, die diese Richtlinie verwenden.

5. Holen Sie sich das Dokument der Richtlinie. Führen Sie den folgenden Befehl aus und *GreengrassV2IoTThingPolicy* ersetzen Sie ihn durch den Namen der Richtlinie.

```
aws iot get-policy --policy-name GreengrassV2IoTThingPolicy
```

Der Vorgang gibt eine Antwort zurück, die das Dokument der Richtlinie und andere Informationen zur Richtlinie enthält. Das Richtliniendokument ist ein JSON-Objekt, das als Zeichenfolge serialisiert ist.

```
{
    "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 Zeichenfolge des Richtliniendokuments in ein JSON-Objekt zu konvertieren, und speichern Sie sie dann in einer Datei mit dem Name`iot-policy.json`.

   Wenn Sie beispielsweise das jq-Tool 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 fehlenden erforderlichen Berechtigungen hinzu.

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

   ```
   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 Netzwerkverbindungsinformationen des Kerngeräts an den AWS IoT Greengrass Cloud-Dienst meldet.
- `iot:GetThingShadowiot:UpdateThingShadow`, und `iot:DeleteThingShadow` — (Optional) Diese Berechtigungen sind erforderlich, um die Shadow-Manager-Komponente zum Synchronisieren von Schatten auf Client-Geräten zu verwenden AWS IoT Core. Diese Funktion erfordert Greengrass Nucleus v2.6.0 oder höher, Shadow Manager v2.2.0 oder höher und MQTT Bridge v2.2.0 oder höher.

8. Speichern Sie die Änderungen als neue Version der Richtlinie. Führen Sie den folgenden Befehl aus und *`GreengrassV2IoTThingPolicy`* ersetzen Sie ihn durch den Namen der Richtlinie.

```
aws iot create-policy-version --policy-name GreengrassV2IoTThingPolicy --policy-
document file://iot-policy.json --set-as-default
```

Der Vorgang gibt eine Antwort zurück, die dem folgenden Beispiel ähnelt, wenn er 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**
>
> Auf dem Kerngerät muss Greengrass Nucleus v2.2.0 oder höher ausgeführt werden, 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 für Client-Geräte bereit, um Client-Geräte zu authentifizieren und Aktionen auf Client-Geräten zu autorisieren. Mit dieser Komponente können Ihre Geräte eine AWS IoT Verbindung zu einem Kerngerät herstellen.

  Diese Komponente erfordert eine gewisse Konfiguration, um sie verwenden zu können. Sie müssen Gruppen von Client-Geräten und die Operationen angeben, zu deren Ausführung jede Gruppe berechtigt ist, z. B. Verbindungen herzustellen und über MQTT zu kommunizieren. Weitere Informationen finden Sie unter Konfiguration der Authentifizierungskomponente für Client-Geräte.
- MQTT 3.1.1-Broker (Moquette) (`aws.greengrass.clientdevices.mqtt.Moquette`)

  Stellen Sie die Moquette MQTT-Broker-Komponente bereit, um einen einfachen MQTT-Broker auszuführen. Der Moquette MQTT-Broker ist mit MQTT 3.1.1 kompatibel und bietet lokale Unterstützung für QoS 0, QoS 1, QoS 2, gespeicherte Nachrichten, Lastwill-Nachrichten und persistente Abonnements.

  Sie müssen diese Komponente nicht konfigurieren, um sie verwenden zu können. Sie können jedoch den Port konfigurieren, an dem diese Komponente den MQTT-Broker betreibt. Standardmäßig verwendet es Port 8883.
- MQTT 5-Broker (EMAX) (`aws.greengrass.clientdevices.mqtt.EMQX`)

  > **Note**
  >
  > Um den EMQX MQTT 5-Broker zu verwenden, müssen Sie Greengrass Nucleus v2.6.0 oder höher und Client Device Auth v2.2.0 oder höher verwenden.

Stellen Sie die EMQX MQTT-Broker-Komponente bereit, um die Funktionen von MQTT 5.0 bei der Kommunikation zwischen Client-Geräten und dem Core-Gerät zu verwenden. Der EMQX MQTT-Broker ist mit MQTT 5.0 kompatibel und unterstützt Ablaufintervalle für Sitzungen und Nachrichten, Benutzereigenschaften, gemeinsame Abonnements, Themenaliase und mehr.

Sie müssen diese Komponente nicht konfigurieren, um sie verwenden zu können. Sie können jedoch den Port konfigurieren, an dem diese Komponente den MQTT-Broker betreibt. Standardmäßig verwendet es Port 8883.

- MQTT-Brücke (`aws.greengrass.clientdevices.mqtt.Bridge`)

  (Optional) Stellen Sie die MQTT-Bridge-Komponente bereit, um Nachrichten zwischen Client-Geräten (lokales MQTT), lokales Publish/Subscribe und MQTT weiterzuleiten. AWS IoT Core Konfigurieren Sie diese Komponente, um Client-Geräte mit Client-Geräten von Greengrass-Komponenten zu synchronisieren AWS IoT Core und mit diesen zu interagieren.

  Für die Verwendung dieser 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 Kerngeräts automatisch an den AWS IoT Greengrass Cloud-Dienst zu melden. Sie können diese Komponente nicht verwenden, wenn Sie eine komplexe Netzwerkkonfiguration haben, z. B. eine, bei der ein Router den MQTT-Broker-Port an das Kerngerät weiterleitet.

  Sie müssen diese Komponente nicht konfigurieren, um sie zu verwenden.

- Schattenmanager (`aws.greengrass.ShadowManager`)

  > **Note**
  >
  > Um Client-Geräteschatten zu verwalten, müssen Sie Greengrass Nucleus v2.6.0 oder höher, Shadow Manager v2.2.0 oder höher und MQTT Bridge v2.2.0 oder höher verwenden.

  (Optional) Stellen Sie die Shadow Manager-Komponente bereit, um Client-Geräteschatten auf dem Kerngerät zu verwalten. Greengrass-Komponenten können Client-Geräteschatten 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-Dienst synchronisiert werden.

Um diese Komponente mit Client-Geräteshadows zu verwenden, müssen Sie die MQTT-Bridge-Komponente so konfigurieren, dass sie Nachrichten zwischen Client-Geräten und dem Shadow Manager weiterleitet, der lokales Publizieren/Abonnieren verwendet. Andernfalls ist für die Verwendung dieser Komponente keine Konfiguration erforderlich, für die Synchronisierung von Geräteschatten jedoch eine Konfiguration.

> **Note**
>
> Wir empfehlen, nur eine MQTT-Broker-Komponente bereitzustellen. Die Komponenten MQTT-Bridge und IP-Detektor funktionieren jeweils nur mit einer MQTT-Broker-Komponente. Wenn Sie mehrere MQTT-Broker-Komponenten bereitstellen, müssen Sie sie so konfigurieren, dass sie unterschiedliche Ports verwenden.

Konfigurieren Sie Cloud Discovery (Konsole)

Sie können die AWS IoT Greengrass Konsole verwenden, um Client-Geräte zuzuordnen, zentrale Geräteendpunkte zu verwalten und Komponenten bereitzustellen, um die Unterstützung von Client-Geräten zu aktivieren. Weitere Informationen finden Sie unter Schritt 2: Aktivieren Sie die Unterstützung für Client-Geräte.

Cloud Discovery konfigurieren ()AWS CLI

Sie können das AWS Command Line Interface (AWS CLI) verwenden, um Client-Geräte zuzuordnen, zentrale Geräteendpunkte zu verwalten und Komponenten bereitzustellen, um die Unterstützung von Client-Geräten zu aktivieren. Weitere Informationen finden Sie hier:

- Zuordnungen von Client-Geräten verwalten ()AWS CLI
- Endpunkte von Kerngeräten verwalten
- AWS-bereitgestellte Client-Gerätekomponenten
- Erstellen von Bereitstellungen

Client-Geräte zuordnen

Um Cloud Discovery zu verwenden, ordnen Sie Client-Geräte einem Core-Gerät zu, sodass sie das Core-Gerät erkennen können. Anschließend können sie die Greengrass Discovery API verwenden, um Verbindungsinformationen und Zertifikate für ihre zugehörigen Kerngeräte abzurufen.

Ebenso können Sie Client-Geräte von einem Kerngerät trennen, um zu verhindern, dass sie das Kerngerät erkennen.

Themen

- Zuordnungen von Client-Geräten verwalten (Konsole)
- Zuordnungen von Client-Geräten verwalten ()AWS CLI
- Client-Gerätezuordnungen (API) verwalten

Zuordnungen von Client-Geräten verwalten (Konsole)

Sie können die AWS IoT Greengrass Konsole verwenden, um Client-Gerätezuordnungen anzuzeigen, hinzuzufügen und zu löschen.

So zeigen Sie Client-Gerätezuordnungen für ein Kerngerät (Konsole) an

1. Navigieren Sie zur AWS IoT Greengrass -Konsole.
2. Wählen Sie Core-Geräte aus.
3. Wählen Sie das zu verwaltende Kerngerät aus.
4. Wählen Sie auf der Detailseite des Kerngeräts die Registerkarte Client-Geräte aus.
5. Im Abschnitt Zugeordnete Client-Geräte können Sie sehen, welche Client-Geräte (AWS IoT Dinge) mit dem Core-Gerät verknüpft sind.

So ordnen Sie Client-Geräte einem Core-Gerät (Konsole) zu

1. Navigieren Sie zur AWS IoT Greengrass -Konsole.
2. Wählen Sie Core-Geräte aus.
3. Wählen Sie das zu verwaltende Kerngerät aus.
4. Wählen Sie auf der Detailseite des Kerngeräts die Registerkarte Client-Geräte aus.
5. Wählen Sie im Abschnitt Zugeordnete Client-Geräte die Option Client-Geräte zuordnen aus.

6. Gehen Sie im Modalmodus Client-Geräte mit Kerngerät verknüpfen für jedes zu verknüpfende Client-Gerät wie folgt vor:

   a. Geben Sie den Namen des Objekts AWS IoT ein, das als Client-Gerät verknüpft werden soll.

   b. Wählen Sie Hinzufügen aus.

7. Wählen Sie Associate aus.

   Die Client-Geräte, die Sie verknüpft haben, können jetzt die Greengrass Discovery API verwenden, um dieses Kerngerät zu erkennen.

Um Client-Geräte von einem Kerngerät (Konsole) zu trennen

1. Navigieren Sie zur AWS IoT Greengrass -Konsole.
2. Wählen Sie Core-Geräte aus.
3. Wählen Sie das zu verwaltende Kerngerät aus.
4. Wählen Sie auf der Detailseite des Kerngeräts 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 Bestätigungsmodus die Option Trennen aus.

   Die Client-Geräte, deren Verbindung Sie getrennt haben, können die Greengrass Discovery API nicht mehr verwenden, um dieses Kerngerät zu erkennen.

Zuordnungen von Client-Geräten verwalten ()AWS CLI

Sie können das AWS Command Line Interface (AWS CLI) verwenden, um Client-Gerätezuordnungen für ein Kerngerät zu verwalten.

Um Client-Gerätezuordnungen für ein Kerngerät anzuzeigen (AWS CLI)

- Verwenden Sie den folgenden Befehl: list-client-devices-associated- with-core-device.

Um Client-Geräte einem Kerngerät zuzuordnen (AWS CLI)

- Verwenden Sie den folgenden Befehl: batch-associate-client-device- with-core-device.

Um Client-Geräte von einem Kerngerät zu trennen ()AWS CLI

- Verwenden Sie den folgenden Befehl: batch-disassociate-client-device- from-core-device.

Client-Gerätezuordnungen (API) verwalten

Sie können die AWS API verwenden, um Client-Gerätezuordnungen für ein Kerngerät zu verwalten.

Um Client-Gerätezuordnungen für ein Kerngerät (AWS API) anzuzeigen

- Verwenden Sie den folgenden Vorgang: ListClientDevicesAssociatedWithCoreDevice.

Um Client-Geräte einem Kerngerät (AWS API) zuzuordnen

- Verwenden Sie den folgenden Vorgang: BatchAssociateClientDeviceWithCoreDevice.

Um Client-Geräte von einem Kerngerät (AWS API) zu trennen

- Verwenden Sie den folgenden Vorgang: 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. Client-Geräte zuordnen

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.

Endpunkte von Kerngeräten verwalten

Wenn Sie Cloud Discovery verwenden, speichern Sie MQTT-Broker-Endpunkte für Kerngeräte im Cloud-Dienst. AWS IoT Greengrass Client-Geräte stellen eine Verbindung her AWS IoT Greengrass , um diese Endpunkte und andere Informationen für ihre zugehörigen Kerngeräte abzurufen.

Für jedes Kerngerät können Sie Endpunkte automatisch oder manuell verwalten.

- Automatische Verwaltung von Endpunkten mit dem IP-Detektor

  Sie können die IP-Detektorkomponente einsetzen, um die Endpunkte der Kerngeräte automatisch für Sie zu verwalten, wenn Sie über eine nicht komplexe Netzwerkkonfiguration verfügen, z. B. wenn sich die Client-Geräte im selben Netzwerk wie das Kerngerät befinden. Sie können die IP-Detektorkomponente nicht verwenden, wenn sich das Kerngerät beispielsweise hinter einem Router befindet, der den MQTT-Broker-Port an das Kerngerät weiterleitet.

  Die IP-Detektorkomponente ist auch nützlich, wenn Sie sie in Dinggruppen bereitstellen, da sie die Endpunkte für alle Kerngeräte in der Dinggruppe 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 der Kerngeräte 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
- Manuelles Verwalten von Endpunkten

Verwenden Sie den IP-Detektor, um Endpunkte automatisch zu verwalten

Wenn Sie über ein einfaches Netzwerk-Setup verfügen, z. B. wenn sich die Client-Geräte im selben Netzwerk wie das Kerngerät befinden, können Sie die IP-Detektorkomponente einsetzen, um Folgendes zu tun:

- Überwachen Sie die lokalen Netzwerkverbindungsinformationen des Greengrass Core-Geräts. Zu diesen Informationen gehören die Netzwerkendpunkte des Kerngeräts und der Port, an dem der MQTT-Broker betrieben wird.
- Melden Sie die Konnektivitätsinformationen des Kerngeräts an den AWS IoT Greengrass Cloud-Dienst.

Die IP-Detektorkomponente überschreibt Endpunkte, die Sie manuell festlegen.

> **Important**
>
> Die AWS IoT Richtlinie des Kerngeräts muss die `greengrass:UpdateConnectivityInfo` Erlaubnis zur Verwendung der IP-Detektorkomponente zulassen. Weitere Informationen erhalten Sie unter AWS IoT Richtlinien für den Betrieb auf Datenebene und Konfigurieren Sie die AWS IoT Ding-Richtlinie.

Sie können einen der folgenden Schritte ausführen, um die IP-Detektorkomponente bereitzustellen:

- Verwenden Sie die Seite Discovery konfigurieren in der Konsole. Weitere Informationen finden Sie unter Konfigurieren Sie Cloud Discovery (Konsole).
- Erstellen und überarbeiten Sie Bereitstellungen, um den IP-Detektor einzubeziehen. Sie können die Konsole oder die AWS API verwenden AWS CLI, um Bereitstellungen zu verwalten. Weitere Informationen finden Sie unter Erstellen von Bereitstellungen.

Stellen Sie die IP-Detektorkomponente (Konsole) bereit

1. Wählen Sie im Navigationsmenü der AWS IoT Greengrass Konsole die Option Komponenten aus.
2. Wählen Sie auf der Seite Komponenten die Registerkarte Öffentliche Komponenten und wählen Sie dann aws.greengrass.clientdevices.IPDetector.
3. Auf der aws.greengrass.clientdevices.IPDetectorWählen Sie auf der Seite Deploy aus.
4. Wählen Sie unter Zur Bereitstellung hinzufügen eine vorhandene Bereitstellung aus, die Sie überarbeiten möchten, oder erstellen Sie eine neue Bereitstellung und klicken Sie dann auf Weiter.

5. Wenn Sie eine neue Bereitstellung erstellen möchten, wählen Sie das Ziel-Core-Gerät oder die Dinggruppe für die Bereitstellung aus. Wählen Sie auf der Seite „Ziel angeben“ unter Bereitstellungsziel ein Kerngerät oder eine Dinggruppe aus, und klicken Sie dann auf Weiter.
6. Vergewissern Sie sich auf der Seite „Komponenten auswählen“, dass aws.greengrass.clientdevices.IPDetectorDie Komponente ist ausgewählt, wählen Sie Weiter.
7. Wählen Sie auf der Seite Komponenten konfigurieren aws.greengrass.clientdevices.IPDetector, und gehen Sie dann wie folgt vor:

   a. Wählen Sie Komponente konfigurieren aus.

   b. In der Konfiguration aws.greengrass.clientdevices.IPDetectorModal, unter Konfigurationsupdate können Sie unter Konfiguration zum Zusammenführen 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, 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.
      - `includeIPv4LinkLocalAddrs`— (Optional) Sie können diese Option aktivieren, um IPv4 Link-Local-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.
      - `includeIPv6LoopbackAddrs`— (Optional) Sie können diese Option aktivieren, um IPv6 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. Sie müssen `includeIPv4Addrs` auf `false` und auf einstellen`includeIPv6Addrs`, `true` um diese Option verwenden zu können. Sie benötigen IP Detector v2.2.0 oder höher, um diese Option verwenden zu können.
      - `includeIPv6LinkLocalAddrs`— (Optional) Sie können diese Option aktivieren, um IPv6 Link-Local-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. Sie müssen `includeIPv4Addrs` auf `false` und auf einstellen, `includeIPv6Addrs` um diese Option verwenden `true` zu können. Sie benötigen IP Detector v2.2.0 oder höher, um diese Option verwenden zu können.

- `includeIPv4Addrs`— (Optional) Die Standardeinstellung ist auf true gesetzt. Sie können diese Option aktivieren, um IPv4 Adressen zu veröffentlichen, die auf dem Core-Gerät gefunden wurden. Sie benötigen IP Detector v2.2.0 oder höher, um diese Option verwenden zu können.
- `includeIPv6Addrs`— (Optional) Sie können diese Option aktivieren, um IPv6 Adressen zu veröffentlichen, die auf dem Core-Gerät gefunden wurden. Stellen Sie `includeIPv4Addrs` auf `false` ein, um diese Option zu verwenden. Sie benötigen IP Detector v2.2.0 oder höher, um diese Option verwenden zu können.

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.

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.

   Es kann bis zu einer Minute dauern, bis die Bereitstellung abgeschlossen ist.

Stellen Sie die IP-Detektorkomponente bereit (AWS CLI)

Um die IP-Detektorkomponente bereitzustellen, erstellen Sie ein Bereitstellungsdokument, das das `components` Objekt enthält`aws.greengrass.clientdevices.IPDetector`, und geben Sie das Konfigurationsupdate für die Komponente an. Folgen Sie den Anweisungen unter[Erstellen von Bereitstellungen](), um eine neue Einrichtung zu erstellen oder eine bestehende Einrichtung zu überarbeiten.

Sie können bei der Erstellung 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, 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.
- `includeIPv4LinkLocalAddrs`— (Optional) Sie können diese Option aktivieren, um IPv4 Link-Local-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.
- `includeIPv6LoopbackAddrs`— (Optional) Sie können diese Option aktivieren, um IPv6 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. Sie müssen `includeIPv4Addrs` auf `false` und auf einstellen`includeIPv6Addrs`, `true` um diese Option verwenden zu können. Sie benötigen IP Detector v2.2.0 oder höher, um diese Option verwenden zu können.
- `includeIPv6LinkLocalAddrs`— (Optional) Sie können diese Option aktivieren, um IPv6 Link-Local-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. Sie müssen `includeIPv4Addrs` auf `false` und auf einstellen, `includeIPv6Addrs` um diese Option verwenden `true` zu können. Sie benötigen IP Detector v2.2.0 oder höher, um diese Option verwenden zu können.
- `includeIPv4Addrs`— (Optional) Die Standardeinstellung ist auf true gesetzt. Sie können diese Option aktivieren, um IPv4 Adressen zu veröffentlichen, die auf dem Core-Gerät gefunden wurden. Sie benötigen IP Detector v2.2.0 oder höher, um diese Option verwenden zu können.
- `includeIPv6Addrs`— (Optional) Sie können diese Option aktivieren, um IPv6 Adressen zu veröffentlichen, die auf dem Core-Gerät gefunden wurden. Stellen Sie `includeIPv4Addrs` auf `false` ein, um diese Option zu verwenden. Sie benötigen IP Detector v2.2.0 oder höher, um diese Option verwenden zu können.

Im folgenden Beispieldokument zur teilweisen Bereitstellung wird angegeben, dass Port 8883 als MQTT-Broker-Port gemeldet werden soll.

```
{
  ...,
  "components": {
    ...,
    "aws.greengrass.clientdevices.IPDetector": {
      "componentVersion": "2.1.1",
      "configurationUpdate": {
        "merge": "{\"defaultPort\":\"8883\",}"
      }
    }
  }
}
```

Manuelles Verwalten von Endpunkten

Sie können MQTT-Broker-Endpunkte für Kerngeräte manuell verwalten.

Jeder MQTT-Broker-Endpunkt enthält die folgenden Informationen:

Endpunkt () `HostAddress`

Eine IP-Adresse oder DNS-Adresse, über die Client-Geräte eine Verbindung zu einem MQTT-Broker auf dem Kerngerät herstellen können.

Port (`PortNumber`)

Der Port, an dem der MQTT-Broker auf dem Kerngerät arbeitet.

Sie können diesen Port in der Moquette MQTT-Broker-Komponente konfigurieren, die standardmäßig Port 8883 verwendet.

Metadaten () `Metadata`

Zusätzliche Metadaten, die den Client-Geräten zur Verfügung gestellt werden, die eine Verbindung zu diesem Endpunkt herstellen.

Themen

- Endgeräte verwalten (Konsole)
- Endpunkte verwalten ()AWS CLI

- Endpunkte verwalten (API)

Endgeräte verwalten (Konsole)

Sie können die AWS IoT Greengrass Konsole verwenden, um Endgeräte für ein Kerngerät anzuzeigen, zu aktualisieren und zu entfernen.

Um Endpunkte für ein Kerngerät (Konsole) zu verwalten

1. Navigieren Sie zur AWS IoT Greengrass -Konsole.
2. Wählen Sie Core-Geräte.
3. Wählen Sie das zu verwaltende Kerngerät aus.
4. Wählen Sie auf der Detailseite des Kerngeräts die Registerkarte Client-Geräte aus.
5. Im Abschnitt MQTT-Broker-Endpunkte können Sie die MQTT-Broker-Endpunkte des Kerngeräts sehen. Wählen Sie Endpunkte verwalten aus.
6. Fügen Sie im Modal „Endpunkte verwalten“ MQTT-Broker-Endpunkte für das Kerngerät hinzu oder entfernen Sie sie.
7. Wählen Sie Aktualisieren.

Endpunkte verwalten ()AWS CLI

Sie können das AWS Command Line Interface (AWS CLI) verwenden, um Endpunkte für ein Kerngerät zu verwalten.

> **Note**
>
> Da die Unterstützung von Client-Geräten in abwärtskompatibel AWS IoT Greengrass V2 ist AWS IoT Greengrass V1, können Sie AWS IoT Greengrass V1 API-Operationen verwenden AWS IoT Greengrass V2 , um die Endpunkte von Kerngeräten zu verwalten.

Um Endpunkte für ein Kerngerät abzurufen ()AWS CLI

- Verwenden Sie einen der folgenden Befehle:
  - greengrassv2: get-connectivity-info
  - grünes Gras: get-connectivity-info

Um Endpunkte für ein Kerngerät zu aktualisieren ()AWS CLI

- Verwenden Sie einen der folgenden Befehle:
  - greengrassv2: update-connectivity-info
  - grünes Gras: update-connectivity-info

Endpunkte verwalten (API)

Sie können die AWS API verwenden, um Endpunkte für ein Kerngerät zu verwalten.

> **Note**
>
> Da die Unterstützung von Client-Geräten in abwärtskompatibel AWS IoT Greengrass V2 ist AWS IoT Greengrass V1, können Sie unsere AWS IoT Greengrass V1 API-Operationen verwenden AWS IoT Greengrass V2 , um die Endpunkte von Kerngeräten zu verwalten.

Um Endpunkte für ein Kerngerät (API) abzurufen AWS

- Verwenden Sie eine der folgenden Operationen:
  - V2: GetConnectivityInfo
  - V1: GetConnectivityInfo

Um Endpunkte für ein Kerngerät (AWS API) zu aktualisieren

- Verwenden Sie eine der folgenden Operationen:
  - V2: UpdateConnectivityInfo
  - V1: UpdateConnectivityInfo

Wählen Sie einen MQTT-Broker

AWS IoT Greengrass bietet Optionen, mit denen Sie auswählen können, welcher lokale MQTT-Broker auf Ihren Kerngeräten ausgeführt werden soll. Client-Geräte stellen eine Verbindung mit dem MQTT-Broker her, der auf einem Core-Gerät läuft. Wählen Sie also einen MQTT-Broker, der mit den Client-Geräten kompatibel ist, die Sie verbinden möchten.

> **Note**
>
> Wir empfehlen, dass Sie nur eine MQTT-Broker-Komponente bereitstellen. Die Komponenten MQTT-Bridge und IP-Detektor funktionieren jeweils nur mit einer MQTT-Broker-Komponente. Wenn Sie mehrere MQTT-Broker-Komponenten bereitstellen, müssen Sie sie so konfigurieren, dass sie unterschiedliche Ports verwenden.

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 einfachen MQTT-Broker, der dem MQTT 3.1.1-Standard entspricht. Sowohl der AWS IoT Core MQTT-Broker als auch der MQTT 3.1.1-Standard entsprechen, sodass Sie diese Funktionen verwenden können, um eine Anwendung zu erstellen, die MQTT 3.1.1 auf Ihren Geräten und dem verwendet. AWS IoT Device SDK AWS Cloud
- MQTT 5-Broker (EMQX) — `aws.greengrass.clientdevices.mqtt.EMQX`

  Wählen Sie diese Option, um MQTT 5-Funktionen für die Kommunikation zwischen Kerngeräten und Client-Geräten zu verwenden. Diese Komponente verwendet mehr Ressourcen als der Moquette MQTT 3.1.1-Broker und benötigt auf Linux-Kerngeräten 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, sodass die Client-Geräte weiterhin eine Verbindung herstellen und wie gewohnt arbeiten können.

- Implementieren Sie einen benutzerdefinierten Broker

  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 stellt ein Komponenten-SDK bereit, mit dem Sie Client-Geräte authentifizieren und autorisieren können. Weitere Informationen erhalten Sie unter Verwenden Sie den AWS IoT Device SDK , um mit dem Greengrass-Kern und anderen Komponenten zu kommunizieren und AWS IoT Core und Client-Geräte authentifizieren und autorisieren.

Client-Geräte über einen AWS IoT Greengrass MQTT-Broker mit einem Core-Gerät verbinden

Wenn Sie einen MQTT-Broker auf Ihrem AWS IoT Greengrass Core-Gerät verwenden, verwendet das Gerät eine für das Gerät einzigartige Zertifizierungsstelle (Core Device Certificate Authority, CA), um dem Broker ein Zertifikat auszustellen, das gegenseitige TLS-Verbindungen mit Clients herstellt.

AWS IoT Greengrass generiert automatisch eine Zertifizierungsstelle für das Kerngerät, oder Sie können Ihre eigene angeben. Die CA für das Kerngerät ist registriert AWS IoT Greengrass , wenn die Authentifizierung auf Client-Geräten Komponente angeschlossen wird. Die automatisch generierte Zertifizierungsstelle für das Kerngerät ist persistent. Das Gerät verwendet weiterhin dieselbe CA, solange die Authentifizierungskomponente für das Client-Gerät konfiguriert ist.

Wenn der MQTT-Broker gestartet wird, fordert er ein Zertifikat an. Die Authentifizierungskomponente für das Clientgerät stellt unter Verwendung der Zertifizierungsstelle des Kerngeräts ein X.509-Zertifikat 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 Zertifikatsrotation auf dem lokalen MQTT-Broker.

Um einen Client mit dem MQTT-Broker zu verbinden, benötigen Sie Folgendes:

- Das Client-Gerät muss über die AWS 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 Domainname (FQDN) oder die IP-Adresse des Kerngeräts müssen im Brokerzertifikat enthalten sein, das von der Zertifizierungsstelle des Kerngeräts ausgestellt wurde. Sie stellen dies sicher, indem Sie die IP-Detektor Komponente verwenden oder die IP-Adresse manuell konfigurieren. Weitere Informationen finden Sie unter Endpunkte von Kerngeräten verwalten.
- Die Authentifizierungskomponente für das Client-Gerät muss dem Client-Gerät die Erlaubnis erteilen, 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 erkennen, können Sie eine Zertifizierungsstelle (CA) für das Kerngerät bereitstellen. Ihr Greengrass-Core-Gerät verwendet die Core-Geräte-CA, um Zertifikate für Ihren MQTT-Broker auszustellen. Sobald Sie das

Core-Gerät konfiguriert und Ihr Client-Gerät mit seiner CA ausgestattet haben, können Ihre Client-Geräte eine Verbindung zum Endpoint herstellen und den TLS-Handshake mithilfe der zentralen Geräte-CA (selbst bereitgestellte oder automatisch generierte CA) verifizieren.

Um die Authentifizierung auf Client-Geräten Komponente so zu konfigurieren, dass sie die CA Ihres Kerngeräts verwendet, legen Sie bei der Bereitstellung der Komponente den `certificateAuthority` Konfigurationsparameter fest. Sie müssen bei der Konfiguration die folgenden Details angeben:

- Der Speicherort eines CA-Zertifikats für das Kerngerät.
- Der private Schlüssel des CA-Zertifikats für das Kerngerät.
- (Optional) Die Zertifikatskette zum Stammzertifikat, wenn es sich bei der Zertifizierungsstelle des Kerngeräts um eine Zwischenzertifizierungsstelle handelt.

Wenn Sie eine Zertifizierungsstelle für das Kerngerät angeben, AWS IoT Greengrass registriert diese Zertifizierungsstelle in der Cloud.

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 wird. 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 Client-Gerät 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 Ihrem AWS IoT Greengrass Core-Gerät zu verbinden:

1. Erstellen Sie mithilfe der Stammzertifizierungsstelle Ihrer Organisation eine Zwischenzertifizierungsstelle (CA) für das Greengrass-Core-Gerät. Aus Sicherheitsgründen empfehlen wir Ihnen, eine Zwischenzertifizierungsstelle zu verwenden.
2. Stellen Sie das Zwischenzertifikat, den privaten Schlüssel und die Zertifikatskette für Ihre Stammzertifizierungsstelle auf dem Greengrass-Core-Gerät bereit. Weitere Informationen finden Sie unter Authentifizierung auf Client-Geräten. Die Zwischen-CA wird zur Core-Device-CA für das Greengrass-Core-Gerät, und das Gerät registriert die CA bei AWS IoT Greengrass.
3. Registrieren Sie das Client-Gerät als eine beliebige AWS IoT Sache. Weitere Informationen finden Sie im AWS IoT Core Entwicklerhandbuch unter Ein Ding-Objekt erstellen. 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 den öffentlichen Schlüsselbund verwenden, um eine Verbindung zum Greengrass-Core-Gerät herzustellen. Ihre Software ist dafür verantwortlich, die Endpunkte der Kerngeräte zu finden. Sie können den Endpunkt für das Kerngerät manuell festlegen. Weitere Informationen finden Sie unter Manuelles Verwalten von Endpunkten.

Testen Sie die Kommunikation mit Client-Geräten

Client-Geräte können das verwenden, AWS IoT Device SDK um ein Kerngerät zu erkennen, eine Verbindung herzustellen und mit ihm zu kommunizieren. Sie können den Greengrass Discovery Client in verwenden, AWS IoT Device SDK um die Greengrass Discovery API zu verwenden, die Informationen über Kerngeräte zurückgibt, mit denen ein Client-Gerät eine Verbindung herstellen kann. Die API-Antwort umfasst MQTT-Broker-Endpunkte für die Verbindung und Zertifikate zur Überprüfung der Identität der einzelnen Kerngeräte. Anschließend kann das Client-Gerät jeden Endpunkt ausprobieren, bis es erfolgreich eine Verbindung zu einem Kerngerät herstellt.

Client-Geräte können nur Kerngeräte 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 Client-Geräte zuordnen.

Die Greengrass Discovery API gibt die von Ihnen angegebenen MQTT-Broker-Endpunkte für das Kerngerät zurück. 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 Endpunkte von Kerngeräten verwalten.

> **Note**
>
> Um die Greengrass Discovery API verwenden zu können, muss ein Client-Gerät über die `greengrass:Discover` entsprechende Genehmigung verfügen. Weitere Informationen finden Sie unter Minimale AWS IoT Richtlinie für Client-Geräte.

Das AWS IoT Device SDK ist in mehreren Programmiersprachen verfügbar. Weitere Informationen finden Sie unter AWS IoT Gerät SDKs im AWS IoT Core Entwicklerhandbuch.

Themen

- Kommunikation testen (Python)
- Testen Sie die Kommunikation (C++)
- Testen Sie die Kommunikation () JavaScript
- Testen Sie die Kommunikation (Java)

Kommunikation testen (Python)

In diesem Abschnitt verwenden Sie das Greengrass Discovery-Beispiel in AWS IoT Device SDK Version 2 für Python, um die Kommunikation zwischen einem Client-Gerät und einem Core-Gerät zu testen.

> **Important**
>
> Um AWS IoT Device SDK Version 2 für Python zu verwenden, muss auf einem Gerät Python 3.6 oder höher ausgeführt werden.

Um die Kommunikation zu testen (AWS IoT Device SDK v2 für Python)

1. Laden Sie die AWS IoT Device SDK Version v2 für Python herunter und installieren Sie sie auf dem AWS IoT Ding, 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 for Python, um es herunterzuladen.

```
git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
```

b. Installieren Sie die AWS IoT Device SDK Version v2 für Python.

```
python3 -m pip install --user ./aws-iot-device-sdk-python-v2
```

2. Wechseln Sie in den Samples-Ordner in der AWS IoT Device SDK Version v2 für Python.

```
cd aws-iot-device-sdk-python-v2/samples/greengrass
```

3. Führen Sie die Greengrass-Discovery-Beispielanwendung aus. Diese Anwendung erwartet Argumente, die den Namen des Dings auf dem Client-Gerät, das zu verwendende MQTT-Thema und die zu verwendende 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.

   - *`MyClientDevice1`*Ersetzen Sie es durch den Dingnamen des Client-Geräts.
   - *`~/certs/AmazonRootCA1.pem`*Ersetzen Sie durch den Pfad zum Amazon-Root-CA-Zertifikat auf dem Client-Gerät.
   - *`~/certs/device.pem.crt`*Ersetzen Sie durch den Pfad zum Gerätezertifikat auf dem Client-Gerät.
   - *`~/certs/private.pem.key`*Ersetzen Sie durch den Pfad zur privaten Schlüsseldatei auf dem Client-Gerät.
   - *`us-east-1`*Ersetzen Sie 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 zehnmal und trennt die Verbindung. Außerdem abonniert sie dasselbe Thema, in dem sie Nachrichten veröffentlicht. Wenn die Ausgabe anzeigt, dass die Anwendung MQTT-Nachrichten zu diesem Thema empfangen 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 Behebung von Problemen mit der Greengrass-Erkennung.

Sie können auch die Greengrass-Protokolle auf dem Kerngerät einsehen, um zu überprüfen, ob das Client-Gerät erfolgreich eine Verbindung herstellt und Nachrichten sendet. Weitere Informationen finden Sie unter AWS IoT Greengrass Protokolle überwachen.

Testen Sie die Kommunikation (C++)

In diesem Abschnitt verwenden Sie das Greengrass Discovery-Beispiel in AWS IoT Device SDK Version 2 für C++, um die Kommunikation zwischen einem Client-Gerät und einem Core-Gerät zu testen.

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 später
- Einer der folgenden Compiler:
  - GCC 4.8 oder höher
  - Clang 3.9 oder höher
  - MSVC 2015 oder später

Um die Kommunikation zu testen (AWS IoT Device SDK v2 für C++)

1. Laden Sie die AWS IoT Device SDK Version v2 für C++ herunter und erstellen Sie sie zu dem AWS IoT Ding, das Sie als Client-Gerät verbinden möchten.

   Gehen Sie auf dem Client-Gerät wie folgt vor:

   a. Erstellen Sie einen Ordner für den Arbeitsbereich 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 der AWS IoT Device SDK Version 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 die AWS IoT Device SDK Version 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 AWS IoT Device SDK Version 2 für C++. Gehen Sie wie folgt vor:

a. Wechseln Sie in der AWS IoT Device SDK Version v2 für C++ zum Greengrass Discovery-Beispielordner.

```
cd ../aws-iot-device-sdk-cpp-v2/samples/greengrass/basic_discovery
```

b. Erstellen Sie einen Ordner für die Greengrass Discovery-Beispiel-Build-Ausgabe und wechseln Sie zu diesem.

```
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-Discovery-Beispielanwendung aus. Diese Anwendung erwartet Argumente, die den Namen der Sache auf dem Client-Gerät, das zu verwendende MQTT-Thema und die Zertifikate angeben, die die Verbindung authentifizieren und sichern. Das folgende Beispiel abonniert das `clients/`*`MyClientDevice1`*`/hello/world` Thema und veröffentlicht eine Nachricht, die Sie in der Befehlszeile zu demselben Thema eingeben.

- *MyClientDevice1*Ersetzen Sie es durch den Ding-Namen des Client-Geräts.
- *~/certs/AmazonRootCA1.pem*Ersetzen Sie durch den Pfad zum Amazon-Root-CA-Zertifikat auf dem Client-Gerät.
- *~/certs/device.pem.crt*Ersetzen Sie durch den Pfad zum Gerätezertifikat auf dem Client-Gerät.
- *~/certs/private.pem.key*Ersetzen Sie durch den Pfad zur privaten Schlüsseldatei auf dem Client-Gerät.
- *us-east-1*Ersetzen Sie 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 Behebung von Problemen mit der Greengrass-Erkennung.

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 anzeigt, dass die Anwendung die MQTT-Nachricht zu dem 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 Kerngerät einsehen, um zu überprüfen, ob das Client-Gerät erfolgreich eine Verbindung herstellt und Nachrichten sendet. Weitere Informationen finden Sie unter AWS IoT Greengrass Protokolle überwachen.

Testen Sie die Kommunikation () JavaScript

In diesem Abschnitt verwenden Sie das Greengrass Discovery-Beispiel in AWS IoT Device SDK Version 2, JavaScript um die Kommunikation zwischen einem Client-Gerät und einem Core-Gerät zu testen.

> **Important**
>
> Um AWS IoT Device SDK Version 2 für verwenden zu können JavaScript, muss auf einem Gerät Node v10.0 oder höher ausgeführt werden.

Um die Kommunikation zu testen (AWS IoT Device SDK v2 für JavaScript)

1. Laden Sie AWS IoT Device SDK Version 2 für JavaScript das Gerät herunter und installieren Sie AWS IoT es, um eine Verbindung als Client-Gerät herzustellen.

   Gehen Sie auf dem Client-Gerät wie folgt vor:

   a. Klonen Sie AWS IoT Device SDK die Version 2 für das JavaScript Repository, um sie herunterzuladen.

   ```
   git clone https://github.com/aws/aws-iot-device-sdk-js-v2.git
   ```

   b. Installieren Sie die AWS IoT Device SDK Version v2 für JavaScript.

```
cd aws-iot-device-sdk-js-v2
npm install
```

2. Wechseln Sie in der AWS IoT Device SDK Version 2 für JavaScript zum Greengrass Discovery-Beispielordner.

```
cd samples/node/greengrass/basic_discovery
```

3. Installieren Sie die Greengrass Discovery-Beispielanwendung.

```
npm install
```

4. Führen Sie die Greengrass-Discovery-Beispielanwendung aus. Diese Anwendung erwartet Argumente, die den Namen des Dings auf dem Client-Gerät, das zu verwendende MQTT-Thema und die zu verwendende 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.

   - *MyClientDevice1*Ersetzen Sie es durch den Dingnamen des Client-Geräts.
   - *~/certs/AmazonRootCA1.pem*Ersetzen Sie durch den Pfad zum Amazon-Root-CA-Zertifikat auf dem Client-Gerät.
   - *~/certs/device.pem.crt*Ersetzen Sie durch den Pfad zum Gerätezertifikat auf dem Client-Gerät.
   - *~/certs/private.pem.key*Ersetzen Sie durch den Pfad zur privaten Schlüsseldatei auf dem Client-Gerät.
   - *us-east-1*Ersetzen Sie 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 zehnmal und trennt die Verbindung. Außerdem abonniert sie dasselbe Thema, in dem sie Nachrichten veröffentlicht. Wenn die Ausgabe anzeigt, dass die Anwendung MQTT-Nachrichten zu diesem Thema empfangen 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 Behebung von Problemen mit der Greengrass-Erkennung.

Sie können auch die Greengrass-Protokolle auf dem Kerngerät einsehen, um zu überprüfen, ob das Client-Gerät erfolgreich eine Verbindung herstellt und Nachrichten sendet. Weitere Informationen finden Sie unter AWS IoT Greengrass Protokolle überwachen.

Testen Sie die Kommunikation (Java)

In diesem Abschnitt verwenden Sie das Greengrass Discovery-Beispiel in AWS IoT Device SDK Version 2 für Java, um die Kommunikation zwischen einem Client-Gerät und einem Core-Gerät zu testen.

> **Important**
>
> Um die AWS IoT Device SDK Version 2 für Java zu erstellen, muss ein Gerät über die folgenden Tools verfügen:
>
> - Java 8 oder höher, mit `JAVA_HOME` Verweis auf den Java-Ordner.
> - Apache Maven

Um die Kommunikation zu testen (AWS IoT Device SDK v2 für Java)

1. Laden Sie die AWS IoT Device SDK Version v2 für Java herunter und erstellen Sie sie auf AWS IoT dem Gerät, 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 for 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 die AWS IoT Device SDK Version 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-Discovery-Beispielanwendung aus. Diese Anwendung erwartet Argumente, die den Namen der Sache auf dem Client-Gerät, das zu verwendende MQTT-Thema und die Zertifikate angeben, die die Verbindung authentifizieren und sichern. Das folgende Beispiel abonniert das `clients/`*`MyClientDevice1`*`/hello/world` Thema und veröffentlicht eine Nachricht, die Sie in der Befehlszeile zu demselben Thema eingeben.

   - Ersetzen Sie beide Instanzen von *`MyClientDevice1`* durch den Dingnamen des Client-Geräts.
   - *`$HOME/certs/AmazonRootCA1.pem`*Ersetzen Sie durch den Pfad zum Amazon-Root-CA-Zertifikat auf dem Client-Gerät.
   - *`$HOME/certs/device.pem.crt`*Ersetzen Sie durch den Pfad zum Gerätezertifikat auf dem Client-Gerät.
   - *`$HOME/certs/private.pem.key`*Ersetzen Sie durch den Pfad zur privaten Schlüsseldatei auf dem Client-Gerät.
   - *`us-east-1`*Ersetzen Sie durch den AWS-Region Ort, an dem Ihr Client-Gerät und Ihr Core-Gerät betrieben 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/Discovery \
  -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 Behebung von Problemen mit der Greengrass-Erkennung.

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 anzeigt, dass die Anwendung die MQTT-Nachricht zu dem 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 Kerngerät einsehen, um zu überprüfen, ob das Client-Gerät erfolgreich eine Verbindung herstellt und Nachrichten sendet. Weitere Informationen finden Sie unter AWS IoT Greengrass Protokolle überwachen.

Greengrass-Entdeckungs-API RESTful

AWS IoT Greengrass stellt die `Discover` API-Operation bereit, mit der Client-Geräte Greengrass-Core-Geräte identifizieren können, mit denen sie sich verbinden können. Client-Geräte verwenden diesen Datenebenenvorgang, um Informationen abzurufen, die für die Verbindung mit Greengrass-Core-Geräten erforderlich sind, wo Sie sie dem BatchAssociateClientDeviceWithCoreDeviceAPI-Vorgang zuordnen. Wenn ein Client-Gerät online geht, kann es eine Verbindung zum AWS IoT Greengrass Cloud-Dienst herstellen und mithilfe der Discovery-API Folgendes ermitteln:

- Die IP-Adresse und der Port für jedes zugehörige Greengrass-Core-Gerät.

- Das CA-Zertifikat für das Kerngerät, mit dem Client-Geräte das Greengrass-Core-Gerät authentifizieren können.

> **Note**
>
> Client-Geräte können auch den Discovery-Client im verwenden AWS IoT Device SDK , um Konnektivitätsinformationen für Greengrass-Kerngeräte zu ermitteln. Der Discovery-Client verwendet die Discovery-API. Weitere Informationen finden Sie hier:
>
> - Testen Sie die Kommunikation mit Client-Geräten
> - Greengrass Discovery RESTful API im AWS IoT Greengrass Version 1 Developer Guide.

Um diesen API-Vorgang zu verwenden, senden Sie HTTP-Anfragen an die Discovery-API auf dem Greengrass-Datenebenen-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 für die AWS IoT Greengrass Discovery-API unterstützten Endpunkte und Endpunkte finden Sie unter AWS IoT Greengrass V2 Endpunkte und Kontingente in der. Allgemeine AWS-Referenz Dieser API-Vorgang ist nur auf dem Endpunkt der Greengrass-Datenebene verfügbar. Der Endpunkt der Steuerungsebene, den Sie zur Verwaltung von Komponenten und Bereitstellungen verwenden, unterscheidet sich vom Endpunkt der Datenebene.

> **Note**
>
> Die Discovery-API ist dieselbe für AWS IoT Greengrass V1 und AWS IoT Greengrass V2. Wenn Sie über Client-Geräte verfügen, die eine Verbindung zu einem AWS IoT Greengrass V1 Core 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 Developer Guide.

Themen

- Discovery-Authentifizierung und Autorisierung
- Anforderung

- Antwort
- Testen Sie die Discovery-API mit cURL

Discovery-Authentifizierung 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 Developer Guide.

Ein Client-Gerät muss außerdem über die Berechtigung verfügen, die `greengrass:Discover` Aktion auszuführen. Mit der folgenden AWS IoT Beispielrichtlinie kann ein AWS IoT Objekt mit `MyClientDevice1` dem Namen selbstständig ausgeführt `Discover` werden.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "greengrass:Discover",
      "Resource": [
        "arn:aws:iot:us-west-2:123456789012:thing/MyClientDevice1"
      ]
    }
  ]
}
```

Important

Ding-Richtlinienvariablen (`iot:Connection.Thing.*`) werden in AWS IoT Richtlinien für Kerngeräte oder Greengrass-Datenebenenoperationen nicht unterstützt. Stattdessen können Sie einen Platzhalter verwenden, der für mehrere Geräte mit ähnlichen Namen steht. Sie können beispielsweise angeben, ob `MyGreengrassDevice*` ein Treffer gefunden werden soll `MyGreengrassDevice1MyGreengrassDevice2`, usw.

Weitere Informationen finden Sie unter AWS IoT Core Richtlinien im AWS IoT Core Entwicklerhandbuch.

Anforderung

Die Anfrage enthält die Standard-HTTP-Header und wird an den Greengrass Discovery-Endpunkt gesendet, wie in den folgenden Beispielen gezeigt.

Die Portnummer hängt davon ab, ob das Kerngerät so konfiguriert ist, dass es HTTPS-Verkehr über Port 8443 oder Port 443 sendet. 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-Root-CA-Zertifikaten arbeitet. 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` sie als `ProtocolName` in 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 das Discover-Antwortdokument.

> **Note**
>
> Da die Antwort dieselbe Discovery-API AWS IoT Greengrass V2 verwendet wie AWS IoT Greengrass V1, organisiert sie Informationen nach AWS IoT Greengrass V1 Konzepten wie Greengrass-Gruppen. Die Antwort enthält eine Liste von Greengrass-Gruppen. In befindet AWS IoT Greengrass V2 sich jedes Kerngerät in einer eigenen Gruppe, wobei die Gruppe nur dieses Kerngerät und seine Konnektivitätsinformationen enthält.

Beispieldokumente für Discovery-Antwort

Das folgende Dokument zeigt die Antwort für ein Client-Gerät, das einem Greengrass-Core-Gerät zugeordnet ist. Das Kerngerät hat 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 Kerngeräten zugeordnet ist. Die Kerngeräte 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 Sie die Discovery-API mit cURL

Wenn Sie die Discovery-API cURL installiert haben, können Sie sie testen. Das folgende Beispiel spezifiziert die Zertifikate eines Client-Geräts zur Authentifizierung einer Anfrage an den Greengrass Discovery API-Endpunkt.

```
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 Anfrage erfolgreich ist, gibt dieser Befehl eine Antwort aus, die dem folgenden Beispiel ähnelt.

```
{
  "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 Problemen mit der Greengrass-Erkennung.

MQTT-Nachrichten zwischen Client-Geräten weiterleiten und AWS IoT Core

Sie können MQTT-Nachrichten und andere Daten zwischen Client-Geräten und weiterleiten. AWS IoT Core Client-Geräte stellen eine Verbindung zur MQTT-Broker-Komponente her, die auf dem Kerngerät ausgeführt wird. Standardmäßig leiten Kerngeräte keine MQTT-Nachrichten oder Daten zwischen Client-Geräten und weiter. AWS IoT Core Client-Geräte können standardmäßig nur miteinander über MQTT kommunizieren.

Um MQTT-Nachrichten zwischen Client-Geräten weiterzuleiten und AWS IoT Core, konfigurieren Sie die MQTT-Bridge-Komponente wie folgt:

- Nachrichten von Client-Geräten weiterleiten an. AWS IoT Core
- Leitet Nachrichten von AWS IoT Core zu Client-Geräten weiter.

> **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

- Konfigurieren und implementieren Sie die MQTT-Bridge-Komponente
- MQTT-Nachrichten weiterleiten

Konfigurieren und implementieren Sie die 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 weiterzuleiten und AWS 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 Kerngerät oder einer Gruppe von Kerngeräten bereitzustellen, erstellen Sie eine Bereitstellung, die die Komponente enthält. `aws.greengrass.clientdevices.mqtt.Bridge` Geben Sie die Themenzuordnungen in der Konfiguration der MQTT-Bridge-Komponente in der Bereitstellung an. `mqttTopicMapping`

Das folgende Beispiel definiert eine Bereitstellung, die die MQTT-Bridge-Komponente so konfiguriert, dass sie Nachrichten zu Themen, die dem `clients/+/hello/world` Themenfilter entsprechen, von Client-Geräten weiterleitet. AWS 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\"}}}"
      }
    }
    ...
  }
}
```

MQTT-Nachrichten weiterleiten

Um MQTT-Nachrichten zwischen Client-Geräten weiterzuleiten und AWS IoT Core, konfigurieren und implementieren Sie die MQTT Bridge-Komponente 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 spezifiziert die Weiterleitung von Nachrichten zu Themen, die dem `clients/+/hello/world/event` Themenfilter entsprechen, von Client-Geräten an. AWS 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 zu Client-Geräten

Die folgende Konfiguration der MQTT-Bridge-Komponente spezifiziert die Weiterleitung von Nachrichten zu Themen, die dem `clients/+/hello/world/event/response` Themenfilter entsprechen, von AWS IoT Core an Client-Geräte.

```
{
  "mqttTopicMapping": {
    "HelloWorldEventConfirmation": {
      "topic": "clients/+/hello/world/event/response",
      "source": "IotCore",
      "target": "LocalMqtt"
    }
  }
}
```

Interagieren Sie mit Client-Geräten in Komponenten

Sie können benutzerdefinierte Greengrass-Komponenten entwickeln, die mit Client-Geräten 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-Broker-Komponente, die auf dem Kerngerät ausgeführt wird, eine Verbindung zu einem Kerngerät 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 Kerngerät zu kommunizieren. Um mit Client-Geräten in Greengrass-Komponenten zu kommunizieren, konfigurieren Sie die MQTT-Bridge-Komponente wie folgt:

- Leiten Sie MQTT-Nachrichten von Client-Geräten an Local Publish/Subscribe weiter.
- Leiten Sie MQTT-Nachrichten von Local Publish/Subscribe an Client-Geräte weiter.

Sie können auch mit Schatten von Client-Geräten in Greengrass-Komponenten interagieren. Weitere Informationen finden Sie unter Interagieren Sie mit Schatten auf Client-Geräten und synchronisieren Sie sie.

Themen

- Konfigurieren und implementieren Sie die MQTT-Bridge-Komponente
- Empfangen Sie MQTT-Nachrichten von Client-Geräten
- Senden Sie MQTT-Nachrichten an Client-Geräte

Konfigurieren und implementieren Sie die 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 Kerngerät oder einer Gruppe von Kerngeräten bereitzustellen, erstellen Sie eine Bereitstellung, die die Komponente enthält. `aws.greengrass.clientdevices.mqtt.Bridge` Geben Sie die Themenzuordnungen in der Konfiguration der MQTT-Bridge-Komponente in der Bereitstellung an. `mqttTopicMapping`

Das folgende Beispiel definiert eine Bereitstellung, die die MQTT-Bridge-Komponente so konfiguriert, dass sie das `clients/MyClientDevice1/hello/world` Thema von den Client-Geräten an den lokalen Publish/Subscribe-Broker weiterleitet. 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 Quellbroker lokal veröffentlichen/abonnieren ist. Weitere Informationen finden Sie unter Konfiguration der MQTT-Bridge-Komponente.

Empfangen Sie MQTT-Nachrichten von Client-Geräten

Sie können die lokalen Themen zum Veröffentlichen und Abonnieren abonnieren, die Sie für die MQTT-Bridge-Komponente für den Empfang von Nachrichten von Client-Geräten konfigurieren.

Um MQTT-Nachrichten von Client-Geräten in benutzerdefinierten Komponenten zu empfangen

1. Konfigurieren und implementieren Sie die MQTT-Bridge-Komponente, um Nachrichten von einem MQTT-Thema, bei dem Client-Geräte veröffentlichen, an ein lokales Veröffentlichungs-/Abonnement-Thema weiterzuleiten.
2. Verwenden Sie die lokale Publish/Subscribe-IPC-Schnittstelle, um das Thema zu abonnieren, bei dem die MQTT-Bridge Nachrichten weiterleitet. Weitere Informationen erhalten Sie unter Lokale Nachrichten veröffentlichen/abonnieren und SubscribeToTopic.

Das Tutorial Client-Geräte Connect 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 Sie eine Komponente, die mit Client-Geräten kommuniziert.

Senden Sie MQTT-Nachrichten an Client-Geräte

Sie können in den lokalen Themen zum Veröffentlichen und Abonnieren veröffentlichen, die Sie für die MQTT-Bridge-Komponente zum Senden von Nachrichten an Client-Geräte konfigurieren.

Um MQTT-Nachrichten auf Client-Geräten in benutzerdefinierten Komponenten zu veröffentlichen

1. Konfigurieren und implementieren Sie die MQTT-Bridge-Komponente, um Nachrichten von einem lokalen Publish/Subscribe-Thema an ein MQTT-Thema weiterzuleiten, bei dem Client-Geräte abonnieren.
2. Verwenden Sie die lokale Publish/Subscribe-IPC-Schnittstelle, um zu dem Thema zu veröffentlichen, an das die MQTT-Bridge Nachrichten weiterleitet. Weitere Informationen erhalten Sie unter Lokale Nachrichten veröffentlichen/abonnieren und PublishToTopic.

Interagieren Sie mit Schatten auf Client-Geräten und synchronisieren Sie sie

Sie können die Shadow-Manager-Komponente verwenden, um lokale Schatten, einschließlich Client-Geräteschatten, zu verwalten. Sie können den Schattenmanager für folgende Zwecke verwenden:

- Interagieren Sie mit Schatten von Client-Geräten in Greengrass-Komponenten.
- Synchronisieren Sie die Schatten von Client-Geräten mit AWS IoT Core.

> **Note**
>
> Die Shadow-Manager-Komponente synchronisiert Schatten AWS IoT Core standardmäßig nicht mit. Sie müssen die Shadow-Manager-Komponente konfigurieren, um anzugeben, welche Schatten auf Client-Geräten synchronisiert werden sollen.

Themen

- Voraussetzungen

- Aktivieren Sie den Shadow-Manager für die Kommunikation mit Client-Geräten
- Interagieren Sie mit Schatten auf Client-Geräten in Komponenten
- Synchronisieren Sie Schatten auf Client-Geräten mit AWS IoT Core

Voraussetzungen

Um mit Client-Geräteschatten zu interagieren und Client-Geräteschatten zu synchronisieren AWS IoT Core, muss ein Kerngerät die folgenden Anforderungen erfüllen:

- Auf dem Kerngerät müssen zusätzlich zu den Greengrass-Komponenten für die Unterstützung von Client-Geräten die folgenden Komponenten ausgeführt werden:
  - Greengrass Nucleus v2.6.0 oder höher
  - Shadow Manager v2.2.0 oder höher
  - MQTT Bridge v2.2.0 oder höher
- Die Authentifizierungskomponente für Client-Geräte muss so konfiguriert sein, dass Client-Geräte über Device-Shadow-Themen kommunizieren können.

Aktivieren Sie den Shadow-Manager für die Kommunikation mit Client-Geräten

Standardmäßig verwaltet die Shadow-Manager-Komponente keine Schatten auf Client-Geräten. Um diese Funktion zu aktivieren, müssen Sie MQTT-Nachrichten zwischen den Client-Geräten und der Shadow Manager-Komponente weiterleiten. Client-Geräte verwenden MQTT-Nachrichten, um Device Shadow-Updates zu empfangen und zu senden. Die Shadow-Manager-Komponente abonniert die lokale Publish/Subscribe-Schnittstelle von Greengrass, sodass Sie die MQTT-Bridge-Komponente so konfigurieren können, dass sie MQTT-Nachrichten zu Device-Shadow-Themen weiterleitet.

Die MQTT-Bridge-Komponente verwendet 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 Shadow-Themen für die Client-Geräteschatten an. Sie müssen die Bridge so konfigurieren, dass sie Nachrichten in beide Richtungen zwischen lokalem MQTT und lokalem Veröffentlichen/Abonnieren weiterleitet.

Um die MQTT-Bridge-Komponente auf einem Kerngerät oder einer Gruppe von Kerngeräten bereitzustellen, erstellen Sie eine Bereitstellung, die die Komponente enthält.

`aws.greengrass.clientdevices.mqtt.Bridge` Geben Sie die Themenzuordnungen in der Konfiguration der MQTT-Bridge-Komponente in der Bereitstellung an. `mqttTopicMapping`

Verwenden Sie die folgenden Beispiele, um die MQTT-Bridge-Komponente so zu konfigurieren, dass die Kommunikation zwischen Client-Geräten und der Shadow Manager-Komponente ermöglicht wird.

> **Note**
>
> Sie können diese Konfigurationsbeispiele in der AWS IoT Greengrass Konsole verwenden. Wenn Sie die AWS IoT Greengrass API verwenden, erfordert das `merge` Konfigurationsupdate ein serialisiertes JSON-Objekt, sodass Sie die folgenden JSON-Objekte in Zeichenketten serialisieren müssen. Weitere Informationen finden Sie unter Komponentenkonfigurationen aktualisieren.

Example Beispiel: Alle Schatten auf Client-Geräten verwalten

Das folgende Beispiel für die MQTT-Bridge-Konfiguration ermöglicht es dem Shadow Manager, alle Shadows 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: Schatten für ein Client-Gerät verwalten

Das folgende Beispiel für die MQTT-Bridge-Konfiguration ermöglicht es dem Shadow Manager, alle Shadows für ein Client-Gerät mit dem Namen `MyClientDevice` zu verwalten.

```
{
```

```
  "mqttTopicMapping": {
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/MyClientDevice/shadow/#",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/MyClientDevice/shadow/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Example Beispiel: Einen benannten Shadow für alle Client-Geräte verwalten

Das folgende Konfigurationsbeispiel für eine MQTT-Bridge ermöglicht es dem Shadow-Manager, einen `DeviceConfiguration` nach allen Client-Geräten benannten Shadow 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: Verwaltet die unbenannten Schatten aller Client-Geräte

Das folgende Beispiel für die MQTT-Bridge-Konfiguration ermöglicht es dem Shadow-Manager, unbenannte Schatten, aber nicht benannte 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 Sie mit Schatten auf Client-Geräten in Komponenten

Sie können benutzerdefinierte Komponenten entwickeln, die den lokalen Shadow-Dienst verwenden, um die lokalen Shadow-Dokumente von Client-Geräten zu lesen und zu ändern. Weitere Informationen finden Sie unter Interagiere mit Schatten in Komponenten.

Synchronisieren Sie Schatten auf Client-Geräten mit AWS IoT Core

Sie können die Shadow-Manager-Komponente so konfigurieren, dass Shadow-Status auf lokalen Client-Geräten mit synchronisiert AWS IoT Core werden. Weitere Informationen finden Sie unter Synchronisieren Sie lokale Geräteschatten mit AWS IoT Core.

IPv6 Für lokales Messaging verwenden

Sie können die IP-Detektorkomponente so konfigurieren, dass sie IPv6 zum Senden lokaler Nachrichten verwendet wird.

> **Note**
>
> Sie benötigen den IP-Detektor v2.2.0 oder höher, um lokale Nachrichten senden IPv6 zu können.

Sie können die IP-Detektorkomponente einsetzen, um Adressen zu erkennen und zu verwenden IPv6 . Sie müssen die Konfiguration der IP-Detektorkomponente aktualisieren, um sie IPv6 anstelle von zu verwenden IPv4. Weitere Informationen finden Sie unter Verwenden Sie den IP-Detektor, um Endpunkte automatisch zu verwalten.

Themen

- Konfigurieren Sie den zu verwendenden IP-Detektor IPv6

Konfigurieren Sie den zu verwendenden IP-Detektor IPv6

Wenn Sie über eine einfache Netzwerkkonfiguration verfügen, z. B. wenn sich die Client-Geräte im selben Netzwerk wie das Kerngerät befinden, können Sie die IP-Detektorkomponente IPv6 für lokales Messaging einsetzen.

Die IP-Detektorkomponente überschreibt Endpunkte, die Sie manuell einrichten.

> **Important**
>
> Die AWS IoT Richtlinie des Kerngeräts muss die `greengrass:UpdateConnectivityInfo` Erlaubnis zur Verwendung der IP-Detektorkomponente zulassen. Weitere Informationen erhalten Sie unter AWS IoT Richtlinien für den Betrieb auf Datenebene und Konfigurieren Sie die AWS IoT Ding-Richtlinie.

Sie können einen der folgenden Schritte ausführen, um die IP-Detektorkomponente bereitzustellen:

- Verwenden Sie die Seite Discovery konfigurieren in der Konsole. Weitere Informationen finden Sie unter Konfigurieren Sie Cloud Discovery (Konsole).
- Erstellen und überarbeiten Sie Bereitstellungen, um den IP-Detektor einzubeziehen. Sie können die Konsole oder die AWS API verwenden AWS CLI, um Bereitstellungen zu verwalten. Weitere Informationen finden Sie unter Erstellen von Bereitstellungen.

Stellen Sie die IP-Detektorkomponente (Konsole) bereit

1. Wählen Sie im Navigationsmenü der AWS IoT Greengrass Konsole die Option Komponenten aus.
2. Wählen Sie auf der Seite Komponenten die Registerkarte Öffentliche Komponenten und wählen Sie dann aws.greengrass.clientdevices.IPDetector.
3. Auf der aws.greengrass.clientdevices.IPDetectorWählen Sie auf der Seite Deploy aus.
4. Wählen Sie unter Zur Bereitstellung hinzufügen eine vorhandene Bereitstellung aus, die Sie überarbeiten möchten, oder erstellen Sie eine neue Bereitstellung und klicken Sie dann auf Weiter.
5. Wenn Sie eine neue Bereitstellung erstellen möchten, wählen Sie das Ziel-Core-Gerät oder die Dinggruppe für die Bereitstellung aus. Wählen Sie auf der Seite „Ziel angeben“ unter Bereitstellungsziel ein Kerngerät oder eine Dinggruppe aus, und klicken Sie dann auf Weiter.
6. Vergewissern Sie sich auf der Seite „Komponenten auswählen“, dass aws.greengrass.clientdevices.IPDetectorDie Komponente ist ausgewählt, wählen Sie Weiter.
7. Wählen Sie auf der Seite Komponenten konfigurieren aws.greengrass.clientdevices.IPDetector, und gehen Sie dann wie folgt vor:

   a. Wählen Sie Komponente konfigurieren aus.

   b. In der Konfiguration aws.greengrass.clientdevices.IPDetectorModal, unter Konfigurationsupdate können Sie unter Konfiguration zum Zusammenführen ein Konfigurationsupdate eingeben, um die IP-Detektorkomponente zu konfigurieren.
   Sie können jede der folgenden Konfigurationsoptionen angeben. Setzen Sie `includeIPv4Addrs` auf `false` und `includeIPv6Addrs` auf `true`. Sie können dann die anderen IPv6 Konfigurationsoptionen aktualisieren.

      - `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, über die 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.
- `includeIPv4LinkLocalAddrs`— (Optional) Sie können diese Option aktivieren, um IPv4 verknüpfungslokale 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.
- `includeIPv6LoopbackAddrs`— (Optional) Sie können diese Option aktivieren, um IPv6 Loopback-Adressen zu erkennen und zu melden. Dies sind IP-Adressen, über die 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. Sie müssen `includeIPv4Addrs` auf `false` und auf einstellen`includeIPv6Addrs`, `true` um diese Option verwenden zu können. Sie benötigen IP Detector v2.2.0 oder höher, um diese Option verwenden zu können.
- `includeIPv6LinkLocalAddrs`— (Optional) Sie können diese Option aktivieren, um IPv6 Link-Local-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. Sie müssen `includeIPv4Addrs` auf `false` und auf einstellen, `includeIPv6Addrs` um diese Option verwenden `true` zu können. Sie benötigen IP Detector v2.2.0 oder höher, um diese Option verwenden zu können.
- `includeIPv4Addrs`— (Optional) Die Standardeinstellung ist auf true gesetzt. Sie können diese Option aktivieren, um IPv4 Adressen zu veröffentlichen, die auf dem Core-Gerät gefunden wurden. Sie benötigen IP Detector v2.2.0 oder höher, um diese Option verwenden zu können.
- `includeIPv6Addrs`— (Optional) Sie können diese Option aktivieren, um IPv6 Adressen zu veröffentlichen, die auf dem Core-Gerät gefunden wurden. Stellen Sie `includeIPv4Addrs` auf `false` ein, um diese Option zu verwenden. Sie benötigen IP Detector v2.2.0 oder höher, um diese Option verwenden zu können.

Das Konfigurationsupdate könnte dem folgenden Beispiel ähneln.

```
{
```

```
  "defaultPort": "8883",
  "includeIPv4LoopbackAddrs": false,
  "includeIPv4LinkLocalAddrs": false,
  "includeIPv6LoopbackAddrs": true,
  "includeIPv6LinkLocalAddrs": true,
  "includeIPv4Addrs": false,
  "includeIPv6Addrs": true
}
```

c. Wählen Sie Bestätigen, um das Modal zu schließen, 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 Review (Prüfen) die Option Deploy (Bereitstellen) aus.

   Es kann bis zu einer Minute dauern, bis die Bereitstellung abgeschlossen ist.

Stellen Sie die IP-Detektorkomponente bereit (AWS CLI)

Um die IP-Detektorkomponente bereitzustellen, erstellen Sie ein Bereitstellungsdokument, das das `components` Objekt enthält`aws.greengrass.clientdevices.IPDetector`, und geben Sie das Konfigurationsupdate für die Komponente an. Folgen Sie den Anweisungen unterErstellen von Bereitstellungen, um eine neue Einrichtung zu erstellen oder eine bestehende Einrichtung zu überarbeiten.

Sie können bei der Erstellung 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, über die 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.
- `includeIPv4LinkLocalAddrs`— (Optional) Sie können diese Option aktivieren, um IPv4 verknüpfungslokale 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.

- `includeIPv6LoopbackAddrs`— (Optional) Sie können diese Option aktivieren, um IPv6 Loopback-Adressen zu erkennen und zu melden. Dies sind IP-Adressen, über die 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. Sie müssen `includeIPv4Addrs` auf `false` und auf einstellen`includeIPv6Addrs`, `true` um diese Option verwenden zu können. Sie benötigen IP Detector v2.2.0 oder höher, um diese Option verwenden zu können.
- `includeIPv6LinkLocalAddrs`— (Optional) Sie können diese Option aktivieren, um IPv6 Link-Local-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. Sie müssen `includeIPv4Addrs` auf `false` und auf einstellen, `includeIPv6Addrs` um diese Option verwenden `true` zu können. Sie benötigen IP Detector v2.2.0 oder höher, um diese Option verwenden zu können.
- `includeIPv4Addrs`— (Optional) Die Standardeinstellung ist auf true gesetzt. Sie können diese Option aktivieren, um IPv4 Adressen zu veröffentlichen, die auf dem Core-Gerät gefunden wurden. Sie benötigen IP Detector v2.2.0 oder höher, um diese Option verwenden zu können.
- `includeIPv6Addrs`— (Optional) Sie können diese Option aktivieren, um IPv6 Adressen zu veröffentlichen, die auf dem Core-Gerät gefunden wurden. Stellen Sie `includeIPv4Addrs` auf `false` ein, um diese Option zu verwenden. Sie benötigen IP Detector v2.2.0 oder höher, um diese Option verwenden zu können.

Das folgende Beispieldokument zur teilweisen Bereitstellung gibt die Verwendung an. IPv6

```
{
  ...,
  "components": {
    ...,
    "aws.greengrass.clientdevices.IPDetector": {
      "componentVersion": "2.1.1",
      "configurationUpdate": {
        "merge": "{\"defaultPort\":\"8883\",}"
      }
    }
  }
}
```

Problembehandlung bei Client-Geräten

Verwenden Sie die Informationen und Lösungen zur Fehlerbehebung in diesem Abschnitt, um Probleme mit Greengrass-Client-Geräten und Client-Gerätekomponenten zu lösen.

Themen

- Probleme bei der Entdeckung von Greengrass
- Probleme mit der MQTT-Verbindung

Probleme bei der Entdeckung von Greengrass

Verwenden Sie die folgenden Informationen, um Probleme mit Greengrass Discovery zu beheben. Diese Probleme können auftreten, wenn Client-Geräte die Greengrass Discovery API verwenden, um ein Greengrass-Core-Gerät zu identifizieren, mit dem sie sich verbinden können.

Themen

- Probleme mit der Greengrass-Erkennung (HTTP-API)
- Probleme bei der Erkennung von Greengrass (AWS IoT Device SDK v2 für Python)
- Probleme bei der Erkennung von Greengrass (AWS IoT Device SDK v2 für C++)
- Probleme bei der Erkennung von Greengrass (AWS IoT Device SDK v2 für JavaScript)
- Probleme bei der Erkennung von Greengrass (AWS IoT Device SDK v2 für Java)

Probleme mit der Greengrass-Erkennung (HTTP-API)

Verwenden Sie die folgenden Informationen, um Probleme mit Greengrass Discovery zu beheben. Diese Fehler werden möglicherweise angezeigt, wenn Sie die Discovery-API mit cURL testen.

Themen

- curl: (52) Empty reply from server
- HTTP 403: {"message":null,"traceId":"a1b2c3d4-5678-90ab-cdef-11111EXAMPLE"}
- HTTP 404: {"errorMessage":"The thing provided for discovery was not found"}

curl: (52) Empty reply from server

Dieser Fehler wird möglicherweise angezeigt, wenn Sie in der Anfrage ein inaktives AWS IoT Zertifikat angeben.

Vergewissern Sie sich, dass dem Client-Gerät ein Zertifikat angehängt ist und ob das Zertifikat aktiv ist. Weitere Informationen finden Sie unter Anhängen einer Sache oder 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, sich selbst `greengrass:Discover` anzurufen.

Vergewissern Sie sich, dass das Zertifikat des Client-Geräts über eine Richtlinie verfügt, die dies zulässt`greengrass:Discover`. Sie können im `Resource` Abschnitt für diese Berechtigung keine `iot:Connection.Thing.*`Ding-Richtlinienvariablen () verwenden. Weitere Informationen finden Sie unter Discovery-Authentifizierung 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 V1 -Gruppen zugeordnet.
- Keines der dem Client-Gerät zugehörigen Greengrass-Core-Geräte oder AWS IoT Greengrass V1 -Gruppen hat einen MQTT-Broker-Endpunkt.
- Auf keinem der mit dem Client-Gerät verknüpften Greengrass-Core-Geräte wird die Authentifizierungskomponente für das Client-Gerät ausgeführt.

Vergewissern Sie sich, dass das Client-Gerät dem Core-Gerät zugeordnet ist, mit dem es eine Verbindung herstellen soll. Überprüfen Sie dann, ob auf dem Kerngerät die Authentifizierungskomponente für das Clientgerät ausgeführt wird und dass es mindestens einen MQTT-Broker-Endpunkt hat. Weitere Informationen finden Sie hier:

- Client-Geräte zuordnen
- Endpunkte von Kerngeräten verwalten
- Konfigurieren Sie Cloud Discovery (Konsole)

Probleme bei der Erkennung von Greengrass (AWS IoT Device SDK v2 für Python)

Verwenden Sie die folgenden Informationen, um Probleme mit Greengrass Discovery in der AWS IoT Device SDK Version 2 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 Anfrage ein inaktives AWS IoT Zertifikat angeben.

Vergewissern Sie sich, dass dem Client-Gerät ein Zertifikat angehängt ist und ob das Zertifikat aktiv ist. Weitere Informationen finden Sie unter Anhängen einer Sache oder 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, sich selbst `greengrass:Discover` anzurufen.

Vergewissern Sie sich, dass das Zertifikat des Client-Geräts über eine Richtlinie verfügt, die dies zulässt`greengrass:Discover`. Sie können im `Resource` Abschnitt für diese Berechtigung keine `iot:Connection.Thing.*`Ding-Richtlinienvariablen () verwenden. Weitere Informationen finden Sie unter Discovery-Authentifizierung 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 V1 -Gruppen zugeordnet.
- Keines der dem Client-Gerät zugehörigen Greengrass-Core-Geräte oder AWS IoT Greengrass V1 -Gruppen hat einen MQTT-Broker-Endpunkt.
- Auf keinem der mit dem Client-Gerät verknüpften Greengrass-Core-Geräte wird die Authentifizierungskomponente für das Client-Gerät ausgeführt.

Vergewissern Sie sich, dass das Client-Gerät dem Core-Gerät zugeordnet ist, mit dem es eine Verbindung herstellen soll. Überprüfen Sie dann, ob auf dem Kerngerät die Authentifizierungskomponente für das Clientgerät ausgeführt wird und dass es mindestens einen MQTT-Broker-Endpunkt hat. Weitere Informationen finden Sie hier:

- Client-Geräte zuordnen
- Endpunkte von Kerngeräten verwalten
- Konfigurieren Sie Cloud Discovery (Konsole)

Probleme bei der Erkennung von Greengrass (AWS IoT Device SDK v2 für C++)

Verwenden Sie die folgenden Informationen, um Probleme mit Greengrass Discovery in der AWS IoT Device SDK Version 2 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 Anfrage ein inaktives AWS IoT Zertifikat angeben.

Vergewissern Sie sich, dass dem Client-Gerät ein Zertifikat angehängt ist und ob das Zertifikat aktiv ist. Weitere Informationen finden Sie unter Anhängen einer Sache oder 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, sich selbst `greengrass:Discover` anzurufen.

Vergewissern Sie sich, dass das Zertifikat des Client-Geräts über eine Richtlinie verfügt, die dies zulässt`greengrass:Discover`. Sie können im `Resource` Abschnitt für diese Berechtigung keine `iot:Connection.Thing.*`Ding-Richtlinienvariablen () verwenden. Weitere Informationen finden Sie unter Discovery-Authentifizierung 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 V1 -Gruppen zugeordnet.
- Keines der dem Client-Gerät zugehörigen Greengrass-Core-Geräte oder AWS IoT Greengrass V1 -Gruppen hat einen MQTT-Broker-Endpunkt.
- Auf keinem der mit dem Client-Gerät verknüpften Greengrass-Core-Geräte wird die Authentifizierungskomponente für das Client-Gerät ausgeführt.

Vergewissern Sie sich, dass das Client-Gerät dem Core-Gerät zugeordnet ist, mit dem es eine Verbindung herstellen soll. Überprüfen Sie dann, ob auf dem Kerngerät die Authentifizierungskomponente für das Clientgerät ausgeführt wird und dass es mindestens einen MQTT-Broker-Endpunkt hat. Weitere Informationen finden Sie hier:

- Client-Geräte zuordnen
- Endpunkte von Kerngeräten verwalten
- Konfigurieren Sie Cloud Discovery (Konsole)

Probleme bei der Erkennung von Greengrass (AWS IoT Device SDK v2 für JavaScript)

Verwenden Sie die folgenden Informationen, um Probleme mit Greengrass Discovery in der AWS IoT Device SDK Version 2 für JavaScript zu beheben.

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 Anfrage ein inaktives AWS IoT Zertifikat angeben.

Vergewissern Sie sich, dass dem Client-Gerät ein Zertifikat angehängt ist und ob das Zertifikat aktiv ist. Weitere Informationen finden Sie unter Anhängen einer Sache oder 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, sich selbst `greengrass:Discover` anzurufen.

Vergewissern Sie sich, dass das Zertifikat des Client-Geräts über eine Richtlinie verfügt, die dies zulässt`greengrass:Discover`. Sie können im `Resource` Abschnitt für diese Berechtigung keine `iot:Connection.Thing.*`Ding-Richtlinienvariablen () verwenden. Weitere Informationen finden Sie unter Discovery-Authentifizierung 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 V1 -Gruppen zugeordnet.
- Keines der dem Client-Gerät zugehörigen Greengrass-Core-Geräte oder AWS IoT Greengrass V1 -Gruppen hat einen MQTT-Broker-Endpunkt.
- Auf keinem der mit dem Client-Gerät verknüpften Greengrass-Core-Geräte wird die Authentifizierungskomponente für das Client-Gerät ausgeführt.

Vergewissern Sie sich, dass das Client-Gerät dem Core-Gerät zugeordnet ist, mit dem es eine Verbindung herstellen soll. Überprüfen Sie dann, ob auf dem Kerngerät die

Authentifizierungskomponente für das Clientgerät ausgeführt wird und dass es mindestens einen MQTT-Broker-Endpunkt hat. Weitere Informationen finden Sie hier:

- Client-Geräte zuordnen
- Endpunkte von Kerngeräten verwalten
- Konfigurieren Sie Cloud Discovery (Konsole)

Error: Discovery failed (headers: [object Object])

Möglicherweise wird dieser Fehler (ohne HTTP-Antwortcode) angezeigt, wenn Sie das Greengrass Discovery-Beispiel 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 Anrufe zu tätigen.

  Vergewissern Sie sich, dass das Zertifikat des Client-Geräts über eine Richtlinie verfügt, die dies zulässt`greengrass:Discover`. Sie können im `Resource` Abschnitt für diese Berechtigung keine `iot:Connection.Thing.*`Ding-Richtlinienvariablen () verwenden. Weitere Informationen finden Sie unter Discovery-Authentifizierung und Autorisierung.
- Dieser Fehler kann in den folgenden Fällen auftreten:
  - Das Client-Gerät ist keinen Greengrass-Core-Geräten oder AWS IoT Greengrass V1 -Gruppen zugeordnet.
  - Keines der dem Client-Gerät zugehörigen Greengrass-Core-Geräte oder AWS IoT Greengrass V1 -Gruppen hat einen MQTT-Broker-Endpunkt.
  - Auf keinem der mit dem Client-Gerät verknüpften Greengrass-Core-Geräte wird die Authentifizierungskomponente für das Client-Gerät ausgeführt.

  Vergewissern Sie sich, dass das Client-Gerät dem Core-Gerät zugeordnet ist, mit dem es eine Verbindung herstellen soll. Überprüfen Sie dann, ob auf dem Kerngerät die Authentifizierungskomponente für das Clientgerät ausgeführt wird und dass es mindestens einen MQTT-Broker-Endpunkt hat. Weitere Informationen finden Sie hier:
  - Client-Geräte zuordnen
  - Endpunkte von Kerngeräten verwalten
  - Konfigurieren Sie Cloud Discovery (Konsole)

Probleme bei der Erkennung von Greengrass (AWS IoT Device SDK v2 für Java)

Verwenden Sie die folgenden Informationen, um Probleme mit Greengrass Discovery in der AWS IoT Device SDK Version 2 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 Anfrage ein inaktives AWS IoT Zertifikat angeben.

Vergewissern Sie sich, dass dem Client-Gerät ein Zertifikat angehängt ist und ob das Zertifikat aktiv ist. Weitere Informationen finden Sie unter Anhängen einer Sache oder 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, sich selbst `greengrass:Discover` anzurufen.

Vergewissern Sie sich, dass das Zertifikat des Client-Geräts über eine Richtlinie verfügt, die dies zulässt`greengrass:Discover`. Sie können im `Resource` Abschnitt für diese Berechtigung keine `iot:Connection.Thing.*`Ding-Richtlinienvariablen () verwenden. Weitere Informationen finden Sie unter Discovery-Authentifizierung 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 V1 -Gruppen zugeordnet.

- Keines der dem Client-Gerät zugehörigen Greengrass-Core-Geräte oder AWS IoT Greengrass V1 - Gruppen hat einen MQTT-Broker-Endpunkt.
- Auf keinem der mit dem Client-Gerät verknüpften Greengrass-Core-Geräte wird die Authentifizierungskomponente für das Client-Gerät ausgeführt.

Vergewissern Sie sich, dass das Client-Gerät dem Core-Gerät zugeordnet ist, mit dem es eine Verbindung herstellen soll. Überprüfen Sie dann, ob auf dem Kerngerät die Authentifizierungskomponente für das Clientgerät ausgeführt wird und dass es mindestens einen MQTT-Broker-Endpunkt hat. Weitere Informationen finden Sie hier:

- Client-Geräte zuordnen
- Endpunkte von Kerngeräten verwalten
- Konfigurieren Sie Cloud Discovery (Konsole)

Probleme mit der MQTT-Verbindung

Verwenden Sie die folgenden Informationen, um Probleme mit MQTT-Verbindungen auf Client-Geräten zu beheben. Diese Probleme können auftreten, wenn Client-Geräte versuchen, über MQTT eine Verbindung zu einem Kerngerät herzustellen.

Themen

- io.moquette.broker.Authorizator: Client does not have read permissions on the topic
- Probleme mit der MQTT-Verbindung (Python)
- Probleme mit der MQTT-Verbindung (C++)
- Probleme mit der MQTT-Verbindung (Java)
- Probleme mit der MQTT-Verbindung () JavaScript

io.moquette.broker.Authorizator: Client does not have read permissions on the topic

Möglicherweise wird dieser Fehler in den Greengrass-Protokollen angezeigt, wenn ein Client-Gerät versucht, ein MQTT-Thema zu abonnieren, für das es keine entsprechenden Rechte hat. Die Fehlermeldung beinhaltet das Thema.

Vergewissern Sie sich, dass die Konfiguration der Komponente für die Authentifizierung auf dem Client-Gerät Folgendes umfasst:

- Eine Gerätegruppe, die dem Client-Gerät entspricht.
- Eine Autorisierungsrichtlinie für Clientgeräte für diese Gerätegruppe, die die `mqtt:subscribe` Berechtigung für das Thema erteilt.

Weitere Informationen zur Bereitstellung und Konfiguration der Authentifizierungskomponente für Clientgeräte finden Sie im Folgenden:

- Konfigurieren Sie Cloud Discovery (Konsole)
- Authentifizierung auf Client-Geräten
- Erstellen von Bereitstellungen

Probleme mit der MQTT-Verbindung (Python)

Verwenden Sie die folgenden Informationen, um Probleme mit MQTT-Verbindungen auf Client-Geräten zu beheben, wenn Sie AWS IoT Device SDK Version 2 für Python 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 für das Clientgerät keine Autorisierungsrichtlinie für das Clientgerät definiert, die dem Clientgerät die Berechtigung zum Herstellen einer Verbindung erteilt.

Vergewissern Sie sich, dass die Konfiguration der Authentifizierungskomponente für das Clientgerät Folgendes umfasst:

- Eine Gerätegruppe, die dem Client-Gerät entspricht.
- Eine Autorisierungsrichtlinie für Clientgeräte für diese Gerätegruppe, die die `mqtt:connect` Berechtigung für das Client-Gerät erteilt.

Weitere Informationen zur Bereitstellung und Konfiguration der Authentifizierungskomponente für Clientgeräte finden Sie im Folgenden:

- Konfigurieren Sie Cloud Discovery (Konsole)
- Authentifizierung auf Client-Geräten
- Erstellen von Bereitstellungen

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

Dieser Fehler wird möglicherweise angezeigt, wenn die Komponente für die Authentifizierung auf dem Clientgerät keine Autorisierungsrichtlinie für Clientgeräte definiert, die dem Clientgerät die Berechtigung zum Herstellen einer Verbindung erteilt.

Vergewissern Sie sich, dass die Konfiguration der Authentifizierungskomponente für das Clientgerät Folgendes umfasst:

- Eine Gerätegruppe, die dem Client-Gerät entspricht.
- Eine Autorisierungsrichtlinie für Clientgeräte für diese Gerätegruppe, die die `mqtt:connect` Berechtigung für das Client-Gerät erteilt.

Weitere Informationen zur Bereitstellung und Konfiguration der Authentifizierungskomponente für Clientgeräte finden Sie im Folgenden:

- Konfigurieren Sie Cloud Discovery (Konsole)
- Authentifizierung auf Client-Geräten
- Erstellen von Bereitstellungen

Probleme mit der MQTT-Verbindung (C++)

Verwenden Sie die folgenden Informationen, um Probleme mit MQTT-Verbindungen auf Client-Geräten 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 für das Clientgerät keine Autorisierungsrichtlinie für Clientgeräte definiert, die dem Clientgerät die Berechtigung zum Herstellen einer Verbindung erteilt.

Vergewissern Sie sich, dass die Konfiguration der Authentifizierungskomponente für das Clientgerät Folgendes umfasst:

- Eine Gerätegruppe, die dem Client-Gerät entspricht.
- Eine Autorisierungsrichtlinie für Clientgeräte für diese Gerätegruppe, die die `mqtt:connect` Berechtigung für das Client-Gerät erteilt.

Weitere Informationen zur Bereitstellung und Konfiguration der Authentifizierungskomponente für Clientgeräte finden Sie im Folgenden:

- Konfigurieren Sie Cloud Discovery (Konsole)
- Authentifizierung auf Client-Geräten
- Erstellen von Bereitstellungen

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

Dieser Fehler wird möglicherweise angezeigt, wenn die Komponente für die Authentifizierung auf dem Clientgerät keine Autorisierungsrichtlinie für Clientgeräte definiert, die dem Clientgerät die Berechtigung zum Herstellen einer Verbindung erteilt.

Vergewissern Sie sich, dass die Konfiguration der Authentifizierungskomponente für das Clientgerät Folgendes umfasst:

- Eine Gerätegruppe, die dem Client-Gerät entspricht.
- Eine Autorisierungsrichtlinie für Clientgeräte für diese Gerätegruppe, die die `mqtt:connect` Berechtigung für das Client-Gerät erteilt.

Weitere Informationen zur Bereitstellung und Konfiguration der Authentifizierungskomponente für Clientgeräte finden Sie im Folgenden:

- Konfigurieren Sie Cloud Discovery (Konsole)
- Authentifizierung auf Client-Geräten

- Erstellen von Bereitstellungen

Probleme mit der MQTT-Verbindung (Java)

Verwenden Sie die folgenden Informationen, um Probleme mit MQTT-Verbindungen auf Client-Geräten zu beheben, wenn Sie AWS IoT Device SDK Version 2 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 für das Clientgerät keine Autorisierungsrichtlinie für das Clientgerät definiert, die dem Clientgerät die Berechtigung zum Herstellen einer Verbindung erteilt.

Vergewissern Sie sich, dass die Konfiguration der Authentifizierungskomponente für das Clientgerät Folgendes umfasst:

- Eine Gerätegruppe, die dem Client-Gerät entspricht.
- Eine Autorisierungsrichtlinie für Clientgeräte für diese Gerätegruppe, die die `mqtt:connect` Berechtigung für das Client-Gerät erteilt.

Weitere Informationen zur Bereitstellung und Konfiguration der Authentifizierungskomponente für Clientgeräte finden Sie im Folgenden:

- Konfigurieren Sie Cloud Discovery (Konsole)
- Authentifizierung auf Client-Geräten
- Erstellen von Bereitstellungen

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

Dieser Fehler wird möglicherweise angezeigt, wenn die Komponente für die Authentifizierung auf dem Clientgerät keine Autorisierungsrichtlinie für Clientgeräte definiert, die dem Clientgerät die Berechtigung zum Herstellen einer Verbindung erteilt.

Vergewissern Sie sich, dass die Konfiguration der Authentifizierungskomponente für das Clientgerät Folgendes umfasst:

- Eine Gerätegruppe, die dem Client-Gerät entspricht.
- Eine Autorisierungsrichtlinie für Clientgeräte für diese Gerätegruppe, die die `mqtt:connect` Berechtigung für das Client-Gerät erteilt.

Weitere Informationen zur Bereitstellung und Konfiguration der Authentifizierungskomponente für Clientgeräte finden Sie im Folgenden:

- Konfigurieren Sie Cloud Discovery (Konsole)
- Authentifizierung auf Client-Geräten
- Erstellen von Bereitstellungen

Probleme mit der MQTT-Verbindung () JavaScript

Verwenden Sie die folgenden Informationen, um Probleme mit MQTT-Verbindungen auf Client-Geräten zu beheben, wenn Sie AWS IoT Device SDK v2 für verwenden. JavaScript

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 für das Clientgerät keine Autorisierungsrichtlinie für das Clientgerät definiert, die dem Clientgerät die Berechtigung zum Herstellen einer Verbindung erteilt.

Vergewissern Sie sich, dass die Konfiguration der Authentifizierungskomponente für das Clientgerät Folgendes umfasst:

- Eine Gerätegruppe, die dem Client-Gerät entspricht.
- Eine Autorisierungsrichtlinie für Clientgeräte für diese Gerätegruppe, die die `mqtt:connect` Berechtigung für das Client-Gerät erteilt.

Weitere Informationen zur Bereitstellung und Konfiguration der Authentifizierungskomponente für Clientgeräte finden Sie im Folgenden:

- Konfigurieren Sie Cloud Discovery (Konsole)
- Authentifizierung auf Client-Geräten
- Erstellen von Bereitstellungen

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

Dieser Fehler wird möglicherweise angezeigt, wenn die Komponente für die Authentifizierung auf dem Clientgerät keine Autorisierungsrichtlinie für Clientgeräte definiert, die dem Clientgerät die Berechtigung zum Herstellen einer Verbindung erteilt.

Vergewissern Sie sich, dass die Konfiguration der Authentifizierungskomponente für das Clientgerät Folgendes umfasst:

- Eine Gerätegruppe, die dem Client-Gerät entspricht.
- Eine Autorisierungsrichtlinie für Clientgeräte für diese Gerätegruppe, die die `mqtt:connect` Berechtigung für das Client-Gerät erteilt.

Weitere Informationen zur Bereitstellung und Konfiguration der Authentifizierungskomponente für Clientgeräte finden Sie im Folgenden:

- Konfigurieren Sie Cloud Discovery (Konsole)
- Authentifizierung auf Client-Geräten
- Erstellen von Bereitstellungen

Interagiere mit Geräteschatten

Greengrass-Core-Geräte können mithilfe von Komponenten mit AWS IoT Geräteschatten interagieren. Ein Schatten ist ein JSON-Dokument, das die aktuellen oder gewünschten Statusinformationen für eine AWS IoT Sache speichert. Schatten können den Status eines Geräts für andere AWS IoT Greengrass Komponenten verfügbar machen, unabhängig davon, ob das Gerät angeschlossen 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 Dienste können Cloud-Schatten mithilfe von MQTT und den reservierten MQTT-Shadow-Themen, HTTP mithilfe der Device Shadow-REST-APIund dem for erstellen, aktualisieren und löschen.AWS CLIAWS IoT

Die Shadow-Manager-Komponente ermöglicht es Ihren Greengrass-Komponenten, lokale Shadows zu erstellen, zu aktualisieren und zu löschen, indem sie den lokalen Shadow-Service und die lokalen Publish/Subscribe-Shadow-Themen verwenden. Der Shadow-Manager verwaltet auch die Speicherung dieser lokalen Shadow-Dokumente auf Ihrem Kerngerät und kümmert sich um die Synchronisation von Shadow-Statusinformationen mit Cloud-Shadows.

Sie können die Shadow-Manager-Komponente auch verwenden, um lokale Shadows für Client-Geräte zu verwalten, die eine Verbindung zum Kerngerät herstellen. Damit Shadow Manager Schatten auf Client-Geräten verwalten kann, konfigurieren Sie die MQTT-Bridge-Komponente so, dass sie Nachrichten zwischen dem lokalen MQTT-Broker und dem lokalen Publish/Subscribe-Dienst weiterleitet. Weitere Informationen finden Sie unter Interagieren Sie mit Schatten auf Client-Geräten und synchronisieren Sie sie.

Weitere Informationen zu AWS IoT Device Shadow-Konzepten finden Sie unter AWS IoT Device Shadow Service im AWS IoT Developer Guide.

Themen

- Interagiere mit Schatten in Komponenten
- Synchronisieren Sie lokale Geräteschatten mit AWS IoT Core

Interagiere mit Schatten in Komponenten

Sie können benutzerdefinierte Komponenten, einschließlich Lambda-Funktionskomponenten, entwickeln, die den lokalen Shadow-Dienst verwenden, um lokale Shadow-Dokumente und Shadow-Dokumente auf Client-Geräten zu lesen und zu ändern.

Benutzerdefinierte Komponenten interagieren mit dem lokalen Shadow-Dienst mithilfe der AWS IoT Greengrass IPC-Kernbibliotheken in. AWS IoT Device SDK Die Shadow-Manager-Komponente aktiviert den lokalen Shadow-Dienst auf Ihrem Kerngerät.

Um die Shadow Manager-Komponente auf einem Greengrass-Core-Gerät bereitzustellen, erstellen Sie eine Bereitstellung, die die `aws.greengrass.ShadowManager` Komponente enthält.

> **Note**
>
> Standardmäßig ermöglicht die Bereitstellung der Shadow-Manager-Komponente nur lokale Shadow-Operationen. Um die Synchronisierung von Shadow-Statusinformationen für Core-Geräteshadows oder Shadows für Client-Geräte mit den entsprechenden Cloud-Shadow-Dokumenten in zu ermöglichen AWS IoT Greengrass AWS IoT Core, müssen Sie ein Konfigurationsupdate für die Shadow-Manager-Komponente erstellen, das den `synchronize` Parameter enthält. Weitere Informationen finden Sie unter Synchronisieren Sie lokale Geräteschatten mit AWS IoT Core.

Themen

- Schattenstatus abrufen und ändern
- Reagieren Sie auf Änderungen des Schattenstatus

Schattenstatus abrufen und ändern

Die Shadow-IPC-Operationen rufen Statusinformationen in lokalen Shadow-Dokumenten ab und aktualisieren sie. Die Shadow-Manager-Komponente kümmert sich um die Speicherung dieser Shadow-Dokumente auf Ihrem Kerngerät.

Um lokale Shadow-Status zu ändern

1. Fügen Sie dem Rezept für Ihre benutzerdefinierte Komponente Autorisierungsrichtlinien hinzu, damit die Komponente Nachrichten zu lokalen Shadow-Themen empfangen kann.

Beispiele für Autorisierungsrichtlinien finden Sie unter Beispiele für lokale Shadow-IPC-Autorisierungsrichtlinien.

2. Verwenden Sie die Shadow-IPC-Operationen, um Informationen zum Schattenstatus abzurufen und zu ändern. Weitere Hinweise zur Verwendung von Shadow-IPC-Operationen im Komponentencode finden Sie unter. Interagiere mit lokalen Schatten

> **Note**
>
> Damit ein Core-Gerät mit Client-Geräteshadows interagieren kann, müssen Sie auch die MQTT-Bridge-Komponente konfigurieren und bereitstellen. Weitere Informationen finden Sie unter Aktivieren des Shadow-Managers für die Kommunikation mit Client-Geräten.

Reagieren Sie auf Änderungen des Schattenstatus

Greengrass-Komponenten verwenden die lokalen publish/subscribe interface to communicate on a core device. To enable a custom component to react to shadow state changes, you can subscribe to the local publish/subscribe Themen. Dadurch kann die Komponente Nachrichten zu den lokalen Shadow-Themen empfangen und dann auf diese Nachrichten reagieren.

Lokale Shadow-Themen verwenden dasselbe Format wie die AWS IoT Device-Shadow-MQTT-Themen. Weitere Informationen zu Shadow-Themen finden Sie unter Device Shadow MQTT-Themen im AWS IoT Developer Guide.

Um auf lokale Änderungen des Shadow-Status zu reagieren

1. Fügen Sie dem Rezept für Ihre benutzerdefinierte Komponente Zugriffskontrollrichtlinien hinzu, damit die Komponente Nachrichten zu lokalen Shadow-Themen empfangen kann.

   Beispiele für Autorisierungsrichtlinien finden Sie unter Beispiele für lokale Shadow-IPC-Autorisierungsrichtlinien.

2. Um eine benutzerdefinierte Aktion in einer Komponente zu initiieren, verwenden Sie `SubscribeToTopic` IPC-Operationen, um die Shadow-Themen zu abonnieren, zu denen Sie Nachrichten erhalten möchten. Weitere Hinweise zur Verwendung von lokalen IPC-Vorgängen zum Veröffentlichen und Abonnieren im Komponentencode finden Sie unter. Lokale Nachrichten veröffentlichen/abonnieren

3. Um eine Lambda-Funktion aufzurufen, geben Sie in der Konfiguration der Ereignisquelle den Namen des Schattenthemas an und geben Sie an, dass es sich um ein lokales Veröffentlichungs-/Abonnement-Thema handelt. Hinweise zum Erstellen von Lambda-Funktionskomponenten finden Sie unterAWS Lambda Funktionen ausführen.

> **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 Aktivieren des Shadow-Managers für die Kommunikation mit Client-Geräten.

Synchronisieren Sie lokale Geräteschatten mit AWS IoT Core

Die Shadow-Manager-Komponente ermöglicht AWS IoT Greengrass die Synchronisierung von Shadow-Status auf lokalen Geräten mit AWS IoT Core. Sie müssen die Konfiguration der Shadow-Manager-Komponente so ändern, dass sie den `synchronization` Konfigurationsparameter enthält, und die Namen der AWS IoT Dinge für Ihre Geräte sowie die Schatten, die Sie synchronisieren möchten, angeben.

Wenn Sie den Schattenmanager für die Synchronisierung von Schatten konfigurieren, werden alle Statusänderungen für bestimmte Schatten synchronisiert, unabhängig davon, ob die Änderungen in lokalen Schattendokumenten oder in Cloud-Schattendokumenten auftreten.

Sie können auch angeben, ob die Schatten-Manager-Komponente Schatten in Echtzeit oder in regelmäßigen Abständen synchronisiert. Standardmäßig synchronisiert die Shadow-Manager-Komponente Schatten in Echtzeit, sodass das Kerngerät Shadow-Updates von AWS IoT Core und zu jedem Update sendet und empfängt. Sie können regelmäßige Intervalle konfigurieren, um die Bandbreitennutzung und die Gebühren zu reduzieren.

Themen

- Voraussetzungen
- Konfigurieren Sie die Shadow Manager-Komponente
- Synchronisieren Sie lokale Schatten
- Konfliktverhalten bei Shadow Merge

Voraussetzungen

Um lokale Shadows mit zu synchronisieren AWS IoT Core, müssen Sie die AWS IoT Richtlinie des Greengrass Core-Geräts so konfigurieren, dass die folgenden AWS IoT Core Shadow-Richtlinienaktionen zulässig sind.

- `iot:GetThingShadow`
- `iot:UpdateThingShadow`
- `iot:DeleteThingShadow`

Weitere Informationen finden Sie hier:

- AWS IoT Core Richtlinienaktionen im AWS IoT Developer Guide
- Minimale AWS IoT Richtlinie für AWS IoT Greengrass V2 Kerngeräte
- Aktualisieren Sie die AWS IoT Richtlinie eines Kerngeräts

Konfigurieren Sie die Shadow Manager-Komponente

Der Shadow-Manager benötigt eine Liste von Shadow-Namenszuordnungen, um Shadow-Statusinformationen in lokalen Shadow-Dokumenten mit Cloud-Shadow-Dokumenten in synchronisieren zu können. AWS IoT Core

Um Shadow-Status zu synchronisieren, erstellen Sie eine Bereitstellung, die die `aws.greengrass.ShadowManager` Komponente enthält, und geben Sie die Shadows, die Sie synchronisieren möchten, im `synchronize` Konfigurationsparameter in der Shadow-Manager-Konfiguration in der Bereitstellung an.

> **Note**
>
> Damit ein Kerngerät mit den Schatten von Client-Geräten interagieren kann, müssen Sie auch die MQTT-Bridge-Komponente konfigurieren und bereitstellen. Weitere Informationen finden Sie unter Aktivieren des Shadow-Managers für die Kommunikation mit Client-Geräten.

Das folgende Beispiel für ein Konfigurationsupdate weist die Shadow Manager-Komponente an, die folgenden Shadows mit AWS IoT Core zu synchronisieren:

- Der klassische Shadow für das Core-Gerät
- Das `MyCoreShadow` nach dem Kerngerät benannte
- Der klassische Schatten für ein IoT-Ding namens `MyDevice2`
- Die benannten Schatten `MyShadowA` und `MyShadowB` für ein IoT-Ding namens `MyDevice1`

Dieses Konfigurationsupdate legt fest, dass Schatten mit AWS IoT Core in Echtzeit synchronisiert werden sollen. Wenn Sie Shadow Manager v2.1.0 oder höher verwenden, können Sie die Shadow Manager-Komponente so konfigurieren, dass Schatten in regelmäßigen Abständen synchronisiert werden. Um diese Funktion zu konfigurieren, ändern Sie die Synchronisierungsstrategie auf `periodic` und geben Sie a `delay` in Sekunden für das Intervall an. Weitere Informationen finden Sie im Strategiekonfigurationsparameter der Shadow Manager-Komponente.

Dieses Konfigurationsupdate legt fest, dass Schatten in beide Richtungen zwischen AWS IoT Core und dem Kerngerät synchronisiert werden. 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 diese Funktion zu konfigurieren, ändern Sie die Synchronisation `direction` auf `deviceToCloud` oder. `cloudToDevice` Weitere Informationen finden Sie im Konfigurationsparameter 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"
  }
}
```

Synchronisieren Sie lokale Schatten

Wenn das Greengrass-Core-Gerät mit der AWS IoT Cloud verbunden ist, führt der Shadow-Manager die folgenden Aufgaben für die Shadows aus, die Sie in der Komponentenkonfiguration angeben. Das Verhalten hängt von der Konfigurationsoption für die Richtung der Schattensynchronisierung ab, die Sie angeben. Standardmäßig verwendet der Schattenmanager die `betweenDeviceAndCloud` Option, um Schatten in beide Richtungen zu synchronisieren. Wenn Sie Shadow Manager v2.2.0 oder höher verwenden, können Sie das Kerngerät so konfigurieren, dass Schatten nur in eine Richtung synchronisiert werden. Dies kann oder sein`cloudToDevice`. `deviceToCloud`

- Wenn die Konfiguration der Shadow-Synchronisierungsrichtung `betweenDeviceAndCloud` oder laute`cloudToDevice`, ruft Shadow Manager die gemeldeten Statusinformationen aus dem Cloud-Shadow-Dokument in ab. AWS IoT Core Anschließend werden lokal gespeicherte Shadow-Dokumente aktualisiert, um den Gerätestatus zu synchronisieren.
- Wenn die Konfiguration der Shadow-Synchronisierungsrichtung `betweenDeviceAndCloud` oder ist`deviceToCloud`, veröffentlicht Shadow Manager den aktuellen Status des Geräts im Cloud-Shadow-Dokument.

Konfliktverhalten bei Shadow Merge

In einigen Fällen, z. B. wenn das Kerngerät vom Internet getrennt wird, kann sich ein Shadow im lokalen Shadow-Dienst und in der AWS IoT Cloud ändern, bevor der Shadow-Manager die Änderungen synchronisiert. Aus diesem Grund unterscheiden sich der gewünschte und der gemeldete Status zwischen dem lokalen Shadow-Dienst und der Cloud AWS IoT

Wenn der Shadow-Manager den Shadow synchronisiert, führt er die Änderungen gemäß dem folgenden Verhalten zusammen:

- Wenn Sie eine ältere Version von Shadow Manager als Version 2.2.0 verwenden oder wenn Sie die Richtung der `betweenDeviceAndCloud` Shadow-Synchronisierung angeben, gilt das folgende Verhalten:
  - Wenn im gewünschten Zustand eines Shadows ein Zusammenführungskonflikt auftritt, überschreibt der Shadow-Manager den widersprüchlichen Abschnitt des lokalen Shadow-Dokuments mit dem Wert aus der Cloud. AWS IoT
  - Wenn im gemeldeten Status eines Shadows ein Zusammenführungskonflikt auftritt, überschreibt der Shadow-Manager den widersprüchlichen Abschnitt des Schattens in der AWS IoT Cloud mit dem Wert aus dem lokalen Shadow-Dokument.
- Wenn Sie die Richtung der `deviceToCloud` Schattensynchronisierung angeben, überschreibt der Schattenmanager den Abschnitt des Schattens in der AWS IoT Cloud, der den Konflikt verursacht, mit dem Wert aus dem lokalen Schattendokument.
- Wenn Sie die Richtung der `cloudToDevice` Schattensynchronisierung angeben, überschreibt der Schattenmanager den Abschnitt des lokalen Schattendokuments, der den Konflikt verursacht, mit dem Wert aus der Cloud. AWS IoT

Datenströme auf Greengrass-Kerngeräten verwalten

AWS IoT Greengrass Stream Manager macht es effizienter und zuverlässiger, großvolumige IoT-Daten auf den AWS Cloud zu übertragen. Stream Manager verarbeitet Datenströme auf dem AWS IoT Greengrass Core, bevor er sie in den AWS Cloud exportiert. Stream Manager lässt sich in gängige Szenarien wie maschinelles Lernen (ML) -Inferenz integrieren, bei denen das AWS IoT Greengrass Core-Gerät Daten verarbeitet und analysiert, bevor es die Daten an die AWS Cloud oder lokale Speicherziele exportiert.

Stream Manager bietet eine gemeinsame Schnittstelle zur Vereinfachung der Entwicklung benutzerdefinierter Komponenten, sodass Sie keine benutzerdefinierten Stream-Management-Funktionen 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 Datenspeicherung für jeden Stream definieren, um zu steuern, wie 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 mit Stream-Daten umgeht, wenn er verbunden oder getrennt wird. Sie können auch Prioritäten festlegen, um die Reihenfolge zu steuern, in der der AWS IoT Greengrass Core Streams in die exportiert. AWS Cloud Auf diese Weise können Sie kritische Daten früher verarbeiten als andere Daten.

Sie können den Stream-Manager so konfigurieren, dass Daten automatisch AWS Cloud zur Speicherung oder Weiterverarbeitung und Analyse in den exportiert werden. Stream Manager unterstützt Exporte an die folgenden AWS Cloud Ziele:

- Kanäle rein AWS IoT Analytics. AWS IoT Analytics ermöglicht es Ihnen, erweiterte Analysen Ihrer Daten durchzuführen, um Geschäftsentscheidungen zu treffen und Modelle für maschinelles Lernen zu verbessern. Weitere Informationen finden Sie unter Was ist AWS IoT Analytics? im AWS IoT Analytics -Benutzerhandbuch.
- Streams in Amazon Kinesis Data Streams. Sie können Kinesis Data Streams verwenden, um große Datenmengen zu aggregieren und in ein Data Warehouse oder MapReduce Cluster zu laden. Weitere Informationen finden Sie unter Was ist Amazon Kinesis Data Streams? im Entwicklerhandbuch für Amazon Kinesis Data Streams.

- Eigenschaften von Vermögenswerten in. AWS IoT SiteWise AWS IoT SiteWise ermöglicht das Sammeln, Organisieren und Analysieren von Daten aus Industrieanlagen in großem Maßstab. Weitere Informationen finden Sie unter Was ist AWS IoT SiteWise? im AWS IoT SiteWise Benutzerhandbuch.
- Objekte in Amazon 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 Developer Guide.

Stream-Management-Workflow

Ihre IoT-Anwendungen interagieren über das Stream Manager SDK mit dem Stream Manager.

In einem einfachen Arbeitsablauf verwendet eine Komponente im AWS IoT Greengrass Kern IoT-Daten, wie z. B. Temperatur- und Druckmesswerte in Zeitreihen. Die Komponente könnte die Daten filtern oder komprimieren und dann das Stream Manager SDK aufrufen, um die Daten in einen Stream im Stream Manager zu schreiben. Stream Manager kann den Stream auf der Grundlage der 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 Streams lesen und in sie schreiben, um Daten auf dem AWS IoT Greengrass Kerngerät zu filtern, zu aggregieren und zu analysieren. Dies ermöglicht es, schnell auf lokale Ereignisse zu reagieren und wertvolle Informationen zu extrahieren, bevor die Daten vom Core zu den AWS Cloud oder lokalen Zielen übertragen werden.

Stellen Sie zunächst die Stream Manager-Komponente auf Ihrem AWS IoT Greengrass Kerngerät bereit. Konfigurieren Sie bei der Bereitstellung die Parameter der Stream Manager-Komponente, um Einstellungen zu definieren, die für alle Streams auf dem Greengrass-Core-Gerät gelten. Verwenden Sie diese Parameter, um zu steuern, wie Stream Manager Streams auf der Grundlage Ihrer Geschäftsanforderungen und Umgebungsbeschränkungen speichert, verarbeitet und exportiert.

Nachdem Sie den Stream Manager konfiguriert haben, können Sie Ihre IoT-Anwendungen erstellen und bereitstellen. Dabei handelt es sich in der Regel um benutzerdefinierte Komponenten, die `StreamManagerClient` im Stream Manager SDK verwendet werden, 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

Die folgenden Anforderungen gelten für die Verwendung von Stream Manager:

- 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 dem 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 für 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 den Stream-Manager verwenden zu können.

  > **Note**
  >
  > Wenn Sie den 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 gewähren. Je nach Ziel können auch andere Anforderungen gelten. Weitere Informationen finden Sie unter:
  - the section called “AWS IoT Analytics Kanäle”
  - the section called “Amazon Kinesis Kinesis-Datenströme”
  - the section called “AWS IoT SiteWise Eigenschaften von Vermögenswerten”
  - the section called “Amazon-S3-Objekte”

  Sie sind für die Wartung dieser AWS Cloud Ressourcen verantwortlich.

Datensicherheit

Beachten Sie bei der Verwendung des Stream-Managers die folgenden Sicherheitsüberlegungen.

Lokale Datensicherheit

AWS IoT Greengrass verschlüsselt keine Stream-Daten im Ruhezustand oder bei der Übertragung zwischen lokalen Komponenten auf dem Kerngerät.

- Daten im Ruhezustand. Streamdaten werden lokal in einem Speicherverzeichnis gespeichert. Verlässt sich aus Gründen der Datensicherheit AWS IoT Greengrass auf Dateiberechtigungen und vollständige Festplattenverschlüsselung, sofern diese Option aktiviert ist. Sie können den optionalen Parameter STREAM_MANAGER_STORE_ROOT_DIR verwenden, um das Speicherverzeichnis anzugeben. Wenn Sie diesen Parameter später ändern, um ein anderes Speicherverzeichnis zu verwenden, werden das vorherige Speicherverzeichnis und sein Inhalt AWS IoT Greengrass nicht gelöscht.
- Daten werden lokal übertragen. AWS IoT Greengrass verschlüsselt keine Stream-Daten bei der lokalen Übertragung zwischen Datenquellen, AWS IoT Greengrass Komponenten, dem Stream Manager SDK und Stream Manager.
- Daten bei der Übertragung zum AWS Cloud. Datenströme, die vom Stream Manager exportiert werden, AWS Cloud verwenden die standardmäßige AWS Service-Client-Verschlüsselung mit Transport Layer Security (TLS).

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 Kerngeräten bereitstellen.

	Enabled	Disabled
Parameterwert	`true` (Standard und empfohlen)	`false`
Zulässige Clients	Greengrass-Komponenten auf dem Kerngerät	Greengrass-Komponenten auf dem Kerngerät Andere Prozesse, die auf dem Greengrass Core-Gerät ausgeführt werden

Weitere Informationen finden Sie auch unter

- the section called “Konfigurieren des Stream-Managers”
- the section called “Wird verwendet StreamManagerClient , um mit Streams zu arbeiten”
- the section called “Exportieren Sie Konfigurationen für unterstützte Cloud-Ziele”

AWS IoT Greengrass Streammanager konfigurieren

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 die 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 Kerngeräten 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 Stream-Manager-Stammordner angeben. Mit diesen Befehlen kann der Standardsystembenutzer,`ggc_user`, in 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
```

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`, dass ein zufälliger verfügbarer Port verwendet werden soll.

Client authentifizieren

Parametername: `STREAM_MANAGER_AUTHENTICATE_CLIENT`

Gibt an, ob Clients authentifiziert werden müssen, um mit dem Stream-Manager zu interagieren. Die gesamte Interaktion zwischen Clients und Stream Manager wird vom 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 mit dem Stream Manager SDK zu authentifizieren.

- `false`. Ermöglicht es jedem Prozess, der auf dem AWS IoT Greengrass Core läuft, ein Client zu sein. Setzen Sie den Wert nicht auf, `false` es sei denn, Ihr Geschäftsszenario erfordert dies. Verwenden Sie diese Option beispielsweise `false` nur, wenn Prozesse, die keine Komponenten sind, auf dem Kerngerät 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 beschränkten 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, die hier in der Reihenfolge der Stufen aufgeführt sind:

- `TRACE`
- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Standard: `INFO`

Mindestgröße für mehrteiligen Upload

Parametername: `STREAM_MANAGER_EXPORTER_S3_DESTINATION_MULTIPART_UPLOAD_MIN_PART_SIZE_BYTES`

Die Mindestgröße (in Byte) eines Teils in einem mehrteiligen Upload auf Amazon S3. Stream Manager verwendet diese Einstellung und die Größe der Eingabedatei, um zu bestimmen, wie Daten in einer mehrteiligen PUT-Anfrage gebündelt werden. Der Standard- und Mindestwert ist `5242880` Byte (5 MB).

> **Note**
>
> Stream Manager verwendet die `sizeThresholdForMultipartUploadBytes` Eigenschaft des Streams, um zu bestimmen, ob ein- oder mehrteiliger Upload nach Amazon S3 exportiert werden soll. Benutzerdefinierte 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

- Datenströme auf Greengrass-Kerngeräten verwalten
- Wird verwendet StreamManagerClient , um mit Streams zu arbeiten
- Konfigurationen für unterstützte AWS Cloud Ziele exportieren

Erstellen Sie benutzerdefinierte Komponenten, die den 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 unterAWS IoT Greengrass Komponenten erstellen.

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://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Java/1.0.0/StreamManagerS3.jar"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.StreamManagerS3Java
ComponentVersion: 1.0.0
ComponentDescription: Uses stream manager to upload a file to an S3 bucket.
ComponentPublisher: Amazon
ComponentDependencies:
```

```
  aws.greengrass.StreamManager:
    VersionRequirement: "^2.0.0"
Manifests:
  - Lifecycle:
      Run: java -jar {artifacts:path}/StreamManagerS3.jar
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Java/1.0.0/StreamManagerS3.jar
```

Weitere Informationen zum Entwickeln und Testen von Komponenten finden Sie unterAWS IoT Greengrass Komponenten erstellen.

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. *~/greengrass-components*Ersetzen Sie es 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://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip",
          "Unarchive": "ZIP"
        },
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py"
        },
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
```

```
      },
      "Lifecycle": {
        "install": "pip3 install --user -r {artifacts:path}/requirements.txt",
        "Run": "set \"PYTHONPATH=%PYTHONPATH%;{artifacts:decompressedPath}/
stream_manager_sdk\" & py -3 {artifacts:path}/stream_manager_s3.py"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip",
          "Unarchive": "ZIP"
        },
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py"
        },
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.StreamManagerS3Python
ComponentVersion: 1.0.0
ComponentDescription: Uses stream manager to upload a file to an S3 bucket.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.StreamManager:
    VersionRequirement: "^2.0.0"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: pip3 install --user -r {artifacts:path}/requirements.txt
      Run: |
```

```
        export PYTHONPATH=$PYTHONPATH:{artifacts:decompressedPath}/
stream_manager_sdk
        python3 {artifacts:path}/stream_manager_s3.py
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip
        Unarchive: ZIP
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt
  - Platform:
      os: windows
    Lifecycle:
      install: pip3 install --user -r {artifacts:path}/requirements.txt
      Run: |
        set "PYTHONPATH=%PYTHONPATH%;{artifacts:decompressedPath}/
stream_manager_sdk"
        py -3 {artifacts:path}/stream_manager_s3.py
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip
        Unarchive: ZIP
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt
```

Weitere Informationen zum Entwickeln und Testen von Komponenten finden Sie unterAWS IoT Greengrass Komponenten erstellen.

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. *~/greengrass-components* Ersetzen Sie es 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://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip",
          "Unarchive": "ZIP"
        },
        {
```

```
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "npm install {artifacts:decompressedPath}/stream-manager-sdk/
aws-greengrass-stream-manager-sdk",
        "Run": "set \"NODE_PATH=%NODE_PATH%;{work:path}/node_modules\" & node
 {artifacts:path}/index.js"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip",
          "Unarchive": "ZIP"
        },
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.StreamManagerS3JS
ComponentVersion: 1.0.0
ComponentDescription: Uses stream manager to upload a file to an S3 bucket.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.StreamManager:
    VersionRequirement: "^2.0.0"
Manifests:
  - Platform:
```

```
      os: linux
    Lifecycle:
      install: npm install {artifacts:decompressedPath}/stream-manager-sdk/aws-
greengrass-stream-manager-sdk
      Run: |
        export NODE_PATH=$NODE_PATH:{work:path}/node_modules
        node {artifacts:path}/index.js
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip
        Unarchive: ZIP
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js
  - Platform:
      os: windows
    Lifecycle:
      install: npm install {artifacts:decompressedPath}/stream-manager-sdk/aws-
greengrass-stream-manager-sdk
      Run: |
        set "NODE_PATH=%NODE_PATH%;{work:path}/node_modules"
        node {artifacts:path}/index.js
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip
        Unarchive: ZIP
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js
```

Weitere Informationen zum Entwickeln und Testen von Komponenten finden Sie unterAWS IoT Greengrass Komponenten erstellen.

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 unterWird verwendet StreamManagerClient , um mit Streams zu arbeiten.

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.
```

Wird verwendet StreamManagerClient , um mit Streams zu arbeiten

Benutzerdefinierte Greengrass-Komponenten, die auf dem Greengrass-Core-Gerät 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 der Stream Manager den Stream automatisch.

> **Note**
>
> In der Regel handelt es sich bei den Clients von Stream Manager um benutzerdefinierte Greengrass-Komponenten. Wenn Ihr Geschäftsszenario dies erfordert, können Sie auch zulassen, dass Prozesse, die keine Komponenten sind, die auf dem Greengrass-Kern ausgeführt werden (z. B. ein Docker-Container), mit dem Stream Manager interagieren. Weitere Informationen finden Sie unter the section called “Client-Authentifizierung”.

Die Auszüge in diesem Thema zeigen Ihnen, wie Clients `StreamManagerClient` Methoden aufrufen, um mit Streams zu arbeiten. Implementierungsdetails zu den Methoden und ihren Argumenten finden Sie unter den Links zur SDK-Referenz, die hinter jedem Codeausschnitt aufgeführt sind.

Wenn Sie den Stream-Manager in einer Lambda-Funktion verwenden, sollte Ihre Lambda-Funktion `StreamManagerClient` außerhalb des Funktionshandlers instanziiert werden. Wenn sie in dem Handler instanziiert wird, erstellt die Funktion bei jedem Aufruf eine `client` und eine Verbindung zum Stream-Manager.

> **Note**
>
> Wenn Sie `StreamManagerClient` in dem Handler 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 Methode create auf und übergibt ein `MessageStreamDefinition` Objekt. Dieses Objekt gibt den eindeutigen Namen für den Stream an und definiert, wie der Stream-Manager mit neuen Daten umgehen soll, wenn die maximale Streamgröße erreicht ist. Sie können mit `MessageStreamDefinition` seinen Datentypen (z. B. `ExportDefinition`, `StrategyOnFull`, und `Persistence`) andere Stream-Eigenschaften definieren. Dazu zählen:

- Das Ziel AWS IoT Analytics, Kinesis Data Streams und Amazon S3 S3-Ziele für automatische Exporte. AWS IoT SiteWise Weitere Informationen finden Sie unter the section called “Exportieren Sie Konfigurationen 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.
- Time-to-live (TTL). Die Zeitspanne, um sicherzustellen, dass die Streamdaten für die Verarbeitung verfügbar sind. Sie sollten sicherstellen, dass die Daten innerhalb dieses Zeitraums verbraucht werden können. Dies ist keine Löschrichtlinie. Die Daten werden möglicherweise nicht unmittelbar nach dem TTL-Zeitraum gelöscht.
- Streampersistenz. Wählen Sie, ob Streams im Dateisystem gespeichert werden sollen, um Daten über Core-Neustarts hinweg zu speichern oder Streams im Speicher zu speichern.
- Startsequenznummer. Geben Sie die Sequenznummer der Nachricht an, die als Startnachricht im Export verwendet werden soll.

Weitere Informationen zu `MessageStreamDefinition` finden Sie in der SDK-Referenz für Ihre Zielsprache:

- MessageStreamDefinitionim Java-SDK
- MessageStreamDefinitionim SDK von Node.js
- MessageStreamDefinitionim Python-SDK

> **Note**
>
> `StreamManagerClient`bietet auch ein Zielziel, mit dem Sie Streams auf einen HTTP-Server exportieren können. Dieses Ziel dient nur zu Testzwecken. Es ist weder stabil noch wird es 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 für die lokale Verarbeitung lesen. Die Anzahl der Streams, die Sie erstellen, hängt von Ihren Hardwarefunktionen und Ihrem Geschäftsfall ab. Eine Strategie besteht darin, für jeden Zielkanal im AWS IoT Analytics oder Kinesis-Datenstream einen Stream zu erstellen. Sie können jedoch mehrere Ziele für einen Stream definieren. Ein Stream hat eine dauerhafte Lebensdauer.

Voraussetzungen

Für diesen Vorgang gelten die folgenden Anforderungen:

- Mindestversion des Stream Manager-SDK: 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 `MessageStreamDefinition` und untergeordneten Datentypen.

Python

```
client = StreamManagerClient()

try:
    client.create_message_stream(MessageStreamDefinition(
        name="StreamName",    # Required.
        max_size=268435456,    # Default is 256 MB.
        stream_segment_size=16777216,    # Default is 16 MB.
        time_to_live_millis=None,    # By default, no TTL is enabled.
        strategy_on_full=StrategyOnFull.OverwriteOldestData,    # Required.
        persistence=Persistence.File,    # Default is File.
        flush_on_write=False,    # Default is false.
        export_definition=ExportDefinition(    # Optional. Choose where/how the
 stream is exported to the AWS Cloud.
```

```
            kinesis=None,
            iot_analytics=None,
            iot_sitewise=None,
            s3_task_executor=None
        )
    ))
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python-SDK-Referenz: [create_message_stream | MessageStreamDefinition]

Java

```
try (final StreamManagerClient client =
 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: | [createMessageStreamMessageStreamDefinition]

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
  try {
    await client.createMessageStream(
      new MessageStreamDefinition()
        .withName("StreamName") // Required.
        .withMaxSize(268435456)  // Default is 256 MB.
        .withStreamSegmentSize(16777216)  // Default is 16 MB.
        .withTimeToLiveMillis(null)  // By default, no TTL is enabled.
        .withStrategyOnFull(StrategyOnFull.OverwriteOldestData)  // Required.
        .withPersistence(Persistence.File)  // Default is File.
        .withFlushOnWrite(false)  // Default is false.
        .withExportDefinition(  // Optional. Choose where/how the stream is exported
 to the AWS Cloud.
          new ExportDefinition()
            .withKinesis(null)
            .withIotAnalytics(null)
            .withIotSiteWise(null)
            .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.
});
```

SDK-Referenz für Node.js: createMessageStream| MessageStreamDefinition

Weitere Informationen zur Konfiguration von Exportzielen finden Sie unterthe section called “Exportieren Sie Konfigurationen 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 Zielstream an. Das Exportziel bestimmt den Datentyp, der an diese Methode übergeben wird.

Voraussetzungen

Für diesen Vorgang gelten die folgenden Anforderungen:

- Mindestversion des Stream Manager-SDK: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Beispiele

AWS IoT Analytics oder Kinesis Data Streams Streams-Exportziele

Das folgende Snippet fügt eine Nachricht an den Stream namens `StreamName` an. Für AWS IoT Analytics unsere Kinesis Data Streams Streams-Ziele hängen Ihre Greengrass-Komponenten einen Datenblob an.

Für dieses Snippet gelten die folgenden Anforderungen:

- Mindestversion des Stream Manager-SDK: 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 Ziele exportieren

Das folgende Snippet fügt eine Nachricht an den Stream namens `StreamName` an. Für AWS IoT SiteWise Ziele hängen Ihre Greengrass-Komponenten ein `PutAssetPropertyValueEntry` serialisiertes Objekt an. Weitere Informationen finden Sie unter the section called “Exportieren nach AWS IoT SiteWise”.

> **Note**
>
> Wenn Sie Daten an senden AWS IoT SiteWise, müssen Ihre Daten die Anforderungen der Aktion erfüllen. `BatchPutAssetPropertyValue` Weitere Informationen finden Sie unter BatchPutAssetPropertyValue in der AWS IoT SiteWise -API-Referenz.

Für dieses Snippet gelten die folgenden Anforderungen:

- Mindestversion des Stream Manager-SDK: 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 S3-Exportziele

Das folgende Snippet hängt eine Exportaufgabe an den genannten Stream an. `StreamName` Für Amazon S3 S3-Ziele hängen Ihre Greengrass-Komponenten ein serialisiertes `S3ExportTaskDefinition` Objekt an, das Informationen über die Quell-Eingabedatei und das Amazon S3 S3-Zielobjekt enthält. Wenn das angegebene Objekt nicht existiert, erstellt Stream Manager es für Sie. Weitere Informationen finden Sie unter the section called “Exportieren nach Amazon S3”.

Für dieses Snippet gelten die folgenden Anforderungen:

- Mindestversion des Stream Manager-SDK: 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 | S3 ExportTaskDefinition

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    // Append an Amazon S3 export task definition and print the sequence number.
    S3ExportTaskDefinition s3ExportTaskDefinition = new S3ExportTaskDefinition()
        .withBucket("BucketName")
        .withKey("KeyName")
        .withInputUrl("URLToFile");
    long sequenceNumber = client.appendMessage("StreamName",
 ValidateAndSerialize.validateAndSerializeToJsonBytes(s3ExportTaskDefinition));
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java-SDK-Referenz: appendMessage | S3 ExportTaskDefinition

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
     // Append an Amazon S3 export task definition and print the sequence number.
     const taskDefinition = new S3ExportTaskDefinition()
        .withBucket("BucketName")
        .withKey("KeyName")
        .withInputUrl("URLToFile");
        const sequenceNumber = await client.appendMessage("StreamName",
 util.validateAndSerializeToJsonBytes(taskDefinition)));
    } catch (e) {
        // Properly handle errors.
```

```
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

SDK-Referenz für Node.js: appendMessage | S3 ExportTaskDefinition

Lesen von Nachrichten

Lesen Sie Nachrichten aus einem Stream.

Voraussetzungen

Für diesen Vorgang gelten die folgenden Anforderungen:

- Mindestversion des Stream Manager-SDK: 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

Ruft die Liste der Streams im Stream-Manager ab.

Voraussetzungen

Für diesen Vorgang gelten die folgenden Anforderungen:

- Mindestversion des Stream Manager-SDK: 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

Ruft Metadaten zu einem Stream ab, einschließlich der Stream-Definition, -Größe und des Exportstatus.

Voraussetzungen

Für diesen Vorgang gelten die folgenden Anforderungen:

- Mindestversion des Stream Manager-SDK: 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.
});
```

SDK-Referenz für Node.js: describeMessageStream

Nachrichtenstream aktualisieren

Aktualisieren Sie die Eigenschaften eines vorhandenen Streams. Möglicherweise möchten Sie einen Stream aktualisieren, wenn sich Ihre Anforderungen nach der Erstellung des Streams ändern. Zum Beispiel:

- Fügen Sie eine neue Exportkonfiguration für ein AWS Cloud Ziel hinzu.
- Erhöhen Sie die maximale Größe eines Streams, um zu ändern, wie Daten exportiert oder aufbewahrt werden. Beispielsweise kann die Stream-Größe in Kombination mit Ihrer Strategie bei vollen Einstellungen dazu führen, dass Daten gelöscht oder zurückgewiesen werden, bevor der Stream-Manager sie verarbeiten kann.
- Unterbrechen Sie Exporte und setzen Sie sie fort, z. B. wenn Exportaufgaben lange dauern und Sie Ihre Upload-Daten rationieren möchten.

Ihre Greengrass-Komponenten folgen diesem allgemeinen Prozess, um einen Stream zu aktualisieren:

1. Rufen Sie die Beschreibung des Streams ab.
2. Aktualisieren Sie die Zieleigenschaften der entsprechenden `MessageStreamDefinition` und untergeordneten Objekte.
3. Übergeben Sie das Aktualisierte`MessageStreamDefinition`. Stellen Sie sicher, dass Sie die vollständigen Objektdefinitionen für den aktualisierten Stream angeben. Undefinierte Eigenschaften werden auf die Standardwerte zurückgesetzt.

Sie können die Sequenznummer der Nachricht angeben, die als Startnachricht im Export verwendet werden soll.

Voraussetzungen

Für diesen Vorgang gelten die folgenden Anforderungen:

- Mindestversion des Stream Manager-SDK: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Beispiele

Das folgende Snippet aktualisiert den Stream mit dem Namen. `StreamName` Es aktualisiert mehrere Eigenschaften eines Streams, der in Kinesis Data Streams exportiert wird.

Python

```
client = StreamManagerClient()

try:
    message_stream_info = client.describe_message_stream(STREAM_NAME)
    message_stream_info.definition.max_size=536870912
    message_stream_info.definition.stream_segment_size=33554432
    message_stream_info.definition.time_to_live_millis=3600000
    message_stream_info.definition.strategy_on_full=StrategyOnFull.RejectNewData
    message_stream_info.definition.persistence=Persistence.Memory
    message_stream_info.definition.flush_on_write=False
    message_stream_info.definition.export_definition.kinesis=
        [KinesisConfig(
            # Updating Export definition to add a Kinesis Stream configuration.
            identifier=str(uuid.uuid4()), kinesis_stream_name=str(uuid.uuid4()))]
    client.update_message_stream(message_stream_info.definition)
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python-SDK-Referenz: updateMessageStream| MessageStreamDefinition

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    MessageStreamInfo messageStreamInfo = client.describeMessageStream(STREAM_NAME);
    // Update the message stream with new values.
    client.updateMessageStream(
        messageStreamInfo.getDefinition()
            .withStrategyOnFull(StrategyOnFull.RejectNewData) // Required. Updating
 Strategy on full to reject new data.
           // Max Size update should be greater than initial Max Size defined in
 Create Message Stream request
            .withMaxSize(536870912L) // Update Max Size to 512 MB.
            .withStreamSegmentSize(33554432L) // Update Segment Size to 32 MB.
            .withFlushOnWrite(true) // Update flush on write to true.
            .withPersistence(Persistence.Memory) // Update the persistence to
 Memory.
            .withTimeToLiveMillis(3600000L)    // Update TTL to 1 hour.
            .withExportDefinition(
                // Optional. Choose where/how the stream is exported to the AWS
 Cloud.
                messageStreamInfo.getDefinition().getExportDefinition().
                    // Updating Export definition to add a Kinesis Stream
 configuration.
                    .withKinesis(new ArrayList<KinesisConfig>() {{
                        add(new KinesisConfig()
                            .withIdentifier(EXPORT_IDENTIFIER)
                            .withKinesisStreamName("test"));
                        }})
            );
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java-SDK-Referenz: update_message_stream | MessageStreamDefinition

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const messageStreamInfo = await c.describeMessageStream(STREAM_NAME);
        await client.updateMessageStream(
            messageStreamInfo.definition
```

```
                // Max Size update should be greater than initial Max Size defined
 in Create Message Stream request
                .withMaxSize(536870912)    // Default is 256 MB. Updating Max Size
 to 512 MB.
                .withStreamSegmentSize(33554432)    // Default is 16 MB. Updating
 Segment Size to 32 MB.
                .withTimeToLiveMillis(3600000)    // By default, no TTL is enabled.
 Update TTL to 1 hour.
                .withStrategyOnFull(StrategyOnFull.RejectNewData)    // Required.
 Updating Strategy on full to reject new data.
                .withPersistence(Persistence.Memory)    // Default is File. Update
 the persistence to Memory
                .withFlushOnWrite(true)    // Default is false. Updating to true.
                .withExportDefinition(
                    // Optional. Choose where/how the stream is exported to the AWS
 Cloud.
                    messageStreamInfo.definition.exportDefinition
                        // Updating Export definition to add a Kinesis Stream
 configuration.
                        .withKinesis([new
 KinesisConfig().withIdentifier(uuidv4()).withKinesisStreamName(uuidv4())])
                )
        );
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

SDK-Referenz für Node.js: | updateMessageStreamMessageStreamDefinition

Einschränkungen für die Aktualisierung von Streams

Die folgenden Einschränkungen gelten für die Aktualisierung von Streams. Sofern in der folgenden Liste nicht anders angegeben, werden Aktualisierungen sofort wirksam.

- Sie können die Persistenz eines Streams nicht aktualisieren. Um dieses Verhalten zu ändern, löschen Sie den Stream und erstellen Sie einen Stream, der die neue Persistenzrichtlinie definiert.

- Sie können die maximale Größe eines Streams nur unter den folgenden Bedingungen aktualisieren:
  - Die maximale Größe muss größer oder gleich der aktuellen Größe des Streams sein. Um diese Informationen zu finden, beschreiben Sie den Stream und überprüfen Sie dann den Speicherstatus des zurückgegebenen `MessageStreamInfo` Objekts.
  - Die maximale Größe muss größer oder gleich der Segmentgröße des Streams sein.
- Sie können die Stream-Segmentgröße auf einen Wert aktualisieren, der unter der maximalen Größe des Streams liegt. Die aktualisierte Einstellung gilt für neue Segmente.
- Aktualisierungen der Eigenschaft Time to Live (TTL) gelten für neue Anfügevorgänge. Wenn Sie diesen Wert verringern, löscht Stream Manager möglicherweise auch vorhandene Segmente, die die TTL überschreiten.
- Aktualisierungen der Strategie für die vollständige Eigenschaft gelten auch für neue Anfügevorgänge. Wenn Sie die Strategie so einrichten, dass die ältesten Daten überschrieben werden, überschreibt Stream Manager möglicherweise auch bestehende Segmente, die auf der neuen Einstellung basieren.
- Aktualisierungen der Eigenschaft „Flush on Write“ gelten für neue Nachrichten.
- Aktualisierungen der Exportkonfigurationen gelten für neue Exporte. Die Aktualisierungsanforderung muss alle Exportkonfigurationen enthalten, die Sie unterstützen möchten. Andernfalls löscht der Stream Manager sie.
  - Wenn Sie eine Exportkonfiguration aktualisieren, geben Sie den Bezeichner der Ziel-Exportkonfiguration an.
  - Um eine Exportkonfiguration hinzuzufügen, geben Sie eine eindeutige Kennung für die neue Exportkonfiguration an.
  - Um eine Exportkonfiguration zu löschen, lassen Sie die Exportkonfiguration weg.
- Um die Startsequenznummer einer Exportkonfiguration in einem Stream zu aktualisieren, müssen Sie einen Wert angeben, der kleiner als die letzte Sequenznummer ist. Um diese Informationen zu finden, beschreiben Sie den Stream und überprüfen Sie dann den Speicherstatus des zurückgegebenen `MessageStreamInfo` Objekts.

Löschen eines Nachrichten-Streams

Löscht einen Stream. Wenn Sie einen Stream löschen, werden alle gespeicherten Daten für den Stream von der Festplatte gelöscht.

Voraussetzungen

Für diesen Vorgang gelten die folgenden Anforderungen:

- Mindestversion des Stream Manager-SDK: 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.
});
```

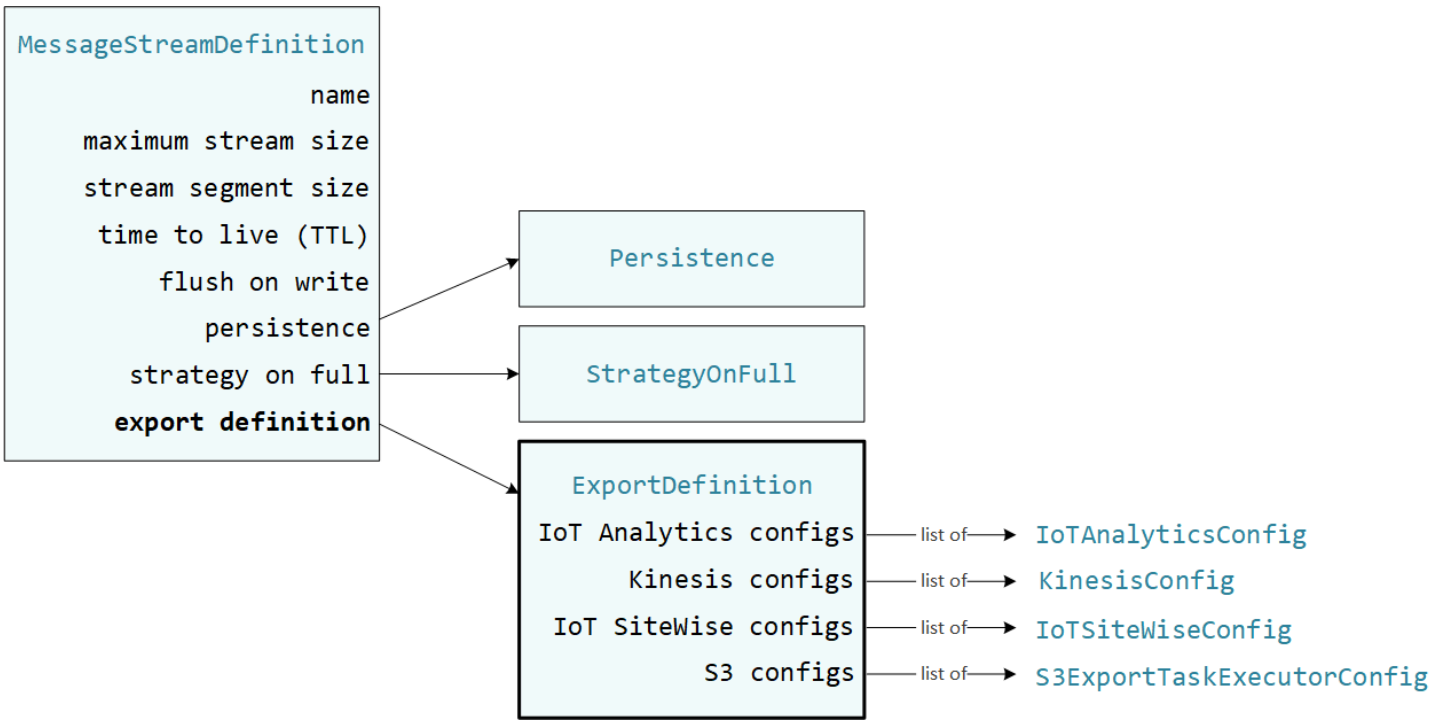
SDK-Referenz für Node.js: deleteMessageStream

Weitere Informationen finden Sie auch unter

- Datenströme auf Greengrass-Kerngeräten verwalten
- AWS IoT Greengrass Streammanager konfigurieren
- Konfigurationen für unterstützte AWS Cloud Ziele exportieren
- `StreamManagerClient`in der Stream Manager SDK-Referenz:
  - Python
  - Java
  - Node.js

Konfigurationen für unterstützte AWS Cloud Ziele exportieren

Benutzerdefinierte Greengrass-Komponenten werden `StreamManagerClient` im Stream Manager SDK verwendet, um mit dem Stream Manager zu interagieren. Wenn eine Komponente einen Stream erstellt oder einen Stream aktualisiert, übergibt sie ein `MessageStreamDefinition` Objekt, das die Stream-Eigenschaften darstellt, einschließlich der Exportdefinition. Das `ExportDefinition` Objekt enthält die für den Stream definierten Exportkonfigurationen. Stream Manager verwendet diese Exportkonfigurationen, um zu bestimmen, wo und wie der Stream exportiert werden soll.



Sie können keine oder mehr Exportkonfigurationen für einen Stream definieren, einschließlich mehrerer Exportkonfigurationen für einen einzelnen Zieltyp. Sie können beispielsweise einen Stream in zwei AWS IoT Analytics Kanäle und einen Kinesis-Datenstream exportieren.

Bei fehlgeschlagenen Exportversuchen versucht Stream Manager kontinuierlich, Daten in Intervallen von bis zu fünf Minuten zu exportieren. AWS Cloud Für die Anzahl der Wiederholungsversuche gibt es keine Obergrenze.

> **Note**
>
> `StreamManagerClient`bietet auch ein Zielziel, mit dem Sie Streams auf einen HTTP-Server exportieren können. Dieses Ziel dient nur zu Testzwecken. Es ist weder stabil noch wird es für die Verwendung in Produktionsumgebungen unterstützt.

Unterstützte AWS Cloud Ziele

- AWS IoT Analytics Kanäle
- Amazon Kinesis Kinesis-Datenströme
- AWS IoT SiteWise Eigenschaften von Vermögenswerten
- Amazon-S3-Objekte

Sie sind für die Wartung dieser AWS Cloud Ressourcen verantwortlich.

AWS IoT Analytics Kanäle

Stream Manager unterstützt automatische Exporte nach AWS IoT Analytics. AWS IoT Analytics ermöglicht es Ihnen, erweiterte Analysen Ihrer Daten durchzuführen, um Geschäftsentscheidungen zu treffen und Modelle für maschinelles Lernen zu verbessern. Weitere Informationen finden Sie unter Was ist AWS IoT Analytics? im AWS IoT Analytics Benutzerhandbuch.

Im 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:

- Io TAnalytics Config im Python-SDK
- Io TAnalytics Config im Java SDK
- Io TAnalytics Config im SDK Node.js

Voraussetzungen

Für dieses Exportziel gelten die folgenden Anforderungen:

- Die eingegebenen Zielkanäle AWS IoT Analytics müssen sich im selben AWS-Konto und AWS-Region wie das Greengrass-Core-Gerät befinden.
- Sie Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS müssen die `iotanalytics:BatchPutMessage` Erlaubnis erteilen, Kanäle anzusprechen. Beispiel:

  JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iotanalytics:BatchPutMessage"
      ],
      "Resource": [
        "arn:aws:iotanalytics:us-east-1:123456789012:channel/channel_1_name",
        "arn:aws:iotanalytics:us-east-1:123456789012:channel/channel_2_name"
      ]
```

```
    }
  ]
}
```

Sie können detaillierten oder bedingten Zugriff auf Ressourcen gewähren, indem Sie beispielsweise ein * Platzhalter-Benennungsschema verwenden. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter Hinzufügen und Entfernen von IAM-Richtlinien.

Exportieren nach AWS IoT Analytics

Um einen Stream zu erstellen, in den exportiert wird AWS IoT Analytics, erstellen Ihre Greengrass-Komponenten einen Stream mit einer Exportdefinition, die ein oder mehrere `IoTAnalyticsConfig` Objekte enthält. Dieses Objekt definiert Exporteinstellungen wie den Zielkanal, die Batchgröße, das Batch-Intervall und die Priorität.

Wenn Ihre Greengrass-Komponenten Daten von Geräten empfangen, hängen sie Nachrichten, die einen Datenblock enthalten, an den Zielstream an.

Anschließend exportiert der Stream Manager die Daten auf der Grundlage der Batch-Einstellungen und der Priorität, die in den Exportkonfigurationen des Streams definiert sind.

Amazon Kinesis Kinesis-Datenströme

Stream Manager unterstützt automatische Exporte nach Amazon Kinesis Data Streams. Kinesis Data Streams wird häufig verwendet, um große Datenmengen zu aggregieren und in ein Data Warehouse oder MapReduce Cluster zu laden. Weitere Informationen finden Sie unter Was ist Amazon Kinesis Data Streams? im Amazon Kinesis Developer Guide.

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:

- KinesisConfigim Python-SDK
- KinesisConfigim Java-SDK
- KinesisConfigim SDK von Node.js

Voraussetzungen

Für dieses Exportziel gelten die folgenden Anforderungen:

- Ziel-Streams in Kinesis Data Streams müssen sich im selben AWS-Konto und AWS-Region wie das Greengrass-Core-Gerät befinden.
- (Empfohlen) Stream Manager v2.2.1 verbessert die Leistung beim Exportieren von Streams zu Kinesis Data Streams Streams-Zielen. Um die Verbesserungen in dieser neuesten Version zu nutzen, aktualisieren Sie Ihre Stream Manager-Komponente auf v2.2.1 und verwenden Sie die `kinesis:ListShards` Richtlinie in der Greengrass-Token-Austauschrolle.
- Sie Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS müssen die `kinesis:PutRecords` Erlaubnis zum Targeting von Datenströmen gewähren. Beispiel:

  JSON

  ```
  {
    "Version":"2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "kinesis:PutRecords"
        ],
        "Resource": [
          "arn:aws:kinesis:us-east-1:123456789012:stream/stream_1_name",
          "arn:aws:kinesis:us-east-1:123456789012:stream/stream_2_name"
        ]
      }
    ]
  }
  ```

  Sie können detaillierten oder bedingten Zugriff auf Ressourcen gewähren, indem Sie beispielsweise ein * Platzhalter-Benennungsschema verwenden. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter Hinzufügen und Entfernen von IAM-Richtlinien.

In Kinesis Data Streams exportieren

Um einen Stream zu erstellen, der in Kinesis Data Streams exportiert wird, erstellen Ihre Greengrass-Komponenten einen Stream mit einer Exportdefinition, die ein oder mehrere `KinesisConfig` Objekte enthält. Dieses Objekt definiert Exporteinstellungen wie den Zieldatenstream, die Batchgröße, das Batch-Intervall und die Priorität.

Wenn Ihre Greengrass-Komponenten Daten von Geräten empfangen, hängen sie Nachrichten, die einen Datenblock enthalten, an den Zielstream an. Anschließend exportiert der Stream Manager die

Daten auf der Grundlage der Batch-Einstellungen und der Priorität, die in den Exportkonfigurationen des Streams definiert sind.

Stream Manager generiert eine eindeutige, zufällige UUID als Partitionsschlüssel für jeden Datensatz, der auf Amazon Kinesis hochgeladen wird.

AWS IoT SiteWise Eigenschaften von Vermögenswerten

Stream Manager unterstützt automatische Exporte nach AWS IoT SiteWise. AWS IoT SiteWise ermöglicht das Sammeln, Organisieren und Analysieren von Daten aus Industrieanlagen in großem Maßstab. Weitere Informationen finden Sie unter Was ist AWS IoT SiteWise? im AWS IoT SiteWise Benutzerhandbuch.

Im 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:

- Io TSite WiseConfig im Python-SDK
- Io TSite WiseConfig im Java-SDK
- Io TSite WiseConfig im SDK von Node.js

> **Note**
>
> AWS stellt auch AWS IoT SiteWise Komponenten bereit, die eine vorgefertigte Lösung bieten, mit der Sie Daten aus OPC-UA-Quellen streamen können. Weitere Informationen finden Sie unter SiteWise IoT-OPC-UA-Kollektor.

Voraussetzungen

Für dieses Exportziel gelten die folgenden Anforderungen:

- Die Eigenschaften des Zielobjekts in AWS IoT SiteWise müssen sich im selben AWS-Konto und AWS-Region wie das Greengrass-Core-Gerät befinden.

> **Note**
>
> Eine Liste der AWS IoT SiteWise unterstützten AWS-Region s finden Sie unter AWS IoT SiteWise Endpunkte und Kontingente in der AWS Allgemeinen Referenz.

- Sie Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS müssen die `iotsitewise:BatchPutAssetPropertyValue` Erlaubnis erteilen, auf Asset-Eigenschaften zu zielen. In der folgenden Beispielrichtlinie wird der `iotsitewise:assetHierarchyPath` Bedingungsschlüssel verwendet, um Zugriff auf ein Zielstamm-Asset und dessen untergeordnete Objekte zu gewähren. Sie können das `Condition` aus der Richtlinie entfernen, um Zugriff auf alle Ihre AWS IoT SiteWise Ressourcen zu gewähren oder einzelne Ressourcen anzugeben ARNs .

  JSON

  ```
  {
    "Version":"2012-10-17",
    "Statement": [
      {
         "Effect": "Allow",
         "Action": "iotsitewise:BatchPutAssetPropertyValue",
         "Resource": "*",
         "Condition": {
           "StringLike": {
             "iotsitewise:assetHierarchyPath": [
               "/root node asset ID",
               "/root node asset ID/*"
             ]
           }
         }
      }
    ]
  }
  ```

  Sie können detaillierten oder bedingten Zugriff auf Ressourcen gewähren, indem Sie beispielsweise ein * Platzhalter-Benennungsschema verwenden. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter Hinzufügen und Entfernen von IAM-Richtlinien.

  Wichtige Sicherheitsinformationen finden Sie im AWS IoT SiteWise Benutzerhandbuch unter BatchPutAssetPropertyValue Autorisierung.

Exportieren nach AWS IoT SiteWise

Um einen Stream zu erstellen, in den exportiert wird AWS IoT SiteWise, erstellen Ihre Greengrass-Komponenten einen Stream mit einer Exportdefinition, die ein oder mehrere `IoTSiteWiseConfig` Objekte enthält. Dieses Objekt definiert Exporteinstellungen wie Batchgröße, Batch-Intervall und Priorität.

Wenn Ihre Greengrass-Komponenten Asset-Eigenschaftsdaten von Geräten empfangen, hängen sie Nachrichten, die die Daten enthalten, an den Zielstream an. Nachrichten sind JSON-serialisierte `PutAssetPropertyValueEntry` Objekte, die Eigenschaftswerte für eine oder mehrere Asset-Eigenschaften enthalten. Weitere Informationen finden Sie unter Nachricht für Exportziele anhängen. AWS IoT SiteWise

> **Note**
>
> Wenn Sie Daten an senden AWS IoT SiteWise, müssen Ihre Daten die Anforderungen der `BatchPutAssetPropertyValue` Aktion erfüllen. Weitere Informationen finden Sie unter BatchPutAssetPropertyValue in der AWS IoT SiteWise -API-Referenz.

Anschließend exportiert Stream Manager die Daten auf der Grundlage der Batch-Einstellungen und der Priorität, die in den Exportkonfigurationen des Streams definiert sind.

Sie können Ihre Stream-Manager-Einstellungen und die Greengrass-Komponentenlogik anpassen, um Ihre Exportstrategie zu entwerfen. Beispiel:

- Für Exporte fast in Echtzeit sollten Sie niedrige Einstellungen für Batchgröße und Intervalle festlegen und die Daten an den Stream anhängen, wenn sie empfangen werden.
- Um die Batchverarbeitung zu optimieren, Bandbreitenbeschränkungen zu minimieren oder Kosten zu minimieren, können Ihre Greengrass-Komponenten die für eine einzelne Anlageneigenschaft empfangenen timestamp-quality-value (TQV) -Datenpunkte bündeln, bevor sie an den Stream angehängt werden. Eine Strategie besteht darin, Einträge für bis zu 10 verschiedene Kombinationen von Immobilien und Vermögenswerten oder Eigenschaftsaliasnamen in einer Nachricht zu bündeln, anstatt mehr als einen Eintrag für dieselbe Immobilie zu senden. Dies hilft dem Streammanager, die Kontingente einzuhalten.AWS IoT SiteWise

Amazon-S3-Objekte

Stream Manager unterstützt automatische Exporte nach Amazon S3. Sie können Amazon S3 verwenden, um große Datenmengen zu speichern und abzurufen. Weitere Informationen finden Sie unter Was ist Amazon S3? im Amazon Simple Storage Service Developer Guide.

Im 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:

- S3 ExportTaskExecutorConfig im Python-SDK
- S3 ExportTaskExecutorConfig im Java-SDK
- S3 ExportTaskExecutorConfig im SDK von Node.js

Voraussetzungen

Für dieses Exportziel gelten die folgenden Anforderungen:

- Die Amazon S3 S3-Ziel-Buckets müssen sich auf demselben Gerät AWS-Konto wie das Greengrass-Core-Gerät 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 Root-Dateisystem geschrieben werden und für die Stream-Manager-Komponente, die außerhalb des Containers ausgeführt wird, sichtbar sind.
- Wenn eine Docker-Container-Komponente Eingabedateien in ein Eingabedateiverzeichnis schreibt, müssen Sie das Verzeichnis als Volume im Container mit Schreibberechtigungen mounten. Dadurch wird sichergestellt, dass die Dateien in das Root-Dateisystem geschrieben werden und für die Stream-Manager-Komponente, die außerhalb des Containers ausgeführt wird, sichtbar sind.
- Sie Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS müssen die folgenden Berechtigungen für die Ziel-Buckets gewähren. Beispiel:

  JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
```

```
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:AbortMultipartUpload",
        "s3:ListMultipartUploadParts"
      ],
      "Resource": [
        "arn:aws:s3:::bucket-1-name/*",
        "arn:aws:s3:::bucket-2-name/*"
      ]
    }
  ]
}
```

Sie können detaillierten oder bedingten Zugriff auf Ressourcen gewähren, indem Sie beispielsweise ein * Platzhalter-Benennungsschema verwenden. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter Hinzufügen und Entfernen von IAM-Richtlinien.

Exportieren nach Amazon S3

Um einen Stream zu erstellen, der nach Amazon S3 exportiert, verwenden Ihre Greengrass-Komponenten das `S3ExportTaskExecutorConfig` Objekt, um die Exportrichtlinie zu konfigurieren. Die Richtlinie definiert Exporteinstellungen, wie z. B. den Schwellenwert und die Priorität für mehrteilige Uploads. Für Amazon S3 S3-Exporte lädt der Stream Manager Daten hoch, die er aus lokalen Dateien auf dem Kerngerät liest. Um einen Upload zu starten, hängen Ihre Greengrass-Komponenten eine Exportaufgabe an den Zielstream an. Die Exportaufgabe enthält Informationen über die Eingabedatei und das Amazon S3 S3-Zielobjekt. Stream Manager führt Aufgaben in der Reihenfolge aus, in der sie an den Stream angehängt werden.

> **Note**
>
> Der Ziel-Bucket muss bereits in Ihrem AWS-Konto vorhanden sein. Wenn ein Objekt für den angegebenen Schlüssel nicht existiert, erstellt Stream Manager das Objekt für Sie.

Stream Manager verwendet die Eigenschaft „Schwellenwert für mehrteilige Uploads“, die Einstellung für die Mindestgröße von Teilen und die Größe der Eingabedatei, um zu bestimmen, wie Daten hochgeladen werden. Der Schwellenwert für mehrteilige Uploads muss größer oder gleich der

Mindestgröße für Teile sein. Wenn Sie Daten parallel hochladen möchten, können Sie mehrere Streams erstellen.

Die Schlüssel, die Ihre Amazon S3 S3-Zielobjekte angeben, können gültige DateTimeFormatterJava-Zeichenfolgen in `!{timestamp:value}` Platzhaltern enthalten. Sie können diese Zeitstempel-Platzhalter verwenden, um Daten in Amazon S3 auf der Grundlage der Uhrzeit zu partitionieren, zu der die Eingabedateidaten hochgeladen wurden. Der folgende Schlüsselname wird beispielsweise in einen Wert wie aufgelöst. `my-key/2020/12/31/data.txt`

```
my-key/!{timestamp:YYYY}/!{timestamp:MM}/!{timestamp:dd}/data.txt
```

> **Note**
>
> Wenn Sie den Exportstatus für einen Stream überwachen möchten, erstellen Sie zuerst einen Status-Stream und konfigurieren Sie dann den Exportstream so, dass er ihn verwendet. Weitere Informationen finden Sie unter the section called “Überwachen Sie die Exportaufgaben”.

Eingabedaten verwalten

Sie können Code verfassen, den IoT-Anwendungen verwenden, um den Lebenszyklus der Eingabedaten zu verwalten. Der folgende Beispiel-Workflow zeigt, wie Sie Greengrass-Komponenten verwenden können, um diese Daten zu verwalten.

1. Ein lokaler Prozess empfängt Daten von Geräten oder Peripheriegeräten und schreibt die Daten dann in Dateien in einem Verzeichnis auf dem Kerngerät. Dies sind die Eingabedateien für den Stream-Manager.
2. Eine Greengrass-Komponente scannt das Verzeichnis und hängt eine Exportaufgabe an den Zielstream an, wenn eine neue Datei erstellt wird. Die Aufgabe ist ein JSON-serialisiertes `S3ExportTaskDefinition` Objekt, das die URL der Eingabedatei, den Amazon S3 S3-Ziel-Bucket und -Schlüssel sowie optionale Benutzermetadaten angibt.
3. Stream Manager liest die Eingabedatei und exportiert die Daten in der Reihenfolge der angehängten Aufgaben nach Amazon S3. Der Ziel-Bucket muss bereits in Ihrem AWS-Konto vorhanden sein. Wenn ein Objekt für den angegebenen Schlüssel nicht existiert, erstellt Stream Manager das Objekt für Sie.
4. Die Greengrass-Komponente liest Nachrichten aus einem Status-Stream, um den Exportstatus zu überwachen. Nachdem die Exportaufgaben abgeschlossen sind, kann die Greengrass-

Komponente die entsprechenden Eingabedateien löschen. Weitere Informationen finden Sie unter the section called “Überwachen Sie die Exportaufgaben”.

Überwachen Sie die Exportaufgaben

Sie können Code verfassen, den IoT-Anwendungen verwenden, um den Status Ihrer Amazon S3 S3-Exporte zu überwachen. Ihre Greengrass-Komponenten müssen einen Status-Stream erstellen und dann den Export-Stream so konfigurieren, dass Status-Updates in den Status-Stream geschrieben werden. Ein einziger Status-Stream kann Statusaktualisierungen von mehreren Streams empfangen, die nach Amazon S3 exportiert werden.

Erstellen Sie zunächst einen Stream, der als Status-Stream verwendet werden soll. Sie können die Größe und die Aufbewahrungsrichtlinien für den Stream konfigurieren, um die Lebensdauer der Statusmeldungen zu steuern. Beispiel:

- Stellen `Persistence` Sie `Memory` diese Option ein, wenn Sie die Statusmeldungen nicht speichern möchten.
- Auf `StrategyOnFull` einstellen, `OverwriteOldestData` damit neue Statusmeldungen nicht verloren gehen.

Erstellen oder aktualisieren Sie dann den Exportstream, um den Status-Stream zu verwenden. Legen Sie insbesondere die Statuskonfigurationseigenschaft der `S3ExportTaskExecutorConfig` Exportkonfiguration des Streams fest. Diese Einstellung weist den Stream-Manager an, Statusmeldungen über die Exportaufgaben in den Status-Stream zu schreiben. Geben Sie im `StatusConfig` Objekt den Namen des Status-Streams und den Grad der Ausführlichkeit an. Die folgenden unterstützten Werte reichen von am wenigsten ausführlich () bis zu den ausführlichsten Werten (`ERROR`). `TRACE` Der Standardwert ist `INFO`.

- `ERROR`
- `WARN`
- `INFO`
- `DEBUG`
- `TRACE`

Der folgende Beispiel-Workflow zeigt, wie Greengrass-Komponenten einen Status-Stream verwenden könnten, 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 er Statusmeldungen über Exportaufgaben in einen Status-Stream schreibt. Die Anfügeoperation gibt eine Sequenznummer zurück, die die Aufgaben-ID darstellt.
2. Eine Greengrass-Komponente liest Nachrichten sequentiell aus dem Status-Stream und filtert die Nachrichten dann basierend auf dem Streamnamen und der Task-ID oder basierend auf einer Exportaufgabeneigenschaft aus dem Nachrichtenkontext. Die Greengrass-Komponente kann beispielsweise nach der Eingabedatei-URL der Exportaufgabe filtern, die durch das `S3ExportTaskDefinition` Objekt im Nachrichtenkontext dargestellt wird.

   Die folgenden Statuscodes geben an, dass eine Exportaufgabe den Status „Abgeschlossen“ erreicht hat:

   - `Success`. Der Upload wurde erfolgreich abgeschlossen.
   - `Failure`. Stream Manager ist auf einen Fehler gestoßen, z. B. ist der angegebene Bucket nicht vorhanden. Nachdem Sie das Problem behoben haben, können Sie die Exportaufgabe erneut an den Stream anhängen.
   - `Canceled`. Die Aufgabe wurde beendet, weil die Stream- oder Exportdefinition gelöscht wurde oder der Zeitraum time-to-live (TTL) der Aufgabe abgelaufen ist.

   > **Note**
   >
   > Die Aufgabe kann auch den Status `InProgress` oder `Warning` haben. Stream Manager gibt Warnungen aus, wenn ein Ereignis einen Fehler zurückgibt, der die Ausführung der Aufgabe nicht beeinträchtigt. Wenn beispielsweise ein teilweiser Upload nicht bereinigt werden kann, wird eine Warnung zurückgegeben.

3. Nachdem die Exportaufgaben abgeschlossen sind, kann die Greengrass-Komponente die entsprechenden Eingabedateien löschen.

Das folgende Beispiel zeigt, wie eine Greengrass-Komponente Statusmeldungen lesen und verarbeiten könnte.

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

Durchführen von Machine Learning-Inferenzen

Mit AWS IoT Greengrass können Sie mithilfe von Cloud-trainierten Modellen Inferenzen für maschinelles Lernen (ML) auf Ihren Edge-Geräten anhand lokal generierter 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 zur Durchführung von Inferenzen erforderlichen Schritte effizienter. Sie können Ihre Inferenzmodelle überall trainieren und sie lokal als Komponenten für maschinelles Lernen einsetzen. Sie können beispielsweise Deep-Learning-Modelle in Amazon SageMaker AI erstellen und trainieren. Anschließend können Sie diese Modelle in einem Amazon S3 S3-Bucket speichern, sodass Sie diese Modelle als Artefakte in Ihren Komponenten verwenden können, um Inferenzen auf Ihren Kerngeräten durchzuführen.

Themen

- Wie funktioniert AWS IoT Greengrass ML-Inferenz
- Was ist in AWS IoT Greengrass Version 2 anders?
- Voraussetzungen
- Unterstützte Modellquellen
- Unterstützte Laufzeiten für maschinelles Lernen
- AWS-bereitgestellte Komponenten für maschinelles Lernen
- Verwenden Sie Amazon SageMaker AI Edge Manager auf Greengrass-Kerngeräten
- Passen Sie Ihre Komponenten für maschinelles Lernen an
- Fehlerbehebung bei Inferenz für maschinelles Lernen

Wie funktioniert AWS IoT Greengrass ML-Inferenz

AWS bietet Komponenten für maschinelles Lernen, mit denen Sie Bereitstellungen in einem Schritt erstellen können, um Inferenzen für maschinelles Lernen 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 Komponenten für maschinelles Lernen:

- Modellkomponente — Enthält Modelle für maschinelles Lernen als Greengrass-Artefakte.
- Runtime-Komponente — Enthält das Skript, das das Machine Learning-Framework und seine Abhängigkeiten auf dem Greengrass-Core-Gerät installiert.
- Inferenzkomponente — Enthält den Inferenzcode und enthält Komponentenabhängigkeiten zur Installation des Frameworks für maschinelles Lernen und zum Herunterladen vorab trainierter Modelle für maschinelles Lernen.

Jede Bereitstellung, die Sie für die Durchführung von Inferenzen für maschinelles Lernen erstellen, besteht aus mindestens einer Komponente, die Ihre Inferenzanwendung ausführt, das Framework für maschinelles Lernen installiert und Ihre Modelle für maschinelles Lernen herunterlädt. Um eine Beispielinferenz mit von AWS-bereitgestellten Komponenten durchzuführen, stellen Sie eine Inferenzkomponente auf Ihrem Kerngerät bereit, die automatisch das entsprechende Modell und die Laufzeitkomponenten als Abhängigkeiten enthält. Um Ihre Bereitstellungen anzupassen, können Sie die Beispielmodellkomponenten durch benutzerdefinierte Modellkomponenten ergänzen oder austauschen oder Sie können die Komponentenrezepte für die AWS bereitgestellten Komponenten als Vorlagen verwenden, um Ihre eigenen benutzerdefinierten Inferenz-, Modell- und Laufzeitkomponenten zu erstellen.

So führen Sie mithilfe benutzerdefinierter Komponenten Inferenzen für maschinelles Lernen durch:

1. Erstellen Sie eine Modellkomponente. Diese Komponente enthält die Modelle für maschinelles Lernen, die Sie für die Durchführung von Inferenzen verwenden möchten. AWS enthält Beispiele für vortrainierte DLR- und TensorFlow Lite-Modelle. Um ein benutzerdefiniertes Modell zu verwenden, erstellen Sie Ihre eigene Modellkomponente.
2. Erstellen Sie eine Runtime-Komponente. Diese Komponente enthält die Skripts, die zur Installation der Laufzeit für maschinelles Lernen für Ihre Modelle erforderlich sind. AWS bietet Beispiel-Laufzeitkomponenten 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 sowie Ihre Modell- und Laufzeitkomponenten als Abhängigkeiten. AWS bietet Beispiele für Inferenzkomponenten für die Bildklassifizierung und Objekterkennung mit DLR und Lite. TensorFlow Um andere Arten von Inferenz 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 sie AWS IoT Greengrass auch automatisch die Modell- und Laufzeitkomponentenabhängigkeiten bereit.

Informationen zu den ersten Schritten mit den von AWS-bereitgestellten Komponenten finden Sie unter. the section called “Führen Sie eine Inferenz zur Klassifizierung von Beispielbildern durch”

Informationen zum Erstellen benutzerdefinierter Komponenten für maschinelles Lernen finden Sie unterPassen Sie Ihre Komponenten für maschinelles Lernen an.

Was ist in AWS IoT Greengrass Version 2 anders?

AWS IoT Greengrass konsolidiert Funktionseinheiten für maschinelles Lernen — wie Modelle, Laufzeiten und Inferenzcode — in Komponenten, mit denen Sie in einem einzigen Schritt die Machine Learning-Runtime installieren, Ihre trainierten Modelle herunterladen und Inferenzen auf Ihrem Gerät durchführen können.

Durch die Verwendung der AWS bereitgestellten Komponenten für maschinelles Lernen haben Sie die Flexibilität, mit der Durchführung von Inferenzen für maschinelles Lernen mit Beispiel-Inferenzcode und vortrainierten Modellen zu beginnen. Sie können benutzerdefinierte Modellkomponenten einbinden, um Ihre eigenen, speziell trainierten Modelle mit den Inferenz- und Laufzeitkomponenten zu verwenden, die das bietet. AWS Für eine vollständig angepasste Lösung für maschinelles Lernen können Sie die öffentlichen Komponenten als Vorlagen verwenden, um benutzerdefinierte Komponenten zu erstellen und jeden beliebigen Laufzeit-, Modell- oder Inferenztyp zu verwenden.

Voraussetzungen

Um Komponenten für maschinelles Lernen 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 der AWS bereitgestellten Beispielkomponenten für maschinelles Lernen.

Unterstützte Modellquellen

AWS IoT Greengrass unterstützt die Verwendung von individuell trainierten Modellen für maschinelles Lernen, die in Amazon S3 gespeichert sind. Sie können Amazon SageMaker AI Edge Packaging Jobs auch verwenden, um direkt Modellkomponenten für Ihre mit SageMaker KI NEO kompilierten Modelle zu erstellen. Informationen zur Verwendung von SageMaker AI Edge Manager mit finden Sie AWS IoT Greengrass unterVerwenden Sie Amazon SageMaker AI Edge Manager auf Greengrass-Kerngeräten.

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 maschinelles Lernen derzeit nur die Verschlüsselungsoptionen SSE-S3 oder SSE-KMS. Weitere Informationen zu serverseitigen Verschlüsselungsoptionen finden Sie unter Schützen von Daten mithilfe serverseitiger Verschlüsselung im Amazon Simple Storage Service-Benutzerhandbuch.
- Ihre Namen dürfen keine Punkte () `.` enthalten. Weitere Informationen finden Sie in der Regel zur Verwendung von Buckets im virtuellen Hosted-Stil mit SSL unter Regeln für die Bucket-Benennung im Amazon Simple Storage Service-Benutzerhandbuch.
- Die S3-Buckets, in denen Ihre Modellquellen gespeichert sind, müssen sich im selben AWS-Konto und AWS-Region wie Ihre maschinellen Lernkomponenten befinden.
- AWS IoT Greengrass muss über eine `read` Genehmigung für die Modellquelle verfügen. Um den Zugriff auf die S3-Buckets AWS IoT Greengrass zu ermöglichen, 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 Laufzeiten für maschinelles Lernen

AWS IoT Greengrass ermöglicht es Ihnen, benutzerdefinierte Komponenten zu erstellen, um eine beliebige Laufzeit für maschinelles Lernen Ihrer Wahl zu verwenden, um mit Ihren individuell trainierten Modellen Inferenzen für maschinelles Lernen durchzuführen. Informationen zum Erstellen benutzerdefinierter Komponenten für maschinelles Lernen finden Sie unter. Passen Sie Ihre Komponenten für maschinelles Lernen an

Um den Einstieg in maschinelles Lernen effizienter zu gestalten, AWS IoT Greengrass bietet dieser Artikel Beispiele für Inferenz-, Modell- und Laufzeitkomponenten, die die folgenden Laufzeiten für maschinelles Lernen verwenden:

- Deep Learning Runtime (DLR) v1.6.0 und v1.3.0
- TensorFlow Lite v2.5.0

AWS-bereitgestellte Komponenten für maschinelles Lernen

In der folgenden Tabelle sind die von AWS-bereitgestellten Komponenten aufgeführt, die für maschinelles Lernen verwendet werden.

> **Note**
>
> Einige der AWS bereitgestellten Komponenten hängen von bestimmten Nebenversionen des Greengrass-Kerns ab. Aufgrund dieser Abhängigkeit müssen Sie diese Komponenten aktualisieren, wenn Sie den Greengrass Nucleus auf eine neue Nebenversion aktualisieren. Informationen zu den spezifischen Versionen von Nucleus, von denen jede Komponente abhängt, finden Sie im entsprechenden Komponententhema. Weitere Informationen zur Aktualisierung des Nucleus finden Sie unterAktualisieren Sie die AWS IoT Greengrass Core-Software (OTA).

Komponente	Description	Art der Komponente	Unterstützes Betriebssystem	Open-Source
SageMaker KI-Edge-Manager	Stellt den Amazon SageMaker AI Edge Manager-Agenten auf dem Greengrass-Core-Gerät bereit.	Generisch	Linux, Windows	Nein

Komponente	Description	Art der Komponente	Unterstützes Betriebssystem	Open-Source
DLR-Bildklassifizierung	Inferenzkomponente, die den DLR-Bildklassifizierungsmodellspeicher und die DLR-Laufzeitkomponente als Abhängigkeiten verwendet, um DLR zu installieren, Beispielmodelle für die Bildklassifizierung herunterzuladen und Inferenzen zur Bildklassifizierung auf unterstützten Geräten durchzuführen.	Generisch	Linux, Windows	Nein

Komponente	Description	Art der Komponente	Unterstützes Betriebssystem	Open-Source
DLR-Objekterkennung	Inferenzkomponente, die den DLR-Objekterkennungsmodellspeicher und die DLR-Laufzeitkomponente als Abhängigkeiten verwendet, um DLR zu installieren, Beispielmodelle zur Objekterkennung herunterzuladen und Objekterkennungsinferenzen auf unterstützten Geräten durchzuführen.	Generisch	Linux, Windows	Nein

Komponente	Description	Art der Komponente	Unterstützes Betriebssystem	Open-Source
Modellspeicher für die DLR-Bildklassifizierung	Modellkomponente, die Stichprobe ResNet -50 Bildklassifizierungsmodelle als Greengrass-Artefakte enthält.	Generisch	Linux, Windows	Nein
Modellspeicher für DLR-Objekterkennung	Modellkomponente, die Beispielmodelle zur YOLOv3 Objekterkennung als Greengrass-Artefakte enthält.	Generisch	Linux, Windows	Nein

Komponente	Description	Art der Komponente	Unterstützes Betriebssystem	Open-Source
DLR-Laufzeit	Runtime-Komponente, die ein Installationsskript enthält, das zur Installation von DLR und seinen Abhängigkeiten auf dem Greengrass-Core-Gerät verwendet wird.	Generisch	Linux, Windows	Nein

Komponente	Description	Art der Komponente	Unterstützes Betriebssystem	Open-Source
TensorFlow Lite-Bildklassifizierung	Inferenzkomponente, die den TensorFlow Lite-Bildklassifizierungsmodellspeicher und die TensorFlow Lite-Runtime-Komponente als Abhängigkeiten verwendet, um TensorFlow Lite zu installieren, Beispielmodelle für die Bildklassifizierung herunterzuladen und Inferenzen zur Bildklassifizierung auf unterstützten Geräten durchzuführen.	Generisch	Linux, Windows	Nein

Komponente	Description	Art der Komponente	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, Beispielmodelle für die Objekterkennung herunterzuladen und Objekterkennungsinferenzen auf unterstützten Geräten durchzuführen.	Generisch	Linux, Windows	Nein

Komponente	Description	Art der Komponente	Unterstützes Betriebssystem	Open-Source
TensorFlow Modellspeicher für Lite-Bildklassifizierung	Modellkomponente, die ein Beispiel für ein MobileNet v1-Modell als Greengrass-Artefakt enthält.	Generisch	Linux, Windows	Nein
TensorFlow Modellspeicher für Lite-Objekterkennung	Modellkomponente, die ein SSD-Beispielmodell (Single Shot Detection) als MobileNet Greengrass-Artefakt enthält.	Generisch	Linux, Windows	Nein

Komponente	Description	Art der Komponente	Unterstützes Betriebssystem	Open-Source
TensorFlow Lite-Laufzeit	Runtime-Komponente, die ein Installationsskript enthält, das zur Installation von TensorFlow Lite und seinen Abhängigkeiten auf dem Greengrass-Core-Gerät verwendet wird.	Generisch	Linux, Windows	Nein

Verwenden Sie Amazon SageMaker AI Edge Manager auf Greengrass-Kerngeräten

> **Important**
>
> SageMaker AI Edge Manager wurde am 26. April 2024 eingestellt. Weitere Informationen zur weiteren Bereitstellung Ihrer Modelle auf Edge-Geräten finden Sie unter Ende der Nutzungsdauer von SageMaker AI Edge Manager.

Amazon SageMaker AI Edge Manager ist ein Softwareagent, der auf Edge-Geräten ausgeführt wird. SageMaker AI Edge Manager bietet Modellverwaltung für Edge-Geräte, sodass Sie mit Amazon SageMaker AI Neo kompilierte Modelle direkt auf Greengrass-Core-Geräten verpacken und verwenden können. Mithilfe von SageMaker AI Edge Manager können Sie auch Modelleingabe-

und -ausgabedaten von Ihren Kerngeräten abfragen und diese Daten AWS Cloud zur Überwachung und Analyse an die senden. Da SageMaker AI Edge Manager SageMaker AI Neo verwendet, um Ihre Modelle für Ihre Zielhardware zu optimieren, müssen Sie die DLR-Laufzeit nicht direkt auf Ihrem Gerät installieren. Auf Greengrass-Geräten lädt SageMaker AI Edge Manager keine lokalen AWS IoT Zertifikate und ruft den Endpunkt des AWS IoT Anmeldeinformationsanbieters nicht direkt auf. Stattdessen verwendet SageMaker AI Edge Manager den Token-Austauschdienst, um temporäre Anmeldeinformationen von einem TES-Endpunkt abzurufen.

In diesem Abschnitt wird beschrieben, wie SageMaker AI Edge Manager auf Greengrass-Kerngeräten funktioniert.

So funktioniert SageMaker AI Edge Manager auf Greengrass-Geräten

Um den SageMaker AI Edge Manager-Agenten auf Ihren Kerngeräten 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 Agent-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 AI Edge Manager mit verwenden AWS IoT Greengrass, umfasst Ihr Workflow die folgenden allgemeinen Schritte:

1. Kompilieren Sie Modelle mit SageMaker AI Neo.
2. Verpacken Sie Ihre mit SageMaker KI Neo kompilierten Modelle mithilfe von SageMaker AI Edge-Paketierungsjobs. Wenn Sie einen Edge-Paketierungsauftrag für Ihr Modell ausführen, können Sie wählen, ob Sie eine Modellkomponente mit dem verpackten Modell als Artefakt erstellen möchten, das auf Ihrem Greengrass-Core-Gerät bereitgestellt werden kann.
3. Erstellen Sie eine benutzerdefinierte Inferenzkomponente. Sie verwenden diese Inferenzkomponente, um mit dem Edge Manager-Agenten zu interagieren und Inferenzen auf dem Kerngerät durchzuführen. Zu diesen Vorgängen gehören das Laden von Modellen, das Aufrufen von Vorhersageanforderungen zur Ausführung von Inferenzen und das Entladen von Modellen, wenn die Komponente heruntergefahren wird.
4. Stellen Sie die SageMaker AI Edge Manager-Komponente, die verpackte Modellkomponente und die Inferenzkomponente bereit, um Ihr Modell auf der SageMaker KI-Inferenz-Engine (Edge Manager-Agent) auf Ihrem Gerät auszuführen.

Weitere Informationen zum Erstellen von Edge-Packaging-Jobs und Inferenzkomponenten, die mit SageMaker AI Edge Manager funktionieren, finden Sie unter Deploy Model Package und Edge Manager Agent with AWS IoT Greengrass im Amazon SageMaker AI Developer Guide.

Das Tutorial: Erste Schritte mit SageMaker AI Edge Manager Tutorial zeigt Ihnen, wie Sie den SageMaker AI Edge Manager-Agenten auf einem vorhandenen AWS Greengrass-Core-Gerät einrichten und verwenden. Dabei wird der bereitgestellte Beispielcode verwendet, mit dem Sie Beispielinferenz- und Modellkomponenten erstellen können.

Wenn Sie SageMaker AI Edge Manager auf Greengrass-Core-Geräten verwenden, können Sie auch die Funktion zum Erfassen von Daten verwenden, um Beispieldaten auf den AWS Cloud hochzuladen. Capture Data ist eine SageMaker KI-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 AI Edge Manager finden Sie unter Modell verwalten im Amazon SageMaker AI Developer Guide.

Voraussetzungen

Sie müssen die folgenden Anforderungen erfüllen, um den SageMaker AI Edge Manager-Agenten auf Greengrass-Kerngeräten 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) 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 AI 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 KI-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 AI Edge-Geräteflotte registriert. Der Name des Edge-Geräts muss mit dem AWS IoT Dingnamen für Ihr Kerngerät übereinstimmen. Weitere Informationen finden Sie unter Registrieren Sie Ihr Greengrass Core-Gerät.

Beginnen Sie mit SageMaker AI Edge Manager

Sie können ein Tutorial abschließen, um mit SageMaker AI Edge Manager zu beginnen. Das Tutorial zeigt Ihnen, wie Sie mit der Verwendung von SageMaker AI Edge Manager beginnen können, und AWS zwar anhand der bereitgestellten Beispielkomponenten auf einem vorhandenen Kerngerät. Diese Beispielkomponenten verwenden die SageMaker AI Edge Manager-Komponente als Abhängigkeit, um den Edge Manager-Agenten bereitzustellen und Inferenzen mithilfe vortrainierter Modelle durchzuführen, die mit AI Neo kompiliert wurden. SageMaker Weitere Informationen finden Sie unter Tutorial: Erste Schritte mit SageMaker AI Edge Manager.

Passen Sie Ihre Komponenten für maschinelles Lernen an

In können Sie Beispielkomponenten für maschinelles Lernen konfigurieren AWS IoT Greengrass, um die Art und Weise, wie Sie Inferenzen für maschinelles Lernen auf Ihren Geräten durchführen, mit den Komponenten Inferenz, Modell und Laufzeit als Bausteinen anzupassen. AWS IoT Greengrass bietet Ihnen außerdem die Flexibilität, die Beispielkomponenten als Vorlagen zu verwenden und nach Bedarf Ihre eigenen benutzerdefinierten Komponenten zu erstellen. Sie können diesen modularen Ansatz beliebig kombinieren, um Ihre Inferenzkomponenten für maschinelles Lernen auf folgende Weise individuell anzupassen:

Verwenden von Beispiel-Inferenzkomponenten

- Ändern Sie die Konfiguration der Inferenzkomponenten, wenn Sie sie bereitstellen.
- Verwenden Sie ein benutzerdefiniertes Modell mit der Beispiel-Inferenzkomponente, indem Sie die Mustermodellspeicherkomponente 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 Komponenten des öffentlichen Modells 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 bereitgestellt wird.

Themen

- Ändern Sie die Konfiguration einer öffentlichen Inferenzkomponente
- Verwenden Sie ein benutzerdefiniertes Modell mit der Beispiel-Inferenzkomponente
- Erstellen Sie benutzerdefinierte Komponenten für maschinelles Lernen
- Erstellen Sie eine benutzerdefinierte Inferenzkomponente

Ändern Sie die Konfiguration einer öffentlichen Inferenzkomponente

In der AWS IoT Greengrass Konsole wird auf der Komponentenseite die Standardkonfiguration dieser Komponente angezeigt. Die Standardkonfiguration der TensorFlow Lite-Komponente zur Bildklassifizierung 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. AWS-bereitgestellte Komponenten für maschinelles Lernen

In diesem Abschnitt wird beschrieben, wie Sie eine modifizierte Komponente von der AWS IoT Greengrass Konsole aus bereitstellen. Hinweise zur Bereitstellung von Komponenten mithilfe von finden Sie unterErstellen von Bereitstellungen. AWS CLI

So stellen Sie eine modifizierte öffentliche Inferenzkomponente (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 Komponente aus, die Sie bereitstellen möchten.
4. Wählen Sie auf der Komponentenseite die Option 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. Vergewissern Sie sich auf der Seite Komponenten auswählen unter Öffentliche Komponenten, dass 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 anschließend Komponente konfigurieren aus.
   b. Geben Sie unter Konfigurationsupdate 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, nach dem `custom.jpg` `/custom-ml-inference/images/` im Ordner genannten Bild 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 `""` in das Feld Pfade zurücksetzen ein.

c. Wählen Sie Bestätigen aus, und wählen Sie dann Weiter.

9. Behalten Sie auf der Seite Erweiterte Einstellungen konfigurieren die Standardkonfigurationseinstellungen bei und wählen Sie Weiter.
10. Wählen Sie auf der Seite „Überprüfen“ die Option Bereitstellen

Verwenden Sie ein benutzerdefiniertes Modell 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 bereitgestellt wird, müssen Sie die Komponenten des öffentlichen Modells durch Komponenten überschreiben, die diese Modelle als Artefakte verwenden. Auf hoher Ebene führen Sie 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. Hinweise zur Aktualisierung der Konfiguration der Inferenzkomponente finden Sie unter Ändern Sie die Konfiguration einer öffentlichen Inferenzkomponente

Sucht bei der Bereitstellung der Inferenzkomponente AWS IoT Greengrass nach der neuesten Version ihrer Komponentenabhängigkeiten. Sie überschreibt die abhängige öffentliche

Modellkomponente, wenn eine spätere benutzerdefinierte Version der Komponente im selben AWS-Konto und vorhanden ist. AWS-Region

Erstellen Sie eine benutzerdefinierte 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 S3-Buckets im Amazon Simple Storage Service-Benutzerhandbuch.

   > **Note**
   >
   > Sie müssen Ihre Artefakte in S3-Buckets speichern, die sich im selben AWS-Konto und AWS-Region wie die Komponenten befinden. Um auf diese Artefakte zugreifen AWS IoT Greengrass zu können, muss die Greengrass-Geräterolle die `s3:GetObject` Aktion zulassen. Weitere Informationen zur Geräterolle finden Sie unterAutorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

2. Wählen Sie im Navigationsmenü der AWS IoT Greengrass Konsole die Option Komponenten aus.
3. Rufen Sie das Komponentenrezept für die Public Model Store-Komponente 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 die Option Rezept anzeigen aus 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 eingeben als Komponentenquelle aus.
6. Fügen Sie in das Feld Rezept das Komponentenrezept ein, das Sie zuvor kopiert haben.
7. Aktualisieren Sie im Rezept die folgenden Werte:

   - `ComponentVersion`: Erhöhen Sie die Nebenversion der Komponente.

     Wenn Sie eine benutzerdefinierte Komponente erstellen, um eine Komponente eines öffentlichen Modells zu überschreiben, müssen Sie nur die Nebenversion der vorhandenen Komponentenversion aktualisieren. Wenn es sich bei der öffentlichen Komponentenversion

beispielsweise um eine Version handelt`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 S3-URI des Modells, das Sie verwenden möchten.

> **Note**
>
> Ändern Sie nicht den Namen der Komponente.

8. Wählen Sie Komponente erstellen.

Erstellen Sie eine benutzerdefinierte 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 S3-Buckets im Amazon Simple Storage Service-Benutzerhandbuch.

> **Note**
>
> Sie müssen Ihre Artefakte in S3-Buckets speichern, die sich im selben AWS-Konto und AWS-Region wie die Komponenten befinden. Um auf diese Artefakte zugreifen AWS IoT Greengrass zu können, muss die Greengrass-Geräterolle die `s3:GetObject` Aktion zulassen. Weitere Informationen zur Geräterolle finden Sie unterAutorisieren 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. Ändern Sie den Namen der Rezeptdatei auf*<component-name>*-*<component-version>*, wobei 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 Komponente eines öffentlichen Modells zu überschreiben, müssen Sie nur die Nebenversion der vorhandenen Komponentenversion aktualisieren. Wenn es sich bei der öffentlichen Komponentenversion beispielsweise um eine Version handelt`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 S3-URI des Modells, das Sie verwenden möchten.

   > **Note**
   >
   > Ändern Sie nicht den Namen der Komponente.

5. Führen Sie den folgenden Befehl aus, um mit dem abgerufenen und geänderten Rezept eine neue Komponente zu erstellen.

```
aws greengrassv2 create-component-version \
    --inline-recipe fileb://path/to/component/recipe
```

> **Note**
>
> In diesem Schritt wird die Komponente im AWS IoT Greengrass Dienst in der erstellt AWS 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 AWS IoT Greengrass Komponenten entwickeln.

Weitere Informationen zum Erstellen von Komponenten finden Sie unterAWS IoT Greengrass Komponenten entwickeln.

Erstellen Sie benutzerdefinierte Komponenten für maschinelles Lernen

Sie müssen benutzerdefinierte Komponenten erstellen, wenn Sie benutzerdefinierten Inferenzcode oder eine Runtime verwenden möchten, für die AWS IoT Greengrass keine Beispielkomponente bereitgestellt wird. Sie können Ihren benutzerdefinierten Inferenzcode mit den AWS bereitgestellten Beispielmodellen und Laufzeiten für maschinelles Lernen verwenden, oder Sie können eine vollständig angepasste Inferenzlösung für maschinelles Lernen mit Ihren eigenen Modellen und Ihrer eigenen Laufzeit entwickeln. Wenn Ihre Modelle eine Runtime verwenden, für die AWS IoT Greengrass eine Beispiel-Laufzeitkomponente bereitgestellt wird, können Sie diese Laufzeitkomponente verwenden, und Sie müssen benutzerdefinierte Komponenten nur für Ihren Inferenzcode und die Modelle erstellen, die Sie verwenden möchten.

Themen

- Rufen Sie das Rezept für eine öffentliche Komponente ab
- Rufen Sie Beispielartefakte von Komponenten ab
- Laden Sie Komponentenartefakte in einen S3-Bucket hoch
- Erstellen Sie benutzerdefinierte Komponenten

Rufen Sie das Rezept für eine öffentliche Komponente ab

Sie können das Rezept einer vorhandenen öffentlichen Komponente für maschinelles Lernen 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 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 die Option Rezept anzeigen aus.

- Verwenden 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>*.

Rufen Sie Beispielartefakte von Komponenten ab

Sie können die von den öffentlichen Machine-Learning-Komponenten verwendeten Artefakte als Vorlagen verwenden, um Ihre benutzerdefinierten Komponentenartefakte wie Inferenzcode oder Runtime-Installationsskripten zu erstellen.

Um die Beispielartefakte anzuzeigen, die in den öffentlichen Komponenten für maschinelles Lernen enthalten sind, stellen Sie die öffentliche Inferenzkomponente bereit und sehen Sie sich dann die Artefakte auf Ihrem Gerät im */greengrass/v2*`/packages/artifacts-unarchived/`*component-name*`/`*component-version*`/` Ordner an.

Laden Sie Komponentenartefakte in einen S3-Bucket hoch

Bevor Sie eine benutzerdefinierte Komponente erstellen können, müssen Sie die Komponentenartefakte in einen S3-Bucket hochladen und den S3 URIs in Ihrem Komponentenrezept verwenden. Um beispielsweise benutzerdefinierten Inferenzcode in Ihrer Inferenzkomponente zu verwenden, laden Sie den Code in einen S3-Bucket hoch. Sie können dann den Amazon S3 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 S3-Buckets im Amazon Simple Storage Service-Benutzerhandbuch.

> **Note**
>
> Sie müssen Ihre Artefakte in S3-Buckets speichern, die sich im selben AWS-Konto und AWS-Region wie die Komponenten befinden. Um auf diese Artefakte zugreifen AWS IoT Greengrass zu können, muss die Greengrass-Geräterolle die `s3:GetObject` Aktion

zulassen. Weitere Informationen zur Geräterolle finden Sie unterAutorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

Erstellen Sie benutzerdefinierte Komponenten

Sie können die Artefakte und Rezepte, die Sie abgerufen haben, verwenden, um Ihre benutzerdefinierten Komponenten für maschinelles Lernen zu erstellen. Ein Beispiel finden Sie unter Erstellen Sie eine benutzerdefinierte Inferenzkomponente.

Ausführliche Informationen zum Erstellen und Bereitstellen von Komponenten auf Greengrass-Geräten finden Sie unter AWS IoT Greengrass Komponenten entwickeln undAWS IoT Greengrass Komponenten auf Geräten bereitstellen.

Erstellen Sie eine benutzerdefinierte Inferenzkomponente

In diesem Abschnitt erfahren Sie, wie Sie mithilfe der DLR-Bildklassifizierungskomponente als Vorlage eine benutzerdefinierte Inferenzkomponente erstellen.

Themen

- Laden Sie Ihren Inferenzcode in einen Amazon S3 S3-Bucket hoch
- Erstellen Sie ein Rezept für Ihre Inferenzkomponente
- Erstellen Sie die Inferenzkomponente

Laden Sie Ihren Inferenzcode in einen Amazon S3 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 S3-Buckets im Amazon Simple Storage Service-Benutzerhandbuch.

> **Note**
>
> Sie müssen Ihre Artefakte in S3-Buckets speichern, die sich im selben AWS-Konto und AWS-Region wie die Komponenten befinden. Um auf diese Artefakte zugreifen AWS IoT Greengrass zu können, muss die Greengrass-Geräterolle die `s3:GetObject` Aktion zulassen. Weitere Informationen zur Geräterolle finden Sie unterAutorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

Erstellen Sie ein Rezept 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>
   ```

*<recipe-file>*Ersetzen Sie es durch den Namen des Rezepts im Format*<component-name>*-*<component-version>*.

2. Führen Sie in dem `ComponentDependencies` Objekt in Ihrem Rezept je nach Modell und Laufzeitkomponenten, die Sie verwenden möchten, eine oder mehrere der folgenden Aktionen aus:

   - Behalten Sie die Abhängigkeit der DLR-Komponenten 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.

     Runtime-Komponente

     JSON

     ```
     {
         "<runtime-component>": {
             "VersionRequirement": "<version>",
             "DependencyType": "HARD"
         }
     }
     ```

     YAML

     ```
     <runtime-component>:
         VersionRequirement: "<version>"
         DependencyType: HARD
     ```

   - Behalten Sie die Speicherabhängigkeit des DLR-Bildklassifizierungsmodells bei, um die bereitgestellten vortrainierten ResNet -50-Modelle zu verwenden, oder ändern Sie es, um eine benutzerdefinierte Modellkomponente zu verwenden. AWS Wenn Sie eine Abhängigkeit für eine Komponente eines öffentlichen Modells hinzufügen AWS-Konto und eine spätere benutzerdefinierte Version der Komponente in derselben Hand vorhanden ist AWS-Region, verwendet die Inferenzkomponente diese benutzerdefinierte Komponente. Geben Sie die Abhängigkeit von 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
```

Komponente für benutzerdefiniertes Modell

JSON

```
{
    "<custom-model-component>": {
        "VersionRequirement": "<version>",
        "DependencyType": "HARD"
    }
}
```

YAML

```
<custom-model-component>:
    VersionRequirement: "<version>"
    DependencyType: HARD
```

3. Fügen Sie im `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 sie so, dass `ModelResourceKey` sie die Namen der Modelle angeben, 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 zu den Artefakten und der Konfiguration dieser Komponente an, die verwendet werden, wenn die Komponente auf verschiedenen Plattformen bereitgestellt wird, sowie alle anderen Informationen, die für die erfolgreiche Ausführung der Komponente erforderlich sind. Der folgende Auszug zeigt die Konfiguration des `Manifests` Objekts für die Linux-Plattform in der DLR-Komponente zur Bildklassifizierung.

   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 Sie die Inferenzkomponente

Verwenden Sie die AWS IoT Greengrass Konsole oder die AWS CLI , um eine Komponente mit dem Rezept zu erstellen, das Sie gerade definiert haben. 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: Führen Sie mit Lite eine Inferenz zur Klassifizierung von Beispielbildern durch TensorFlow

Erstellen Sie eine benutzerdefinierte Inferenzkomponente (Konsole)

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 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 eingeben als Komponentenquelle aus.
5. Geben Sie im Feld Rezept das benutzerdefinierte Rezept ein, das Sie erstellt haben.
6. Klicken Sie auf Komponente erstellen.

Erstellen Sie eine benutzerdefinierte 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 Dienst in der erstellt AWS 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 AWS IoT Greengrass Komponenten entwickeln.

Fehlerbehebung bei Inferenz für maschinelles Lernen

Verwenden Sie die Informationen und Lösungen zur Fehlerbehebung in diesem Abschnitt, um Probleme mit Ihren Komponenten für maschinelles Lernen zu lösen. Informationen zu den öffentlichen Inferenzkomponenten für maschinelles Lernen 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 für die Inferenz verwendet wird.

Problembereiche

- Bibliothek konnte nicht abgerufen werden
- Cannot open shared object file
- Error: ModuleNotFoundError: No module named '<library>'
- Es wurde 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
- Speicherplatzfehler
- Timeout-Fehler

Bibliothek konnte nicht abgerufen werden

Der folgende Fehler tritt auf, wenn das Installationsskript während der Bereitstellung auf einem Raspberry Pi-Gerät eine erforderliche Bibliothek nicht 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 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 eine erforderliche Abhängigkeit für nicht herunterlädt.

```
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>'

Möglicherweise wird dieser Fehler in den Protokollen (`variant.DLR.log`odervariant.TensorFlowLite.log) der ML-Laufzeitkomponente angezeigt, wenn die ML-Laufzeitbibliothek oder ihre Abhängigkeiten nicht korrekt installiert sind. Dieser Fehler kann in den folgenden Fällen auftreten:

- Wenn Sie die `UseInstaller` Option verwenden, 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-Laufzeitoption UseInstaller

  - TensorFlowLite-Laufzeitoption UseInstaller

- Wenn Sie die `UseInstaller` Option nicht verwenden, weist dieser Fehler darauf hin, dass die ML-Laufzeit oder ihre Abhängigkeiten für den Systembenutzer, der die ML-Komponenten ausführt, nicht installiert sind. Gehen Sie wie folgt vor:

  1. Stellen Sie sicher, dass die Bibliothek für den Systembenutzer installiert ist, der die ML-Komponenten ausführt. *`ggc_user`*Ersetzen Sie es durch den Namen des Systembenutzers und *`tflite_runtime`* durch den Namen der Bibliothek, die überprüft werden soll.

     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. *`ggc_user`*Ersetzen Sie es durch den Namen des Systembenutzers und *`tflite_runtime`* ersetzen Sie es 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 der einzelnen ML-Runtime finden Sie im Folgenden:

     - DLR-Laufzeitoption UseInstaller
     - TensorFlowLite-Laufzeitoption UseInstaller

  3. Wenn das Problem weiterhin besteht, installieren Sie die Bibliothek für einen anderen Benutzer, um zu überprüfen, ob dieses Gerät die Bibliothek installieren kann. Bei dem Benutzer kann es sich beispielsweise um Ihren Benutzer, den Root-Benutzer oder einen Administratorbenutzer handeln. Wenn Sie die Bibliothek für keinen Benutzer erfolgreich installieren können, unterstützt Ihr Gerät die Bibliothek möglicherweise nicht. Lesen Sie in der Dokumentation der Bibliothek nach, um die Anforderungen zu überprüfen und Installationsprobleme zu beheben.

Es wurde 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 deuten darauf hin, dass die Runtime-Komponente 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 *MLRootPath* ist auf eingestellt*/greengrass/v2*`/work/`*component-name*`/greengrass_ml`. Um diesen Speicherort zu ändern, nehmen Sie die DLR-Laufzeit oder TensorFlow Lite-Laufzeit Runtime-Komponente direkt in Ihre Bereitstellung auf und geben Sie in einem Update zur Zusammenführung der Konfiguration einen geänderten Wert für den `MLRootPath` Parameter an. Weitere Informationen zur Konfiguration der Komponente finden Sie unterKomponentenkonfigurationen aktualisieren.

> **Note**
>
> Für die DLR-Komponente v1.3.x legen Sie den `MLRootPath` Parameter in der Konfiguration der Inferenzkomponente fest. 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 Inferenz für maschinelles Lernen 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 enthält NumPy als die Version, die OpenCV benötigt. Um dieses Problem zu beheben, führen Sie den folgenden Befehl aus, um auf die neueste Version NumPy zu aktualisieren.

```
pip3 install --upgrade numpy
```

picamera.exc.PiCameraError: Camera is not enabled

Möglicherweise wird der folgende Fehler angezeigt, wenn Sie Machine Learning Inference auf einem Raspberry Pi mit Raspberry Pi OS Bullseye ausführen.

```
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 Kamerastack enthält, der nicht mit den ML-Komponenten kompatibel ist. Um dieses Problem zu beheben, aktivieren Sie den älteren Kamerastack.

Um den älteren 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.

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 maschinelles Lernen.

Speicherplatzfehler

Der `no space left on device` Fehler tritt normalerweise 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 Festplattenspeicher, um eine öffentliche Inferenzkomponente für maschinelles Lernen bereitzustellen.

Timeout-Fehler

Die öffentlichen Komponenten für maschinelles Lernen laden große Modelldateien für maschinelles Lernen herunter, die größer als 200 MB sind. Wenn beim Herunterladen während der Bereitstellung ein Timeout auftritt, überprüfen Sie die Geschwindigkeit Ihrer Internetverbindung und versuchen Sie es erneut.

Verwalten Sie Greengrass-Kerngeräte mit AWS Systems Manager

> **Note**
>
> AWS IoT Greengrass unterstützt diese Funktion derzeit nicht auf Windows-Kerngeräten.

Systems Manager ist ein AWS Service, mit dem Sie Ihre Infrastruktur anzeigen und steuern können AWS, einschließlich EC2 Amazon-Instances, lokalen Servern und virtuellen Maschinen (VMs) sowie Edge-Geräten. Mit Systems Manager können Sie Betriebsdaten anzeigen, Betriebsaufgaben automatisieren und Sicherheit und Compliance gewährleisten. 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 eine 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, registriert er das Kerngerät als von Systems Manager verwalteten Knoten. Der Systems Manager Agent wird auf dem Gerät ausgeführt, um die Kommunikation mit dem Systems Manager Manager-Dienst in der zu ermöglichen AWS Cloud. Weitere Informationen zur Installation und Konfiguration der Systems Manager Agent-Komponente finden Sie unterInstallieren Sie den AWS Systems Manager Agenten.

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.

AWS Systems Manager bietet eine Stufe „Standard-Instances“ und eine Stufe „Advanced-Instances“ für von Systems Manager verwaltete Knoten. Wenn Sie Systems Manager zum ersten Mal verwenden, beginnen Sie mit der Stufe „Standardinstanzen“. Auf der Stufe „Standardinstanzen“ können Sie bis zu 1.000 verwaltete Knoten pro Stufe registrieren. AWS-Region AWS-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 nutzen müssen, verwenden Sie die Stufe Advanced-

Instances. Weitere Informationen finden Sie im AWS Systems Manager Benutzerhandbuch unter Konfiguration von Instanzstufen.

Themen

- Installieren Sie den AWS Systems Manager Agenten
- Den AWS Systems Manager Agenten deinstallieren

Installieren Sie den AWS Systems Manager Agenten

Der AWS Systems Manager Agent (Systems Manager Agent) ist eine Amazon-Software, die Sie installieren, damit Systems Manager Greengrass-Kerngeräte, EC2 Amazon-Instances und andere Ressourcen aktualisieren, verwalten und konfigurieren kann. Der Agent verarbeitet und führt Anfragen vom Systems Manager Manager-Dienst in der aus AWS Cloud. Anschließend sendet der Agent Status- und Laufzeitinformationen zurück an den Systems Manager Manager-Dienst. Weitere Informationen finden Sie im AWS Systems Manager Benutzerhandbuch unter About Systems Manager Agent.

AWS stellt den Systems Manager Agent als Greengrass-Komponente bereit, die Sie auf Ihren Greengrass-Kerngeräten einsetzen können, um sie mit Systems Manager zu verwalten. Die Systems Manager Agent-Komponente installiert die Systems Manager Agent-Software und registriert das Kerngerät als verwalteten Knoten in Systems Manager. Folgen Sie den Schritten auf dieser Seite, um die Voraussetzungen zu erfüllen und die Systems Manager Agent-Komponente auf einem Kerngerät oder einer Gruppe von Kerngeräten bereitzustellen.

Themen

- Schritt 1: Ausführen allgemeiner Systems Manager-Einrichtungsschritte
- Schritt 2: Erstellen Sie eine IAM-Servicerolle für Systems Manager
- Schritt 3: Fügen Sie der Token-Exchange-Rolle Berechtigungen hinzu
- Schritt 4: Bereitstellen der Systems Manager Agent-Komponente
- Schritt 5: Überprüfen Sie die Registrierung des Kerngeräts mit Systems Manager

Schritt 1: Ausführen allgemeiner Systems Manager-Einrichtungsschritte

Falls Sie dies noch nicht getan haben, führen Sie die allgemeinen Einrichtungsschritte für aus AWS Systems Manager. Weitere Informationen finden Sie im AWS Systems Manager Benutzerhandbuch unter Vollständige allgemeine Schritte zur Einrichtung von Systems Manager.

Schritt 2: Erstellen Sie eine IAM-Servicerolle für Systems Manager

Der Systems Manager Agent verwendet eine AWS Identity and Access Management (IAM) - Servicerolle für die Kommunikation mit AWS Systems Manager. Systems Manager übernimmt diese Rolle, um die Systems Manager Manager-Funktionen auf jedem Kerngerät zu aktivieren. Die Systems Manager Agent-Komponente verwendet diese Rolle auch, um das Kerngerät bei der Bereitstellung der Komponente als von Systems Manager verwalteten Knoten zu registrieren. Falls Sie dies noch nicht getan haben, erstellen Sie eine Systems Manager-Dienstrolle, die von der Systems Manager Agent-Komponente verwendet werden soll. Weitere Informationen finden Sie im AWS Systems Manager Benutzerhandbuch unter Erstellen einer IAM-Servicerolle für Edge-Geräte.

Schritt 3: Fügen Sie der Token-Exchange-Rolle Berechtigungen hinzu

Greengrass-Core-Geräte verwenden eine IAM-Servicerolle, die so genannte Token-Exchange-Rolle, um mit AWS Diensten zu interagieren. Jedes Kerngerät verfügt über eine Token-Austauschrolle, die Sie bei der Installation der AWS IoT Greengrass Core-Software erstellen. Viele Greengrass-Komponenten, wie der Systems Manager Agent, erfordern zusätzliche Berechtigungen für diese Rolle. Für die Systems Manager Manager-Agentenkomponente sind die folgenden Berechtigungen erforderlich, darunter die Berechtigung zur Verwendung der Rolle, in der Sie sie erstellt habenSchritt 2: Erstellen Sie eine 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": "*"
    }
```

```
  ]
}
```

Falls Sie dies noch nicht getan haben, fügen Sie diese Berechtigungen der Token-Austauschrolle des Kerngeräts hinzu, damit der Systems Manager Agent arbeiten kann. Sie können der Token-Austauschrolle eine neue Richtlinie hinzufügen, um diese Berechtigung zu erteilen.

Um der Token-Exchange-Rolle (Konsole) Berechtigungen hinzuzufügen

1. Wählen Sie im Navigationsmenü der IAM-Konsole die Option Rollen aus.
2. Wählen Sie die IAM-Rolle aus, die Sie bei der Installation der AWS IoT Greengrass Core-Software als Token-Exchange-Rolle 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 `GreengrassV2TokenExchangeRole` erstellt.
3. Wählen Sie unter Berechtigungen die Option Berechtigungen hinzufügen und anschließend Richtlinien anhängen aus.
4. Wählen Sie Richtlinie erstellen aus. Die Seite „Richtlinie erstellen“ wird in einem neuen Browser-Tab 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. *SSMServiceRole*Ersetzen Sie es durch den Namen der Servicerolle, in der Sie erstellt habenSchritt 2: Erstellen Sie eine 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. Klicken Sie auf der Seite „Berechtigungen hinzufügen“ auf die Schaltfläche „Aktualisieren“ und wählen Sie dann die Greengrass Systems Manager Agentenrichtlinie aus, die Sie im vorherigen Schritt erstellt haben.
7. Wählen Sie Richtlinien anfügen.

   Die Kerngeräte, die diese Token-Austauschrolle verwenden, sind jetzt berechtigt, mit dem Systems Manager Manager-Dienst zu interagieren.

Um der Token-Austauschrolle Berechtigungen hinzuzufügen (AWS CLI)

So fügen Sie eine Richtlinie hinzu, die die Erlaubnis zur Verwendung von Systems Manager erteilt

1. Erstellen Sie eine Datei mit dem Namen `ssm-agent-component-policy.json` und kopieren Sie den folgenden JSON-Code in die Datei. *`SSMServiceRole`*Ersetzen Sie es durch den Namen der Servicerolle, in der Sie erstellt habenSchritt 2: Erstellen Sie eine 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 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 Token-Exchange-Rolle anzuhängen.

   - *GreengrassV2TokenExchangeRole*Ersetzen Sie es durch den Namen der Token-Austauschrolle, 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 Kerngeräte, die diese Token-Austauschrolle verwenden, sind jetzt berechtigt, mit dem Systems Manager Manager-Dienst zu interagieren.

Schritt 4: Bereitstellen der Systems Manager Agent-Komponente

Gehen Sie wie folgt vor, 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 (Konsole) bereit

1. Wählen Sie im Navigationsmenü der AWS IoT Greengrass Konsole die Option Komponenten aus.
2. Wählen Sie auf der Seite Komponenten die Registerkarte Öffentliche Komponenten und wählen Sie dann aws.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 erstellen Sie eine neue Bereitstellung und klicken Sie dann auf Weiter.
5. Wenn Sie eine neue Bereitstellung erstellen möchten, wählen Sie das Ziel-Core-Gerät oder die Dinggruppe für die Bereitstellung aus. Wählen Sie auf der Seite „Ziel angeben“ unter Bereitstellungsziel ein Kerngerät oder eine Dinggruppe aus, und klicken Sie dann auf Weiter.
6. Vergewissern Sie sich auf der Seite „Komponenten auswählen“, dass die aws.greengrass.SystemsManagerAgentKomponente ausgewählt ist, und klicken Sie auf Weiter.
7. Wählen Sie auf der Seite Komponenten konfigurieren die Option aus aws.greengrass.SystemsManagerAgent, und gehen Sie dann wie folgt vor:

   a. Wählen Sie Komponente konfigurieren aus.

   b. Geben Sie im aws.greengrass.SystemsManagerAgent Modal konfigurieren unter Konfigurationsupdate unter Konfiguration zum Zusammenführen das folgende Konfigurationsupdate ein. *SSMServiceRole*Ersetzen Sie es durch den Namen der Servicerolle, in der Sie erstellt habenSchritt 2: Erstellen Sie eine IAM-Servicerolle für Systems Manager.

```
{
  "SSMRegistrationRole": "SSMServiceRole",
  "SSMOverrideExistingRegistration": false
}
```

> **Note**
>
> Wenn auf dem Kerngerät bereits der Systems Manager Agent ausgeführt wird, der mit einer Hybrid-Aktivierung registriert ist, wechseln Sie `SSMOverrideExistingRegistration` zu`true`. Dieser Parameter gibt an, ob die Systems Manager Agent-Komponente das Kerngerät registriert, wenn der Systems Manager Agent bereits auf dem Gerät mit einer Hybridaktivierung 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 Kerngerät erstellt. Weitere Informationen finden Sie unter Konfiguration der Systems Manager Agent-Komponenten.

c. Wählen Sie Bestätigen, um das Modal zu schließen, und klicken Sie dann auf Weiter.

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.

   Es kann bis zu einer Minute dauern, bis die Bereitstellung abgeschlossen ist.

So stellen Sie die Systems Manager Agent-Komponente bereit (AWS CLI)

Um die Systems Manager Agent-Komponente bereitzustellen, erstellen Sie ein Bereitstellungsdokument, das das `components` Objekt enthält`aws.greengrass.SystemsManagerAgent`, und geben Sie das Konfigurationsupdate für die Komponente an. Folgen Sie den Anweisungen unterErstellen von Bereitstellungen, um eine neue Einrichtung zu erstellen oder eine bestehende Einrichtung zu überarbeiten.

Im folgenden Beispieldokument zur teilweisen Bereitstellung wird angegeben, dass eine Servicerolle mit dem Namen `SSMServiceRole` verwendet werden soll. *`SSMServiceRole`*Ersetzen Sie es durch den Namen der Servicerolle, die Sie in erstellt habenSchritt 2: Erstellen Sie eine IAM-Servicerolle für Systems Manager.

```
{
  ...,
  "components": {
    ...,
    "aws.greengrass.SystemsManagerAgent": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"SSMRegistrationRole\":\"SSMServiceRole\",
\"SSMOverrideExistingRegistration\":false}"
      }
    }
  }
}
```

> **Note**
>
> Wenn auf dem Kerngerät bereits der Systems Manager Agent ausgeführt wird, der mit einer Hybrid-Aktivierung registriert ist, wechseln Sie `SSMOverrideExistingRegistration` zu`true`. Dieser Parameter gibt an, ob die Systems Manager Agent-Komponente das Kerngerät registriert, wenn der Systems Manager Agent bereits auf dem Gerät mit einer Hybridaktivierung 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 Kerngerät erstellt. Weitere Informationen finden Sie unter Konfiguration der Systems Manager Agent-Komponenten.

Es kann einige Minuten dauern, bis die Bereitstellung abgeschlossen ist. Sie können den AWS IoT Greengrass Dienst verwenden, um den Status der Bereitstellung zu überprüfen, und Sie können anhand der AWS IoT Greengrass Core-Softwareprotokolle und der Systems Manager Agent-Komponentenprotokolle überprüfen, ob der Systems Manager Agent erfolgreich ausgeführt wird. Weitere Informationen finden Sie hier:

- Überprüfen Sie den Bereitstellungsstatus
- AWS IoT Greengrass Protokolle überwachen
- Systems Manager Agent-Protokolle im AWS Systems Manager Benutzerhandbuch anzeigen

Wenn die Bereitstellung fehlschlägt oder der Systems Manager Agent nicht ausgeführt wird, können Sie Fehler bei der Bereitstellung auf jedem Kerngerät beheben. Weitere Informationen finden Sie hier:

- Problembehebung AWS IoT Greengrass V2
- Problembehandlung bei Systems Manager Agent im AWS Systems Manager Benutzerhandbuch

Schritt 5: Überprüfen Sie die Registrierung des Kerngeräts mit Systems Manager

Wenn die Systems Manager Agent-Komponente ausgeführt wird, registriert sie das Kerngerät als verwalteten Knoten in Systems Manager. Sie können die AWS IoT Greengrass Konsole, die Systems Manager Manager-Konsole und die Systems Manager Manager-API verwenden, um zu überprüfen, ob ein Kerngerät als verwalteter Knoten registriert ist. Verwaltete Knoten werden in Teilen der AWS Konsole und der API auch als Instanzen bezeichnet.

Um die Registrierung des Kerngeräts zu überprüfen (AWS IoT Greengrass Konsole)

1. Wählen Sie im Navigationsmenü der AWS IoT Greengrass Konsole die Option Core-Geräte aus.
2. Wählen Sie das zu verifizierende Kerngerät aus.
3. Suchen Sie auf der Detailseite des Kerngeräts nach der AWS Systems Manager Instanzeigenschaft. Wenn diese Eigenschaft vorhanden ist und ein Link zur Systems Manager Manager-Konsole angezeigt wird, ist das Kerngerät als verwalteter Knoten registriert.

   Sie finden auch die Eigenschaft AWS Systems Manager Ping-Status, mit der Sie den Status des Systems Manager Agents auf dem Core-Gerät überprüfen können. Wenn der Status Online lautet, können Sie das Kerngerät mit Systems Manager verwalten.

Um die Registrierung des Kerngeräts zu überprüfen (Systems Manager Manager-Konsole)

1. Wählen Sie im Navigationsmenü der Systems Manager Manager-Konsole die Option Fleet Manager.
2. Gehen Sie unter Verwaltete Knoten wie folgt vor:

   a. Fügen Sie einen Filter hinzu, bei dem Quelltyp der Wert ist AWS::IoT::Thing.

   b. Fügen Sie einen Filter hinzu, bei dem Quell-ID der Name des zu verifizierenden Kerngeräts ist.

3. Suchen Sie das Kerngerät in der Tabelle mit verwalteten Knoten. Wenn das Kerngerät in der Tabelle aufgeführt ist, ist es als verwalteter Knoten registriert.

   Sie können auch die Eigenschaft Ping-Status des Systems Manager-Agenten aufrufen, um den Status des Systems Manager Agents auf dem Kerngerät zu überprüfen. Wenn der Status Online lautet, können Sie das Kerngerät mit Systems Manager verwalten.

Um die Registrierung des Kerngeräts zu überprüfen (AWS CLI)

- Verwenden Sie den DescribeInstanceInformationVorgang, um die Liste der verwalteten Knoten abzurufen, die einem von Ihnen angegebenen Filter entsprechen. Führen Sie den folgenden Befehl aus, um zu überprüfen, ob ein Core-Gerät als verwalteter Knoten registriert ist. *MyGreengrassCore*Ersetzen Sie es durch den Namen des Kerngeräts, um dies zu überprüfen.

  ```
  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 Kerngerät als verwalteter Knoten registriert. In der Antwort finden Sie auch weitere Informationen zum verwalteten Knoten des Core-Geräts. Wenn die `PingStatus` Eigenschaft lautet`Online`, können Sie das Kerngerät mit Systems Manager verwalten.

Nachdem Sie sich vergewissert haben, dass ein Core-Gerät als verwalteter Knoten in Systems Manager registriert ist, können Sie dieses Core-Gerät mit der Systems Manager Manager-Konsole und der API verwalten. Weitere Informationen zu den Systems Manager Manager-Funktionen, mit denen Sie Greengrass-Kerngeräte verwalten können, finden Sie unter Systems Manager Manager-Funktionen im AWS Systems Manager Benutzerhandbuch.

Den AWS Systems Manager Agenten deinstallieren

Wenn Sie ein Greengrass Core-Gerät nicht mehr mit verwalten möchten AWS Systems Manager, können Sie das Core-Gerät von Systems Manager abmelden und den AWS Systems Manager Agenten (Systems Manager Agent) vom Gerät deinstallieren.

Sie können ein Core-Gerät jederzeit erneut 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 Core-Gerät 30 Tage lang.

Themen

- Schritt 1: Das Kerngerät vom Systems Manager abmelden
- Schritt 2: Deinstallieren Sie die Systems Manager Agent-Komponente
- Schritt 3: Deinstallieren Sie die Systems Manager Agent-Software

Schritt 1: Das Kerngerät vom Systems Manager abmelden

Sie können die Systems Manager Manager-Konsole oder API verwenden, um das Core-Gerät abzumelden. Weitere Informationen finden Sie im Benutzerhandbuch unter Deregistrierung verwalteter Knoten.AWS Systems Manager

Schritt 2: Deinstallieren Sie die Systems Manager Agent-Komponente

Nachdem Sie das Core-Gerät deregistriert 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. Die AWS IoT Greengrass Core-Software deinstalliert eine Komponente, wenn keine der Bereitstellungen eines Kerngeräts diese Komponente spezifiziert. Weitere Informationen finden Sie unter AWS IoT Greengrass Komponenten auf Geräten bereitstellen.

So deinstallieren Sie die Systems Manager Agent-Komponente (Konsole)

1. Wählen Sie im Navigationsmenü der AWS IoT Greengrass Konsole 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 zum Kerngerät die Registerkarte Bereitstellungen aus.
4. Wählen Sie die Bereitstellung, mit der die Systems Manager Agent-Komponente auf dem Kerngerät bereitgestellt wird.
5. Wählen Sie auf der Seite mit den Bereitstellungsdetails die Option Revise aus.
6. Wählen Sie im Modal Bereitstellung überarbeiten die Option Bereitstellung überarbeiten aus.
7. Wählen Sie in Schritt 1: Ziel angeben die Option Weiter aus.
8. Löschen Sie in Schritt 2: Komponenten auswählen die Auswahl für aws.greengrass.SystemsManagerAgentKomponente, und wählen Sie dann Weiter.

9. Wählen Sie in Schritt 3: Komponenten konfigurieren die Option Weiter aus.
10. Wählen Sie in Schritt 4: Erweiterte Einstellungen konfigurieren die Option Weiter.
11. Wählen Sie in Schritt 5: Überprüfen die Option Bereitstellen 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 Bereitstellungen überarbeiten.

Es kann einige Minuten dauern, bis die Bereitstellung abgeschlossen ist. Sie können den AWS IoT Greengrass Dienst verwenden, um den Status der Bereitstellung zu überprüfen. Weitere Informationen finden Sie unter Überprüfen Sie den Bereitstellungsstatus.

Schritt 3: Deinstallieren Sie die Systems Manager Agent-Software

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 Manager Benutzerhandbuch unter Systems Manager Agent von Linux-Instances deinstallieren.

Sicherheit in AWS IoT Greengrass

Cloud-Sicherheit AWS hat höchste Priorität. Als AWS Kunde profitieren Sie von einer Rechenzentrums- und Netzwerkarchitektur, die darauf ausgelegt sind, die Anforderungen der sicherheitssensibelsten Unternehmen zu erfüllen.

Sicherheit ist eine gemeinsame Verantwortung von Ihnen AWS und Ihnen. Das Modell der geteilten Verantwortung beschreibt dies als Sicherheit der Cloud selbst und Sicherheit in der Cloud:

- Sicherheit der Cloud — AWS ist verantwortlich für den Schutz der Infrastruktur, auf der AWS Dienste in der ausgeführt AWS Cloud werden. AWS bietet Ihnen auch Dienste, die Sie sicher nutzen können. Externe Prüfer testen und verifizieren regelmäßig die Wirksamkeit unserer Sicherheitsmaßnahmen im Rahmen der AWS . Weitere Informationen zu den Compliance-Programmen, die für gelten AWS IoT Greengrass, finden Sie unter AWS Services im Umfang nach Compliance-Programmen AWS .
- Sicherheit in der Cloud: Ihr Verantwortungsumfang wird durch den AWS -Service bestimmt, den Sie verwenden. Sie sind auch für andere Faktoren verantwortlich, etwa für die Vertraulichkeit Ihrer Daten, die Anforderungen Ihres Unternehmens und die geltenden Gesetze und Vorschriften.

Bei der Nutzung AWS IoT Greengrass sind Sie auch für die Sicherung Ihrer Geräte, Ihrer lokalen Netzwerkverbindung und Ihrer privaten Schlüssel verantwortlich.

Diese Dokumentation hilft Ihnen zu verstehen, wie Sie das Modell der gemeinsamen Verantwortung bei der Verwendung anwenden können AWS IoT Greengrass. In den folgenden Themen erfahren Sie, wie Sie die Konfiguration vornehmen AWS IoT Greengrass , um Ihre Sicherheits- und Compliance-Ziele zu erreichen. Sie erfahren auch, wie Sie andere AWS Dienste nutzen können, die Sie bei der Überwachung und Sicherung Ihrer AWS IoT Greengrass Ressourcen unterstützen.

Themen

- Datenschutz in AWS IoT Greengrass
- Geräteauthentifizierung und Autorisierung für AWS IoT Greengrass
- Identitäts- und Zugriffsmanagement für AWS IoT Greengrass
- Geräteverkehr über einen Proxy oder eine Firewall zulassen
- Konformitätsvalidierung für AWS IoT Greengrass
- FIPS-Endpunkte

- Resilienz in AWS IoT Greengrass
- Infrastruktursicherheit in AWS IoT Greengrass
- Konfiguration und Schwachstellenanalyse in AWS IoT Greengrass
- Codeintegrität in AWS IoT Greengrass V2
- AWS IoT Greengrass und Schnittstellen-VPC-Endpunkte ()AWS PrivateLink
- Bewährte Sicherheitsmethoden für AWS IoT Greengrass

Datenschutz in AWS IoT Greengrass

Das Modell der AWS gemeinsamen Verantwortung und geteilter Verantwortung gilt für den Datenschutz in AWS IoT Greengrass. Wie in diesem Modell beschrieben, AWS ist verantwortlich für den Schutz der globalen Infrastruktur, auf der alle Systeme laufen AWS Cloud. Sie sind dafür verantwortlich, die Kontrolle über Ihre in dieser Infrastruktur gehosteten Inhalte zu behalten. Sie sind auch für die Sicherheitskonfiguration und die Verwaltungsaufgaben für die von Ihnen verwendeten AWS-Services verantwortlich. Weitere Informationen zum Datenschutz finden Sie unter Häufig gestellte Fragen zum Datenschutz. Informationen zum Datenschutz in Europa finden Sie im Blog-Beitrag AWS -Modell der geteilten Verantwortung und in der DSGVO im AWS -Sicherheitsblog.

Aus Datenschutzgründen empfehlen wir, dass Sie AWS-Konto Anmeldeinformationen schützen und einzelne Benutzer mit AWS IAM Identity Center oder AWS Identity and Access Management (IAM) einrichten. So erhält jeder Benutzer nur die Berechtigungen, die zum Durchführen seiner Aufgaben erforderlich sind. Außerdem empfehlen wir, die Daten mit folgenden Methoden schützen:

- Verwenden Sie für jedes Konto die Multi-Faktor-Authentifizierung (MFA).
- Verwenden Sie SSL/TLS, um mit Ressourcen zu kommunizieren. AWS Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Richten Sie die API und die Protokollierung von Benutzeraktivitäten mit ein. AWS CloudTrail Informationen zur Verwendung von CloudTrail Pfaden zur Erfassung von AWS Aktivitäten finden Sie unter Arbeiten mit CloudTrail Pfaden im AWS CloudTrail Benutzerhandbuch.
- Verwenden Sie AWS Verschlüsselungslösungen zusammen mit allen darin enthaltenen Standardsicherheitskontrollen AWS-Services.
- Verwenden Sie erweiterte verwaltete Sicherheitsservices wie Amazon Macie, die dabei helfen, in Amazon S3 gespeicherte persönliche Daten zu erkennen und zu schützen.
- Wenn Sie für den Zugriff AWS über eine Befehlszeilenschnittstelle oder eine API FIPS 140-3-validierte kryptografische Module benötigen, verwenden Sie einen FIPS-Endpunkt. Weitere

Informationen über verfügbare FIPS-Endpunkte finden Sie unter Federal Information Processing Standard (FIPS) 140-3.

Wir empfehlen dringend, in Freitextfeldern, z. B. im Feld Name, keine vertraulichen oder sensiblen Informationen wie die E-Mail-Adressen Ihrer Kunden einzugeben. Dies gilt auch, wenn Sie mit der Konsole, der AWS IoT Greengrass API oder auf andere AWS-Services Weise arbeiten oder diese verwenden. AWS CLI AWS SDKs Alle Daten, die Sie in Tags oder Freitextfelder eingeben, die für Namen verwendet werden, können für Abrechnungs- oder Diagnoseprotokolle verwendet werden. Wenn Sie eine URL für einen externen Server bereitstellen, empfehlen wir dringend, keine Anmeldeinformationen zur Validierung Ihrer Anforderung an den betreffenden Server in die URL einzuschließen.

Weitere Hinweise zum Schutz vertraulicher Informationen finden Sie unterthe section called “Keine Protokollierung sensibler Informationen”. AWS IoT Greengrass

Weitere Informationen zum Datenschutz enthält der Blog-Beitrag AWS Shared Responsibility Model and GDPR im AWS -Sicherheitsblog.

Themen

- Datenverschlüsselung
- Integration von Hardware-Sicherheit

Datenverschlüsselung

AWS IoT Greengrass verwendet Verschlüsselung, um Daten während der Übertragung (über das Internet oder ein lokales Netzwerk) und im Ruhezustand (gespeichert im AWS Cloud) zu schützen.

Geräte in einer AWS IoT Greengrass Umgebung sammeln häufig Daten, die zur weiteren Verarbeitung an AWS Dienste gesendet werden. Weitere Informationen zur Datenverschlüsselung bei anderen AWS Diensten finden Sie in der Sicherheitsdokumentation für diesen Dienst.

Themen

- Verschlüsselung während der Übertragung
- Verschlüsselung im Ruhezustand
- Schlüsselverwaltung für das Greengrass Core-Gerät

Verschlüsselung während der Übertragung

AWS IoT Greengrass hat zwei Kommunikationsarten, wenn Daten übertragen werden:

- the section called “Daten in Übertragung über das Internet”. Die Kommunikation zwischen einem Greengrass-Kern und AWS IoT Greengrass über das Internet ist verschlüsselt.
- the section called “Daten auf dem Core-Gerät”. Die Kommunikation zwischen Komponenten auf dem Greengrass-Core-Gerät ist nicht verschlüsselt.

Daten in Übertragung über das Internet

AWS IoT Greengrass verwendet Transport Layer Security (TLS), um die gesamte Kommunikation über das Internet zu verschlüsseln. Alle an die AWS Cloud gesendeten Daten werden über eine TLS-Verbindung unter Verwendung der Protokolle MQTT oder HTTPS gesendet, sodass sie standardmäßig sicher sind. AWS IoT Greengrass verwendet das AWS IoT Transportsicherheitsmodell. Weitere Informationen finden Sie unter Transportsicherheit im AWS IoT Core -Entwicklerhandbuch.

Daten auf dem Core-Gerät

AWS IoT Greengrass verschlüsselt keine lokal auf dem Greengrass-Core-Gerät ausgetauschten Daten, da die Daten das Gerät nicht verlassen. Dazu gehört die Kommunikation zwischen benutzerdefinierten Komponenten, dem AWS IoT Geräte-SDK und öffentlichen Komponenten wie dem Stream-Manager.

Verschlüsselung im Ruhezustand

AWS IoT Greengrass speichert Ihre Daten:

- the section called “Daten im Ruhezustand im AWS Cloud”. Diese Daten sind verschlüsselt.
- the section called “Daten im Ruhezustand auf dem Greengrass-Kern”. Diese Daten sind nicht verschlüsselt (außer lokalen Kopien Ihrer Geheimnisse).

Daten im Ruhezustand im AWS Cloud

AWS IoT Greengrass verschlüsselt Kundendaten, die in der AWS Cloud gespeichert sind. Diese Daten werden durch AWS KMS Schlüssel geschützt, die von AWS IoT Greengrass verwaltet werden.

Daten im Ruhezustand auf dem Greengrass-Kern

AWS IoT Greengrass stützt sich auf Unix-Dateiberechtigungen und vollständige Festplattenverschlüsselung (falls aktiviert), um Daten zu schützen, die sich auf dem Kern befinden. Sie sind für den Schutz des Dateisystems und des Geräts verantwortlich.

Verschlüsselt AWS IoT Greengrass jedoch lokale Kopien Ihrer abgerufenen Geheimnisse. AWS Secrets Manager Weitere Informationen finden Sie in der Secret Manager-Komponente.

Schlüsselverwaltung für das Greengrass Core-Gerät

Es liegt in der Verantwortung des Kunden, die sichere Speicherung kryptografischer (öffentlicher und privater) Schlüssel auf dem Greengrass-Core-Gerät zu gewährleisten. AWS IoT Greengrass verwendet 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-Core-Gerät unterstützt die Speicherung privater Schlüssel mithilfe von Dateisystemberechtigungen oder einem Hardware-Sicherheitsmodul. 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 Version 2.5.3 und höher der Greengrass Nucleus-Komponente verfügbar. AWS IoT Greengrass unterstützt diese Funktion derzeit nicht auf Windows Core-Geräten.

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 offengelegt oder in der Software dupliziert werden. Sie können den privaten Schlüssel und das Zertifikat auf einem Hardwaremodul wie einem HSM oder einem Trusted Platform Module (TPM) speichern.

Die AWS IoT Greengrass Core-Software verwendet einen privaten Schlüssel und ein X.509-Zertifikat, um Verbindungen zu den AND-Diensten zu authentifizieren. AWS IoT AWS IoT Greengrass Die Secret Manager-Komponente verwendet diesen privaten Schlüssel, um die Secrets, die Sie auf einem Greengrass-Core-Gerät bereitstellen, sicher zu verschlüsseln und zu entschlüsseln. Wenn Sie ein Kerngerät für die Verwendung eines HSM konfigurieren, verwenden diese Komponenten den privaten Schlüssel und das Zertifikat, die Sie im HSM speichern.

Die Moquette MQTT-Broker-Komponente speichert auch einen privaten Schlüssel für ihr lokales MQTT-Serverzertifikat. Diese Komponente speichert den privaten Schlüssel im Dateisystem des Geräts im Arbeitsordner der Komponente. Unterstützt derzeit AWS IoT Greengrass nicht das Speichern dieses privaten Schlüssels oder Zertifikats in einem HSM.

> **Tip**
>
> Suchen Sie im AWS Partner-Gerätekatalog nach Geräten, die diese Funktion unterstützen.

Themen

- Voraussetzungen
- Bewährte Methoden zur Hardwaresicherheit
- Installieren Sie die AWS IoT Greengrass Core-Software mit Hardwaresicherheit
- Konfigurieren Sie die Hardwaresicherheit auf einem vorhandenen Core-Gerät
- Verwenden Sie Hardware ohne PKCS #11 -Unterstützung
- Weitere Informationen finden Sie auch unter

Voraussetzungen

Sie müssen die folgenden Anforderungen erfüllen, um ein HSM auf einem Greengrass-Core-Gerät verwenden zu können:

- Greengrass Nucleus v2.5.3 oder höher ist auf dem Kerngerät installiert. Sie können eine kompatible Version wählen, wenn Sie die AWS IoT Greengrass Core-Software auf einem Core-Gerät installieren.
- Die PKCS #11 -Anbieterkomponente ist auf dem Kerngerät installiert. 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 Signaturschema PKCS #1 v1.5 und RSA-Schlüssel mit einer RSA-2048-Schlüsselgröße (oder größer) oder ECC-Schlüssel unterstützt.

  > **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.

- Eine PKCS #11 -Anbieterbibliothek, die die AWS IoT Greengrass Core-Software zur Laufzeit laden kann (mit libdl), 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 Steckplatz speichern, und sie müssen dieselbe Objektbezeichnung und Objekt-ID verwenden, wenn das HSM Objekt unterstützt. IDs
- Das Zertifikat und der private Schlüssel müssen durch Objektbezeichnungen aufgelöst werden können.
- Der private Schlüssel muss über die folgenden Berechtigungen verfügen:
  - `sign`
  - `decrypt`
- (Optional) Um die Secret Manager-Komponente verwenden zu können, müssen Sie Version 2.1.0 oder höher verwenden, und der private Schlüssel muss über die folgenden Berechtigungen verfügen:
  - `unwrap`
  - `wrap`

Bewährte Methoden zur Hardwaresicherheit

Beachten Sie bei der Konfiguration der Hardwaresicherheit auf Greengrass-Core-Geräten die folgenden bewährten Methoden.

- 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 im HSM verbleibt.
- Konfigurieren Sie private Schlüssel so, dass sie unveränderlich sind, und verbieten Sie den Export.
- Verwenden Sie das vom HSM-Hardwareanbieter empfohlene Bereitstellungstool, um mithilfe des hardwaregeschützten privaten Schlüssels eine Certificate Signing Request (CSR) zu generieren, und verwenden Sie dann die Konsole oder API, um ein Client-Zertifikat zu generieren. AWS IoT

> **Note**
>
> Die bewährte Sicherheitsmethode, Schlüssel zu rotieren, gilt nicht, wenn Sie private Schlüssel auf einem HSM generieren.

Installieren Sie die AWS IoT Greengrass Core-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 den bewährten Sicherheitsverfahren zur Generierung des privaten Schlüssels im HSM, sodass der private Schlüssel im HSM verbleibt.

Gehen Sie wie folgt vor, um die AWS IoT Greengrass Core-Software mit Hardwaresicherheit zu installieren:

1. Generieren Sie einen privaten Schlüssel im HSM.
2. Erstellen Sie aus dem privaten Schlüssel eine Zertifikatsignieranforderung (CSR).
3. Erstellen Sie ein Zertifikat aus der CSR. Sie können ein Zertifikat erstellen, das von AWS IoT oder von einer anderen Stammzertifizierungsstelle (CA) signiert wurde. Weitere Informationen zur Verwendung einer anderen Stammzertifizierungsstelle finden Sie unter Erstellen eigener Client-Zertifikate 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 die Verwendung der PKCS #11 -Anbieterkomponente sowie des privaten Schlüssels und des Zertifikats im HSM spezifiziert.

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.

Unterstützt derzeit AWS IoT Greengrass nicht die Installation der AWS IoT Greengrass Core-Software mit Hardwaresicherheit, wenn Sie die Installation mit automatischer Ressourcenbereitstellung oder AWS IoT Flottenbereitstellung durchführen.

Konfigurieren Sie die 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 benötigen Root-Zugriff auf das Dateisystem des Kerngeräts.
> - Bei diesem Verfahren fahren Sie die AWS IoT Greengrass Core-Software herunter, sodass das Kerngerät während der Konfiguration der Hardwaresicherheit offline und nicht verfügbar ist.

Gehen Sie wie folgt vor, um die Hardwaresicherheit auf einem vorhandenen Kerngerät zu konfigurieren:

1. Initialisieren Sie das HSM.
2. Stellen Sie die PKCS #11 -Anbieterkomponente auf dem Kerngerät bereit.

3. Stoppen Sie die AWS IoT Greengrass Core-Software.
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 Greengrass Core-Software.

Schritt 1: Initialisieren Sie das Hardware-Sicherheitsmodul

Führen Sie den folgenden Schritt aus, um das HSM auf Ihrem Kerngerät zu initialisieren.

Um das Hardware-Sicherheitsmodul zu initialisieren

- Initialisieren Sie ein PKCS #11 -Token im HSM und speichern Sie die Steckplatz-ID und die Benutzer-PIN für das Token. In der Dokumentation zu Ihrem HSM erfahren Sie, wie Sie ein Token initialisieren. Sie verwenden die Steckplatz-ID und die Benutzer-PIN später, wenn Sie die PKCS #11 -Anbieterkomponente bereitstellen und konfigurieren.

Schritt 2: Stellen Sie die PKCS #11 -Anbieterkomponente bereit

Gehen Sie wie folgt vor, um die PKCS #11 -Anbieterkomponente bereitzustellen und zu konfigurieren. Sie können die Komponente auf einem oder mehreren Kerngeräten bereitstellen.

So stellen Sie die PKCS #11 -Anbieterkomponente (Konsole) bereit

1. Wählen Sie im Navigationsmenü der AWS IoT Greengrass Konsole die Option Components aus.
2. Wählen Sie auf der Seite Komponenten die Registerkarte Öffentliche Komponenten und wählen Sie dann aws.greengrass.crypto.Pkcs11Provider.
3. Auf der aws.greengrass.crypto.Pkcs11ProviderWählen Sie auf der Seite Deploy aus.
4. Wählen Sie unter Zur Bereitstellung hinzufügen eine vorhandene Bereitstellung aus, die Sie überarbeiten möchten, oder erstellen Sie eine neue Bereitstellung und klicken Sie dann auf Weiter.
5. Wenn Sie eine neue Bereitstellung erstellen möchten, wählen Sie das Ziel-Core-Gerät oder die Dinggruppe für die Bereitstellung aus. Wählen Sie auf der Seite „Ziel angeben“ unter Bereitstellungsziel ein Kerngerät oder eine Dinggruppe aus, und klicken Sie dann auf Weiter.
6. Wählen Sie auf der Seite Komponenten auswählen unter Öffentliche Komponenten die Option aws.greengrass.crypto.Pkcs11Provider, und wählen Sie dann Weiter.

7. Wählen Sie auf der Seite Komponenten konfigurieren aws.greengrass.crypto.Pkcs11Provider, und gehen Sie dann wie folgt vor:

   a. Wählen Sie Komponente konfigurieren aus.

   b. In der Konfiguration aws.greengrass.crypto.Pkcs11ProviderModal, unter Konfigurationsupdate, unter Konfiguration zum Zusammenführen, geben Sie das folgende Konfigurationsupdate ein. Aktualisieren Sie die folgenden Konfigurationsparameter mit Werten für die Kernzielgeräte. Geben Sie die Steckplatz-ID und die Benutzer-PIN an, mit der Sie das PKCS #11 -Token zuvor initialisiert haben. Sie importieren den privaten Schlüssel und das Zertifikat später in diesen Steckplatz 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 Steckplatzes, der den privaten Schlüssel und das Gerätezertifikat enthält. Dieser Wert unterscheidet sich vom Steckplatzindex oder der Steckplatzbezeichnung.

      `userPin`

      Die Benutzer-PIN, die für den Zugriff auf den Steckplatz 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 klicken Sie dann auf Weiter.

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.

Es kann bis zu einer Minute dauern, bis die Bereitstellung abgeschlossen ist.

Um die PKCS #11 -Anbieterkomponente ()AWS CLI bereitzustellen

Um die PKCS #11 -Anbieterkomponente bereitzustellen, erstellen Sie ein Bereitstellungsdokument, das das `components` Objekt enthält`aws.greengrass.crypto.Pkcs11Provider`, und geben Sie das Konfigurationsupdate für die Komponente an. Folgen Sie den Anweisungen unterErstellen von Bereitstellungen, um ein neues Deployment zu erstellen oder ein bestehendes Deployment zu überarbeiten.

Im folgenden Beispieldokument zur teilweisen Bereitstellung wird angegeben, dass die PKCS #11 -Anbieterkomponente bereitgestellt und konfiguriert werden soll. Aktualisieren Sie die folgenden Konfigurationsparameter mit Werten für die Core-Zielgeräte. Speichern Sie die Steckplatz-ID und die Benutzer-PIN, um sie später zu verwenden, wenn Sie den privaten Schlüssel und das Zertifikat in das HSM importieren.

`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 Steckplatzes, der den privaten Schlüssel und das Gerätezertifikat enthält. Dieser Wert unterscheidet sich vom Steckplatzindex oder der Steckplatzbezeichnung.

`userPin`

Die Benutzer-PIN, die für den Zugriff auf den Steckplatz 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 Dienst verwenden, um den Status der Bereitstellung zu überprüfen. Sie können anhand der AWS IoT Greengrass Core-Softwareprotokolle überprüfen, ob die PKCS #11 - Anbieterkomponente erfolgreich bereitgestellt wurde. Weitere Informationen finden Sie hier:

- Überprüfen Sie den Bereitstellungsstatus
- AWS IoT Greengrass Protokolle überwachen

Wenn die Bereitstellung fehlschlägt, können Sie Fehler bei der Bereitstellung auf jedem Kerngerät beheben. Weitere Informationen finden Sie unter Problembehebung AWS IoT Greengrass V2.

Schritt 3: Aktualisieren Sie die Konfiguration auf dem Kerngerät

Die AWS IoT Greengrass Core-Software verwendet eine Konfigurationsdatei, die festlegt, wie das Gerät funktioniert. Diese Konfigurationsdatei enthält den Speicherort des privaten Schlüssels und des Zertifikats, mit denen das Gerät eine Verbindung herstellt AWS Cloud. Gehen Sie wie folgt vor, um den privaten Schlüssel und das Zertifikat des Kerngeräts in das HSM zu importieren und die Konfigurationsdatei für die Verwendung des HSM zu aktualisieren.

Um die Konfiguration auf dem Kerngerät zu aktualisieren, um Hardware-Sicherheit zu verwenden

1. Stoppen Sie die AWS IoT Greengrass Core-Software. Wenn Sie die AWS IoT Greengrass Core-Software als Systemdienst 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 Kerngeräts.

   - Wenn Sie die AWS IoT Greengrass Core-Software mit automatischer Bereitstellung oder Flottenbereitstellung installiert haben, befindet sich der private Schlüssel unter */greengrass/v2*/privKey.key und das Zertifikat unter. */greengrass/v2*/thingCert.crt
   - Wenn Sie die AWS IoT Greengrass Core-Software mit manueller Bereitstellung installiert haben, ist der private Schlüssel */greengrass/v2*/private.pem.key standardmäßig und das Zertifikat standardmäßig vorhanden. */greengrass/v2*/device.pem.crt

   Sie können auch die `system.certificateFilePath` Eigenschaften `system.privateKeyPath` und überprüfen*/greengrass/v2*/config/effectiveConfig.yaml, um den Speicherort dieser Dateien zu ermitteln.

3. Importieren Sie den privaten Schlüssel und das Zertifikat in das HSM. In der Dokumentation zu Ihrem HSM erfahren Sie, wie Sie private Schlüssel und Zertifikate in das HSM importieren. Importieren Sie den privaten Schlüssel und das Zertifikat mit der Steckplatz-ID und der Benutzer-PIN, mit 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 Import der einzelnen Dateien angeben. Sie verwenden dieses Label 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 Sicherungskopie der AWS IoT Greengrass Core-Konfigurationsdatei. Sie können dieses Backup verwenden, um das Kerngerät wiederherzustellen, falls Sie bei der Konfiguration der Hardwaresicherheit auf Probleme stoßen.

      ```
      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. */greengrass/v2*Ersetzen Sie 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. *iotdevicekey*Ersetzen Sie ihn durch die Objektbezeichnung, in die Sie zuvor den privaten Schlüssel und das Zertifikat 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. *iotdevicekey*Ersetzen Sie ihn durch die Objektbezeichnung, in die Sie zuvor den privaten Schlüssel und das Zertifikat 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 den `Greengrass.jar` Befehl mit dem `--init-config` Parameter aus, in dem die Konfiguration angewendet werden soll`effectiveConfig.yaml`. */greengrass/v2*Ersetzen Sie 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 Greengrass Core-Software. Wenn Sie die AWS IoT Greengrass Core-Software als Systemdienst 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 Führen Sie die AWS IoT Greengrass Core-Software aus.

7. Überprüfen Sie die AWS IoT Greengrass Core-Softwareprotokolle, um sicherzustellen, dass die Software gestartet wird und eine Verbindung zum AWS Cloud herstellt. Die AWS IoT Greengrass Core-Software verwendet den privaten Schlüssel und das Zertifikat, um eine Verbindung zu den AWS IoT und AWS IoT Greengrass -Diensten herzustellen.

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

   Die folgenden Protokollmeldungen auf Infoebene weisen darauf hin, dass die AWS IoT Greengrass Core-Software erfolgreich eine Verbindung zu den AWS IoT AND-Diensten herstellt. AWS IoT Greengrass

   ```
   2021-12-06T22:47:53.702Z [INFO] (Thread-3)
    com.aws.greengrass.mqttclient.AwsIotMqttClient: Successfully connected to AWS IoT
    Core. {clientId=MyGreengrassCore5, sessionPresent=false}
   ```

8. (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 zu den privaten Schlüssel- und Zertifikatsdateien.

   ```
   sudo rm /greengrass/v2/privKey.key
   sudo rm /greengrass/v2/thingCert.crt
   ```

Verwenden Sie Hardware ohne PKCS #11 -Unterstützung

Die PKCS#11-Bibliothek wird typischerweise vom Hardwarehersteller bereitgestellt oder ist Open Source. Bei standardkonformer Hardware (wie TPM1 .2) könnte es beispielsweise möglich sein, vorhandene Open-Source-Software zu verwenden. Wenn Ihre Hardware jedoch nicht über eine entsprechende PKCS #11 -Bibliotheksimplementierung verfügt oder wenn Sie einen

benutzerdefinierten PKCS #11 -Anbieter schreiben möchten, wenden Sie sich mit Fragen zur Integration an Ihren Amazon Web Services Enterprise Support-Mitarbeiter.

Weitere Informationen finden Sie auch unter

- PKCS #11 — Nutzungshandbuch für die kryptografische Token-Schnittstelle 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 Vorgängen, die für AWS IoT Geräte zulässig sind. Insbesondere erlauben und verweigern sie den Zugriff auf und Operationen auf AWS IoT Core der AWS IoT Greengrass Datenebene, wie das Veröffentlichen von MQTT-Nachrichten und das Abrufen von Geräteschatten.

Alle Geräte benötigen einen Eintrag in der AWS IoT Core Registrierung und ein aktiviertes X.509-Zertifikat mit einer angehängten Richtlinie. AWS IoT Geräte fallen in zwei Kategorien:

- Greengrass-Kerngeräte

  Greengrass-Core-Geräte verwenden Zertifikate und AWS IoT Richtlinien, um eine Verbindung zu AWS IoT Core und AWS IoT Greengrass herzustellen. Die Zertifikate und Richtlinien ermöglichen auch AWS IoT Greengrass die Bereitstellung von Komponenten und Konfigurationen auf Kerngeräten.
- Client-Geräte

MQTT-Client-Geräte verwenden Zertifikate und Richtlinien, um sich mit dem AWS IoT Greengrass Dienst zu AWS IoT Core verbinden. Auf diese Weise können Client-Geräte die AWS IoT Greengrass Cloud-Erkennung verwenden, um ein Greengrass-Core-Gerät zu finden und eine Verbindung zu diesem herzustellen. Ein Client-Gerät verwendet dasselbe Zertifikat, um eine Verbindung zum AWS IoT Core Cloud-Dienst und zu den Kerngeräten herzustellen. Client-Geräte verwenden außerdem Erkennungsinformationen für die gegenseitige Authentifizierung mit dem Kerngerät. Weitere Informationen finden Sie unter Interagieren Sie mit lokalen IoT-Geräten.

X.509-Zertifikate

Die Kommunikation zwischen Kerngeräten und Client-Geräten sowie zwischen Geräten AWS IoT Core und/oder AWS IoT Greengrass muss authentifiziert werden. Diese gegenseitige Authentifizierung basiert auf registrierten X.509-Gerätezertifikaten und kryptografischen Schlüsseln.

In einer AWS IoT Greengrass Umgebung verwenden Geräte Zertifikate mit öffentlichen und privaten Schlüsseln für die folgenden Transport Layer Security (TLS) -Verbindungen:

- Die AWS IoT Client-Komponente auf dem Greengrass-Core-Gerät, 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 herstellen, um Kerngeräte zu erkennen.
- Die MQTT-Broker-Komponente auf dem Greengrass-Core, die über das lokale Netzwerk eine Verbindung zu Greengrass-Geräten in der Gruppe herstellt.

AWS IoT Greengrass Kerngeräte speichern Zertifikate im Greengrass-Stammordner.

CA-Zertifikate

Greengrass-Core-Geräte und Client-Geräte laden ein Root-CA-Zertifikat herunter, das für die Authentifizierung bei den AWS IoT Core AWS IoT Greengrass AND-Diensten verwendet wird. Es wird empfohlen, ein Amazon Trust Services (ATS) CA-Stammzertifikat wie Amazon Root CA 1 zu verwenden. Weitere Informationen finden Sie unter CA-Zertifikate für die Serverauthentifizierung im AWS IoT Core -Entwicklerhandbuch.

Client-Geräte laden auch ein CA-Zertifikat für Greengrass-Core-Geräte herunter. Sie verwenden dieses Zertifikat, um das MQTT-Serverzertifikat auf dem Kerngerät während der gegenseitigen Authentifizierung zu validieren.

Zertifikatsrotation auf dem lokalen MQTT-Broker

Wenn Sie die Unterstützung von Client-Geräten aktivieren, generieren Greengrass-Core-Geräte ein lokales MQTT-Serverzertifikat, das Client-Geräte für die gegenseitige Authentifizierung verwenden. Dieses Zertifikat ist mit dem CA-Zertifikat des Kerngeräts signiert, das das Kerngerät in der AWS IoT Greengrass Cloud speichert. Client-Geräte rufen das CA-Zertifikat für das Kerngerät ab, wenn sie das Kerngerät erkennen. Sie verwenden das CA-Zertifikat des Kerngeräts, um das MQTT-Serverzertifikat des Kerngeräts zu verifizieren, wenn sie eine Verbindung zum Kerngerät herstellen. Das CA-Zertifikat für das Kerngerät läuft nach 5 Jahren ab.

Das MQTT-Serverzertifikat läuft standardmäßig alle 7 Tage ab, und Sie können diese Dauer auf 2 bis 10 Tage konfigurieren. Dieser begrenzte Zeitraum basiert auf bewährten Sicherheitsmethoden. Diese Rotation trägt dazu bei, die Gefahr zu verringern, dass ein Angreifer das MQTT-Serverzertifikat und den privaten Schlüssel stiehlt, um sich als das Greengrass-Core-Gerät auszugeben.

Das Greengrass-Core-Gerät rotiert das MQTT-Serverzertifikat 24 Stunden vor dessen 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 sich nach kurzer Zeit wieder mit dem Greengrass Core-Gerät verbinden.

AWS IoT Richtlinien für den Betrieb 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 ermöglicht den Betrieb von Greengrass-Geräten. Weitere Informationen finden Sie unter AWS IoT Greengrass V2 politische Aktionen. Zu diesen Vorgängen gehört die Fähigkeit, Komponentenabhängigkeiten aufzulösen und Artefakte öffentlicher Komponenten herunterzuladen.

Eine AWS IoT Richtlinie ist ein JSON-Dokument, das einer IAM-Richtlinie ähnelt. Sie enthält eine oder mehrere Richtlinienanweisungen, die die folgenden Eigenschaften angeben:

- `Effect`. Der Zugriffsmodus, der `Allow` oder sein kann`Deny`.
- `Action`. Die Liste der Aktionen, die von der Richtlinie zugelassen oder verweigert werden.
- `Resource`. Die Liste der Ressourcen, für die die Aktion zulässig oder verweigert ist.

AWS IoT Richtlinien unterstützen * als Platzhalterzeichen und behandeln MQTT-Platzhalterzeichen (+und#) als wörtliche Zeichenketten. Weitere Informationen zum Platzhalter finden Sie unter Verwenden von * Platzhaltern in Ressourcen im Benutzerhandbuch. ARNs AWS Identity and Access Management

Weitere Informationen finden Sie unter AWS IoT -Richtlinien und AWS IoT -Richtlinienaktionen im AWS IoT Core -Entwicklerhandbuch.

> **Important**
>
> Ding-Richtlinienvariablen (`iot:Connection.Thing.*`) werden in AWS IoT Richtlinien für Kerngeräte oder Greengrass-Datenebenenoperationen nicht unterstützt. Stattdessen können Sie einen Platzhalter verwenden, der für mehrere Geräte mit ähnlichen Namen steht. Sie können beispielsweise angeben, ob `MyGreengrassDevice*` ein Treffer gefunden werden soll `MyGreengrassDevice1MyGreengrassDevice2`, usw.

> **Note**
>
> AWS IoT Core ermöglicht es Ihnen, AWS IoT Dinggruppen Richtlinien zuzuweisen, um Berechtigungen für Gerätegruppen zu definieren. Dinggruppenrichtlinien ermöglichen keinen Zugriff auf AWS IoT Greengrass Datenebenenoperationen. Um einer Sache den Zugriff auf einen Vorgang auf der AWS IoT Greengrass Datenebene zu gewähren, fügen Sie die Berechtigung zu einer AWS IoT Richtlinie hinzu, die Sie dem Zertifikat der Sache hinzufügen.

AWS IoT Greengrass V2 politische Aktionen

AWS IoT Greengrass V2 definiert die folgenden Richtlinienaktionen, die Greengrass-Kerngeräte und Client-Geräte in AWS IoT Richtlinien verwenden können. Um eine Ressource für eine Richtlinienaktion anzugeben, verwenden Sie den Amazon-Ressourcennamen (ARN) der Ressource.

Hauptaktionen auf dem Gerät

`greengrass:GetComponentVersionArtifact`

Erteilt die Erlaubnis, eine vorsignierte URL zum Herunterladen eines öffentlichen Komponentenartefakts oder eines Lambda-Komponentenartefakts abzurufen.

Diese Berechtigung wird ausgewertet, wenn ein Kerngerät eine Bereitstellung erhält, die eine öffentliche Komponente oder ein Lambda mit Artefakten spezifiziert. Wenn das Kerngerät bereits über das Artefakt verfügt, lädt es das Artefakt nicht erneut herunter.

Art der Ressource: `componentVersion`

ARN-Format der Ressource: `arn:aws:greengrass:`*`region`*`:`*`account-id`*`:components:`*`component-name`*`:versions:`*`component-version`*

`greengrass:ResolveComponentCandidates`

Erteilt die Berechtigung, eine Liste von Komponenten zu identifizieren, die die Komponenten-, Versions- und Plattformanforderungen für eine Bereitstellung erfüllen. Wenn die Anforderungen miteinander in Konflikt stehen 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 Kerngerät eine Bereitstellung erhält, die Komponenten spezifiziert.

Ressourcentyp: Keiner

ARN-Format der Ressource: *

`greengrass:GetDeploymentConfiguration`

Erteilt die Erlaubnis, eine vorsignierte URL zum Herunterladen eines umfangreichen Bereitstellungsdokuments abzurufen.

Diese Berechtigung wird ausgewertet, wenn ein Kerngerät eine Bereitstellung empfängt, die ein Bereitstellungsdokument mit einer Größe von mehr als 7 KB (wenn die Bereitstellung auf ein Objekt abzielt) oder 31 KB (wenn die Bereitstellung auf eine Dinggruppe abzielt) angibt. Das Bereitstellungsdokument umfasst Komponentenkonfigurationen, Bereitstellungsrichtlinien und Bereitstellungsmetadaten. Weitere Informationen finden Sie unter AWS IoT Greengrass Komponenten auf Geräten bereitstellen.

Diese Funktion ist für Version 2.3.0 und höher der Greengrass Nucleus-Komponente verfügbar.

Ressourcentyp: Keiner

ARN-Format der Ressource: *

`greengrass:ListThingGroupsForCoreDevice`

Erteilt die Berechtigung, die Dinggruppenhierarchie eines Kerngeräts abzurufen.

Diese Berechtigung wird überprüft, wenn ein Kerngerät eine Bereitstellung von erhält AWS IoT Greengrass. Das Kerngerät verwendet diese Aktion, um festzustellen, ob es seit der letzten Bereitstellung aus einer Dinggruppe entfernt wurde. Wenn das Kerngerät aus einer Dinggruppe entfernt wurde und diese Dinggruppe das Ziel einer Bereitstellung auf dem Kerngerät ist, entfernt das Kerngerät die durch diese Bereitstellung installierten Komponenten.

Diese Funktion wird von Version 2.5.0 und höher der Greengrass Nucleus-Komponente verwendet.

Ressourcentyp: `thing` (Kerngerät)

ARN-Format der Ressource: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`core-device-thing-name`*

`greengrass:VerifyClientDeviceIdentity`

Erteilt die Berechtigung zur Überprüfung der Identität eines Client-Geräts, das eine Verbindung zu einem Core-Gerät herstellt.

Diese Berechtigung wird ausgewertet, wenn ein Kerngerät die Authentifizierungskomponente für das Client-Gerät ausführt und eine MQTT-Verbindung von einem Client-Gerät empfängt. Das Client-Gerät präsentiert sein AWS IoT Gerätezertifikat. Anschließend sendet das Kerngerät das Gerätezertifikat an den AWS IoT Greengrass Cloud-Dienst, um die Identität des Client-Geräts zu überprüfen. Weitere Informationen finden Sie unter Interagieren Sie mit lokalen IoT-Geräten.

Ressourcentyp: Keiner

ARN-Format der Ressource: *

`greengrass:VerifyClientDeviceIoTCertificateAssociation`

Erteilt die Berechtigung, zu überprüfen, ob ein Client-Gerät mit einem AWS IoT Zertifikat verknüpft ist.

Diese Berechtigung wird ausgewertet, wenn ein Kerngerät die Authentifizierungskomponente für das Clientgerät ausführt und ein Client-Gerät autorisiert, eine Verbindung über MQTT herzustellen. Weitere Informationen finden Sie unter Interagieren Sie mit lokalen IoT-Geräten.

> **Note**
>
> Damit ein Core-Gerät diesen Vorgang nutzen kann, muss die Greengrass-Servicerolle mit Ihrer verknüpft sein AWS-Konto und die `iot:DescribeCertificate` entsprechende Genehmigung erteilt werden.

Ressourcentyp: `thing` (Client-Gerät)

ARN-Format der Ressource: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`client-device-thing-name`*

`greengrass:PutCertificateAuthorities`

Erteilt die Berechtigung zum Hochladen von Zertifikaten der Zertifizierungsstelle (CA), die Client-Geräte herunterladen können, um das Kerngerät zu verifizieren.

Diese Berechtigung wird ausgewertet, wenn ein Kerngerät die Authentifizierungskomponente für Clientgeräte installiert und ausführt. Diese Komponente erstellt eine lokale Zertifizierungsstelle und verwendet diesen Vorgang, um ihre CA-Zertifikate hochzuladen. Client-Geräte laden diese CA-Zertifikate herunter, wenn sie den Discover-Vorgang verwenden, um Kerngeräte zu finden, mit denen sie eine Verbindung herstellen können. Wenn Client-Geräte eine Verbindung zu einem MQTT-Broker auf einem Kerngerät herstellen, verwenden sie diese CA-Zertifikate, um die Identität des Kerngeräts zu überprüfen. Weitere Informationen finden Sie unter Interagieren Sie mit lokalen IoT-Geräten.

Ressourcentyp: Keiner

ARN-Format: *

`greengrass:GetConnectivityInfo`

Erteilt die Berechtigung zum Abrufen von Konnektivitätsinformationen für ein Kerngerät. Diese Informationen beschreiben, wie Client-Geräte eine Verbindung zum Kerngerät herstellen können.

Diese Berechtigung wird ausgewertet, wenn ein Kerngerät die Authentifizierungskomponente für das Clientgerät installiert und ausführt. Diese Komponente verwendet die Konnektivitätsinformationen, um gültige CA-Zertifikate zu generieren, die während des PutCertificateAuthoriesVorgangs in den AWS IoT Greengrass Cloud-Dienst hochgeladen werden. Client-Geräte verwenden diese CA-Zertifikate, um die Identität des Kerngeräts zu überprüfen. Weitere Informationen finden Sie unter Interagieren Sie mit lokalen IoT-Geräten.

Sie können diesen Vorgang auch auf der AWS IoT Greengrass Steuerungsebene verwenden, um Konnektivitätsinformationen für ein Kerngerät anzuzeigen. Weitere Informationen finden Sie unter GetConnectivityInfo in der AWS IoT Greengrass V1 -API-Referenz.

Ressourcentyp: `thing` (Kerngerät)

ARN-Format der Ressource: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`core-device-thing-name`*

`greengrass:UpdateConnectivityInfo`

Erteilt die Erlaubnis, die Konnektivitätsinformationen für ein Kerngerät zu aktualisieren. Diese Informationen beschreiben, wie Client-Geräte eine Verbindung zum Kerngerät herstellen können.

Diese Berechtigung wird ausgewertet, wenn auf einem Core-Gerät die IP-Detektorkomponente ausgeführt wird. Diese Komponente identifiziert die Informationen, die Client-Geräte benötigen, um eine Verbindung zum Kerngerät im lokalen Netzwerk herzustellen. Anschließend verwendet diese Komponente diesen Vorgang, um die Konnektivitätsinformationen in den AWS IoT Greengrass Cloud-Dienst hochzuladen, sodass Client-Geräte diese Informationen mit dem Discover-Vorgang abrufen können. Weitere Informationen finden Sie unter Interagieren Sie mit lokalen IoT-Geräten.

Sie können diesen Vorgang auch auf der AWS IoT Greengrass Steuerungsebene verwenden, um die Konnektivitätsinformationen für ein Kerngerät manuell zu aktualisieren. Weitere Informationen finden Sie unter UpdateConnectivityInfo in der AWS IoT Greengrass V1 -API-Referenz.

Ressourcentyp: `thing` (Kerngerät)

ARN-Format der Ressource: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`core-device-thing-name`*

Aktionen auf Client-Geräten

`greengrass:Discover`

Erteilt die Berechtigung, Verbindungsinformationen für Kerngeräte zu ermitteln, mit denen ein Clientgerät eine Verbindung herstellen kann. Diese Informationen beschreiben, wie das Clientgerät eine Verbindung zu den Kerngeräten herstellen kann. Ein Clientgerät kann mithilfe des BatchAssociateClientDeviceWithCoreDeviceVorgangs nur die Kerngeräte erkennen, denen Sie es zugeordnet haben. Weitere Informationen finden Sie unter Interagieren Sie mit lokalen IoT-Geräten.

Ressourcentyp: `thing` (Client-Gerät)

ARN-Format der Ressource: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`client-device-thing-name`*

Aktualisieren Sie die AWS IoT Richtlinie eines Kerngeräts

Sie können die AWS IoT Greengrass und AWS IoT -Konsolen oder die AWS IoT API verwenden, um die AWS IoT Richtlinien eines Kerngeräts einzusehen und zu aktualisieren.

> **Note**
>
> Wenn Sie das AWS IoT Greengrass Core-Softwareinstallationsprogramm zur Bereitstellung von Ressourcen verwendet haben, verfügt Ihr Kerngerät über eine AWS IoT Richtlinie, die den Zugriff auf alle AWS IoT Greengrass Aktionen ermöglicht (`greengrass:*`). Sie können diese Schritte befolgen, um den Zugriff nur auf die Aktionen zu beschränken, die ein Kerngerät verwendet.

Überprüfen und aktualisieren Sie die AWS IoT Richtlinie eines Kerngeräts (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 Seite mit den Details zum Kerngerät den Link zum Ding des Kerngeräts aus. Dieser Link öffnet die Seite mit den Ding-Details in der AWS IoT Konsole.
4. Wählen Sie auf der Seite mit den Ding-Details die Option Zertifikate aus.
5. Wählen Sie auf der Registerkarte Zertifikate das aktive Zertifikat der Sache aus.
6. Wählen Sie auf der Seite mit den Zertifikatsdetails die Option Richtlinien aus.
7. Wählen Sie auf der Registerkarte Richtlinien die AWS IoT Richtlinie aus, die Sie überprüfen und aktualisieren möchten. Sie können jeder Richtlinie, die mit dem aktiven Zertifikat des Kerngeräts verknüpft ist, die erforderlichen Berechtigungen hinzufügen.

   > **Note**
   >
   > Wenn Sie das AWS IoT Greengrass Core-Softwareinstallationsprogramm zur Bereitstellung von Ressourcen verwendet haben, gibt es zwei AWS IoT Richtlinien.

Wir empfehlen, dass Sie die genannte Richtlinie wählen GreengrassV2IoTThingPolicy, falls sie existiert. Kerngeräte, die Sie mit dem Schnellinstallationsprogramm erstellen, verwenden standardmäßig diesen Richtliniennamen. Wenn Sie dieser Richtlinie Berechtigungen hinzufügen, gewähren Sie diese Berechtigungen auch anderen Kerngeräten, die diese Richtlinie verwenden.

8. Wählen Sie in der Richtlinienübersicht die Option Aktive Version bearbeiten aus.
9. Überprüfen Sie die Richtlinie und fügen Sie nach Bedarf Berechtigungen hinzu, entfernen oder bearbeiten Sie sie.
10. Um eine neue Richtlinienversion als aktive Version festzulegen, wählen Sie unter Status der Richtlinienversion die Option Die bearbeitete Version als aktive Version für diese Richtlinie festlegen aus.
11. Wählen Sie Als neue Version speichern aus.

Überprüfen und aktualisieren Sie die AWS IoT Richtlinie eines Kerngeräts (AWS CLI)

1. Führe die Principals für das Gerät auf, für das das Kerngerät zuständig ist AWS IoT . Bei den Ding-Prinzipalen kann es sich um X.509-Gerätezertifikate oder andere Identifikationen handeln. Führen Sie den folgenden Befehl aus und *MyGreengrassCore* ersetzen Sie ihn durch den Namen des Kerngeräts.

   ```
   aws iot list-thing-principals --thing-name MyGreengrassCore
   ```

   Der Vorgang gibt eine Antwort zurück, in der die Ding-Prinzipale des Kerngeräts aufgeführt sind.

   ```
   {
       "principals": [
           "arn:aws:iot:us-west-2:123456789012:cert/certificateId"
       ]
   }
   ```

2. Identifizieren Sie das aktive Zertifikat des Kerngeräts. Führen Sie den folgenden Befehl aus und *certificateId* ersetzen Sie ihn durch die ID jedes Zertifikats aus dem vorherigen Schritt, bis Sie das aktive Zertifikat gefunden haben. Die Zertifikat-ID ist die hexadezimale Zeichenfolge am Ende des Zertifikat-ARN. Das `--query` Argument gibt an, dass nur der Status des Zertifikats ausgegeben werden soll.

```
aws iot describe-certificate --certificate-id certificateId --query
 'certificateDescription.status'
```

Die Operation gibt den Status des Zertifikats als Zeichenfolge zurück. Wenn das Zertifikat beispielsweise aktiv ist, wird bei diesem Vorgang eine Ausgabe ausgegeben`"ACTIVE"`.

3. Listet die AWS IoT Richtlinien auf, die an das Zertifikat angehängt 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 an das Zertifikat angehängt 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 Richtlinie aus, die Sie anzeigen und aktualisieren möchten.

> **Note**
>
> Wenn Sie das AWS IoT Greengrass Core-Softwareinstallationsprogramm zur Bereitstellung von Ressourcen verwendet haben, gelten zwei AWS IoT Richtlinien. Wir empfehlen, dass Sie die genannte Richtlinie wählen GreengrassV2IoTThingPolicy, falls sie existiert. Kerngeräte, die Sie mit dem Schnellinstallationsprogramm erstellen, verwenden standardmäßig diesen Richtliniennamen. Wenn Sie dieser Richtlinie

Berechtigungen hinzufügen, gewähren Sie diese Berechtigungen auch anderen Kerngeräten, die diese Richtlinie verwenden.

5. Holen Sie sich das Dokument der Richtlinie. Führen Sie den folgenden Befehl aus und *GreengrassV2IoTThingPolicy* ersetzen Sie ihn durch den Namen der Richtlinie.

```
aws iot get-policy --policy-name GreengrassV2IoTThingPolicy
```

Der Vorgang gibt eine Antwort zurück, die das Dokument der Richtlinie und andere Informationen zur Richtlinie enthält. Das Richtliniendokument ist ein JSON-Objekt, das als Zeichenfolge serialisiert ist.

```
{
    "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 Zeichenfolge des Richtliniendokuments in ein JSON-Objekt zu konvertieren, und speichern Sie sie dann in einer Datei mit dem Name`iot-policy.json`.

   Wenn Sie beispielsweise das jq-Tool 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 nach Bedarf Berechtigungen hinzu, entfernen oder bearbeiten Sie sie.

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

   ```
   nano iot-policy.json
   ```

   Wenn Sie fertig sind, ähnelt das Richtliniendokument möglicherweise der AWS IoT Minimalrichtlinie für Kerngeräte.

8. Speichern Sie die Änderungen als neue Version der Richtlinie. Führen Sie den folgenden Befehl aus und *GreengrassV2IoTThingPolicy* ersetzen Sie ihn durch den Namen der Richtlinie.

   ```
   aws iot create-policy-version --policy-name GreengrassV2IoTThingPolicy --policy-
   document file://iot-policy.json --set-as-default
   ```

   Der Vorgang gibt eine Antwort zurück, die dem folgenden Beispiel ähnelt, wenn er 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 V2 Kerngeräte

> **Important**
>
> Spätere Versionen der Greengrass Nucleus-Komponente erfordern zusätzliche Berechtigungen für die AWS IoT Minimalrichtlinie. Möglicherweise müssen Sie die AWS IoT Richtlinien Ihrer Kerngeräte aktualisieren, um zusätzliche Berechtigungen zu gewähren.
>
> - Core-Geräte, auf denen Greengrass Nucleus v2.5.0 und höher ausgeführt wird, verwenden die `greengrass:ListThingGroupsForCoreDevice` Berechtigung, Komponenten zu deinstallieren, wenn Sie ein Kerngerät aus einer Dinggruppe entfernen.
> - Core-Geräte, auf denen Greengrass Nucleus v2.3.0 und höher ausgeführt wird, verwenden die `greengrass:GetDeploymentConfiguration` Berechtigung, umfangreiche 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 hinter dem Namen des Kerngeräts (z. B.). *`core-device-thing-name`** Das Kerngerät verwendet dasselbe Gerätezertifikat, um mehrere gleichzeitige Abonnements abzuschließen AWS IoT Core, aber die Client-ID in einer Verbindung entspricht möglicherweise nicht exakt dem Namen des Kerngeräts. Nach den ersten 50 Abonnements verwendet *`core-device-thing-name`*#*`number`* das Kerngerät die Client-ID, wobei die Anzahl für alle weiteren 50 Abonnements *`number`* erhöht wird. Wenn ein Kerngerät mit

dem Namen beispielsweise 150 gleichzeitige Abonnements `MyCoreDevice` erstellt, verwendet es den 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 Clients verwendet IDs , die über ein Suffix verfügen.

- 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 Erlaubnis, Telemetriedaten zum folgenden Thema zu veröffentlichen.

  ```
  $aws/things/core-device-thing-name/greengrass/health/json
  ```

  Sie können diese Berechtigung für Kerngeräte entfernen, auf denen Sie Telemetrie deaktivieren. Weitere Informationen finden Sie unter Erfassung von Telemetriedaten zur Systemintegrität von AWS IoT Greengrass Kerngeräten.
- Die Richtlinie gewährt die Berechtigung, über einen Rollenalias eine IAM-Rolle anzunehmen. AWS IoT Das Kerngerät verwendet diese Rolle, die als Token-Austauschrolle bezeichnet wird, um AWS Anmeldeinformationen abzurufen, die es zur Authentifizierung AWS von Anfragen verwenden kann. Weitere Informationen finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

  Wenn Sie die AWS IoT Greengrass Core-Software installieren, erstellen und fügen Sie eine zweite AWS IoT Richtlinie hinzu, die nur diese Berechtigung beinhaltet. Wenn Sie diese Berechtigung in die primäre AWS IoT Richtlinie Ihres Core-Geräts aufnehmen, können Sie die andere AWS IoT Richtlinie trennen und löschen.

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "arn:aws:iot:us-east-1:123456789012:client/core-device-
thing-name*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive",
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/core-
device-thing-name/greengrass/health/json",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/core-
device-thing-name/greengrassv2/health/json",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/core-
device-thing-name/jobs/*",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/core-
device-thing-name/shadow/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/core-
device-thing-name/jobs/*",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/core-
device-thing-name/shadow/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iot:AssumeRoleWithCertificate",
            "Resource": "arn:aws:iot:us-east-1:123456789012:rolealias/token-
exchange-role-alias-name"
```

```
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:GetComponentVersionArtifact",
                "greengrass:ResolveComponentCandidates",
                "greengrass:GetDeploymentConfiguration",
                "greengrass:ListThingGroupsForCoreDevice"
            ],
            "Resource": "*"
        }
    ]
}
```

Minimale AWS IoT Richtlinie zur Unterstützung von Client-Geräten

Die folgende Beispielrichtlinie umfasst die Mindestanzahl von Aktionen, die zur Unterstützung der Interaktion mit Client-Geräten auf einem Kerngerät erforderlich sind. Um Client-Geräte zu unterstützen, muss ein Core-Gerät zusätzlich zur AWS IoT Minimalrichtlinie für den Basisbetrieb über die in dieser AWS IoT Richtlinie festgelegten Berechtigungen verfügen.

- Die Richtlinie ermöglicht es dem Kerngerät, seine eigenen Konnektivitätsinformationen zu aktualisieren. Diese Berechtigung (`greengrass:UpdateConnectivityInfo`) ist nur erforderlich, wenn Sie die IP-Detektorkomponente auf dem Kerngerät bereitstellen.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/core-
device-thing-name-gci/shadow/get"
            ]
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/core-
device-thing-name-gci/shadow/update/delta",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/core-
device-thing-name-gci/shadow/get/accepted"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/core-
device-thing-name-gci/shadow/update/delta",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/core-
device-thing-name-gci/shadow/get/accepted"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:PutCertificateAuthorities",
                "greengrass:VerifyClientDeviceIdentity"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:VerifyClientDeviceIoTCertificateAssociation"
            ],
            "Resource": "arn:aws:iot:us-east-1:123456789012:thing/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:GetConnectivityInfo",
                "greengrass:UpdateConnectivityInfo"
```

```
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:thing/core-device-thing-name"
            ]
        }
    ]
}
```

Minimale AWS IoT Richtlinie für Client-Geräte

Die folgende Beispielrichtlinie umfasst die Mindestanzahl an Aktionen, die ein Client-Gerät benötigt, um Kerngeräte zu erkennen, mit denen es sich über MQTT verbindet und kommuniziert. Die AWS IoT Richtlinie des Client-Geräts muss die `greengrass:Discover` Aktion beinhalten, die es dem Gerät ermöglicht, Verbindungsinformationen für die zugehörigen Greengrass-Core-Geräte zu ermitteln. Geben Sie in `Resource` diesem 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 die besten Sicherheitspraktiken einzuhalten, beschränken Sie die `iot:Publish` `iot:Receive` Berechtigungen`iot:Subscribe`, und auf die Mindestanzahl von Themen, die ein Client-Gerät für Ihren Anwendungsfall benötigt.
- Die Richtlinie ermöglicht es dem Gerät, zentrale Geräte für alle AWS IoT Dinge zu erkennen. Um die besten Sicherheitsvorkehrungen zu befolgen, beschränken Sie die `greengrass:Discover` Zugriffsrechte auf das AWS IoT Ding des Client-Geräts oder auf einen Platzhalter, der einer Reihe von AWS IoT Dingen entspricht.

  > **Important**
  >
  > Ding-Richtlinienvariablen (`iot:Connection.Thing.*`) werden in AWS IoT Richtlinien für Kerngeräte oder Greengrass-Datenebenenoperationen nicht unterstützt. Stattdessen können Sie einen Platzhalter verwenden, der für mehrere Geräte mit ähnlichen Namen steht. Sie können beispielsweise angeben, ob `MyGreengrassDevice*` ein Treffer gefunden werden soll `MyGreengrassDevice1MyGreengrassDevice2`, usw.

- Die AWS IoT Richtlinie eines Client-Geräts erfordert normalerweise keine Berechtigungen oder `iot:DeleteThingShadow` Aktionen `iot:GetThingShadowiot:UpdateThingShadow`, da das Greengrass-Core-Gerät Schattensynchronisierungsvorgänge für Client-Geräte abwickelt. Damit das Core-Gerät Shadows von Client-Geräten verarbeiten kann, überprüfen Sie, ob die AWS IoT

Richtlinie des Core-Geräts diese Aktionen zulässt und ob der `Resource` Abschnitt auch die ARNs Client-Geräte enthält.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/*"
            ]
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:Discover"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:thing/*"
            ]
        }
    ]
}
```

Identitäts- und Zugriffsmanagement für AWS IoT Greengrass

AWS Identity and Access Management (IAM) hilft einem Administrator AWS-Service , den Zugriff auf Ressourcen sicher zu AWS kontrollieren. IAM-Administratoren kontrollieren, wer authentifiziert (angemeldet) und autorisiert werden kann (über Berechtigungen verfügt), um Ressourcen zu verwenden. AWS IoT Greengrass IAM ist ein Programm AWS-Service , das Sie ohne zusätzliche Kosten nutzen können.

> **Note**
>
> In diesem Thema werden die Konzepte und Funktionen von IAM beschrieben. Informationen zu den IAM-Funktionen, die von unterstützt werden AWS IoT Greengrass, finden Sie unter. the section called “Wie AWS IoT Greengrass funktioniert mit IAM”

Zielgruppe

Wie Sie AWS Identity and Access Management (IAM) verwenden, hängt von Ihrer Rolle ab:

- Servicebenutzer – Fordern Sie von Ihrem Administrator Berechtigungen an, wenn Sie nicht auf Funktionen zugreifen können (siehe Problembehandlung bei Identitäts- und Zugriffsproblemen für AWS IoT Greengrass).
- Serviceadministrator – Bestimmen Sie den Benutzerzugriff und stellen Sie Berechtigungsanfragen (siehe Wie AWS IoT Greengrass funktioniert mit IAM).
- IAM-Administrator – Schreiben Sie Richtlinien zur Zugriffsverwaltung (siehe Beispiele für identitätsbasierte Richtlinien für AWS IoT Greengrass).

Authentifizierung mit Identitäten

Authentifizierung ist die Art und Weise, wie Sie sich AWS mit Ihren Identitätsdaten anmelden. Sie müssen sich als IAM-Benutzer authentifizieren oder eine IAM-Rolle annehmen. Root-Benutzer des AWS-Kontos

Sie können sich als föderierte Identität anmelden, indem Sie Anmeldeinformationen aus einer Identitätsquelle wie AWS IAM Identity Center (IAM Identity Center), Single Sign-On-Authentifizierung oder Anmeldeinformationen verwenden. Google/Facebook Weitere Informationen zum Anmelden finden Sie unter So melden Sie sich bei Ihrem AWS-Konto an im Benutzerhandbuch für AWS-Anmeldung .

AWS Bietet für den programmatischen Zugriff ein SDK und eine CLI zum kryptografischen Signieren von Anfragen. Weitere Informationen finden Sie unter AWS Signature Version 4 for API requests im IAM-Benutzerhandbuch.

AWS-Konto Root-Benutzer

Wenn Sie einen erstellen AWS-Konto, beginnen Sie mit einer Anmeldeidentität, dem sogenannten AWS-Konto Root-Benutzer, der vollständigen Zugriff auf alle AWS-Services Ressourcen hat. Wir raten ausdrücklich davon ab, den Root-Benutzer für Alltagsaufgaben zu verwenden. Eine Liste der Aufgaben, für die Sie sich als Root-Benutzer anmelden müssen, finden Sie unter Tasks that require root user credentials im IAM-Benutzerhandbuch.

IAM-Benutzer und -Gruppen

Ein IAM-Benutzer ist eine Identität mit bestimmten Berechtigungen für eine einzelne Person oder Anwendung. Verwenden Sie möglichst temporäre Anmeldeinformationen statt IAM-Benutzer mit langfristigen Anmeldeinformationen. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter Erfordern, dass menschliche Benutzer für den Zugriff AWS mithilfe temporärer Anmeldeinformationen einen Verbund mit einem Identitätsanbieter verwenden müssen.

Eine IAM-Gruppe gibt eine Sammlung von IAM-Benutzern an und vereinfacht die Verwaltung von Berechtigungen bei großer Benutzerzahl. Weitere Informationen finden Sie unter Use cases for IAM users im IAM-Benutzerhandbuch.

IAM-Rollen

Eine IAM-Rolle ist eine Identität mit bestimmten Berechtigungen, die temporäre Anmeldeinformationen bereitstellt. Sie können eine Rolle übernehmen, indem Sie von einer Benutzer-

zu einer IAM-Rolle (Konsole) wechseln AWS CLI oder einen AWS API-Vorgang aufrufen. Weitere Informationen finden Sie unter Methods to assume a role im IAM-Benutzerhandbuch.

IAM-Rollen sind nützlich für Verbundbenutzerzugriff, temporäre IAM-Benutzerberechtigungen, kontoübergreifenden Zugriff, dienstübergreifenden Zugriff und Anwendungen, die auf Amazon ausgeführt werden. EC2 Weitere Informationen finden Sie unter Kontoübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch.

Verwalten des Zugriffs mit Richtlinien

Sie kontrollieren den Zugriff, AWS indem Sie Richtlinien erstellen und diese an Identitäten oder Ressourcen anhängen. AWS Eine Richtlinie definiert Berechtigungen, wenn sie mit einer Identität oder Ressource verknüpft sind. AWS bewertet diese Richtlinien, wenn ein Principal eine Anfrage stellt. Die meisten Richtlinien werden AWS als JSON-Dokumente gespeichert. Weitere Informationen zu JSON-Richtliniendokumenten finden Sie unter Overview of JSON policies im IAM-Benutzerhandbuch.

Mithilfe von Richtlinien legen Administratoren fest, wer auf was Zugriff hat, indem sie definieren, welcher Prinzipal Aktionen mit welchen Ressourcen und unter welchen Bedingungen ausführen kann.

Standardmäßig haben Benutzer, Gruppen und Rollen keine Berechtigungen. Ein IAM-Administrator erstellt IAM-Richtlinien und fügt sie Rollen hinzu, die Benutzer dann übernehmen können. IAM-Richtlinien definieren Berechtigungen unabhängig von der Methode, die zur Ausführung der Operation verwendet wird.

Identitätsbasierte Richtlinien

Identitätsbasierte Richtlinien sind JSON-Berechtigungsrichtliniendokumente, die Sie einer Identität (Benutzer, Gruppe oder Rolle) anfügen können. Diese Richtlinien steuern, welche Aktionen Identitäten für welche Ressourcen und unter welchen Bedingungen ausführen können. Informationen zum Erstellen identitätsbasierter Richtlinien finden Sie unter Definieren benutzerdefinierter IAM-Berechtigungen mit vom Kunden verwalteten Richtlinien im IAM-Benutzerhandbuch.

Identitätsbasierte Richtlinien können Inline-Richtlinien (Richtlinien, die direkt in eine einzelne Identität eingebettet sind) oder verwaltete Richtlinien (eigenständige Richtlinien, die mehreren Identitäten zugeordnet sind) sein. Informationen dazu, wie Sie zwischen verwalteten und Inline-Richtlinien wählen, finden Sie unter Choose between managed policies and inline policies im IAM-Benutzerhandbuch.

Ressourcenbasierte Richtlinien

Ressourcenbasierte Richtlinien sind JSON-Richtliniendokumente, die Sie an eine Ressource anfügen. Beispiele hierfür sind Vertrauensrichtlinien für IAM-Rollen und Bucket-Richtlinien von Amazon S3. In Services, die ressourcenbasierte Richtlinien unterstützen, können Service-Administratoren sie verwenden, um den Zugriff auf eine bestimmte Ressource zu steuern. Sie müssen in einer ressourcenbasierten Richtlinie einen Prinzipal angeben.

Ressourcenbasierte Richtlinien sind Richtlinien innerhalb dieses Diensts. Sie können AWS verwaltete Richtlinien von IAM nicht in einer ressourcenbasierten Richtlinie verwenden.

Zugriffskontrolllisten () ACLs

Zugriffskontrolllisten (ACLs) steuern, welche Principals (Kontomitglieder, Benutzer oder Rollen) über Zugriffsberechtigungen für eine Ressource verfügen. ACLs ähneln ressourcenbasierten Richtlinien, verwenden jedoch nicht das JSON-Richtliniendokumentformat.

Amazon S3 und Amazon VPC sind Beispiele für Dienste, die Unterstützung ACLs bieten. AWS WAF Weitere Informationen finden Sie unter Übersicht über ACLs die Zugriffskontrollliste (ACL) im Amazon Simple Storage Service Developer Guide.

Weitere Richtlinientypen

AWS unterstützt zusätzliche Richtlinientypen, mit denen die maximalen Berechtigungen festgelegt werden können, die durch gängigere Richtlinientypen gewährt werden:

- Berechtigungsgrenzen – Eine Berechtigungsgrenze legt die maximalen Berechtigungen fest, die eine identitätsbasierte Richtlinie einer IAM-Entität erteilen kann. Weitere Informationen finden Sie unter Berechtigungsgrenzen für IAM-Entitäten im -IAM-Benutzerhandbuch.
- Richtlinien zur Dienstkontrolle (SCPs) — Geben Sie die maximalen Berechtigungen für eine Organisation oder Organisationseinheit in an AWS Organizations. Weitere Informationen finden Sie unter Service-Kontrollrichtlinien im AWS Organizations -Benutzerhandbuch.
- Richtlinien zur Ressourcenkontrolle (RCPs) — Legen Sie die maximal verfügbaren Berechtigungen für Ressourcen in Ihren Konten fest. Weitere Informationen finden Sie im AWS Organizations Benutzerhandbuch unter Richtlinien zur Ressourcenkontrolle (RCPs).
- Sitzungsrichtlinien – Sitzungsrichtlinien sind erweiterte Richtlinien, die als Parameter übergeben werden, wenn Sie eine temporäre Sitzung für eine Rolle oder einen Verbundbenutzer erstellen. Weitere Informationen finden Sie unter Sitzungsrichtlinien im IAM-Benutzerhandbuch.

Mehrere Richtlinientypen

Wenn für eine Anfrage mehrere Arten von Richtlinien gelten, sind die daraus resultierenden Berechtigungen schwieriger zu verstehen. Informationen darüber, wie AWS bestimmt wird, ob eine Anfrage zulässig ist, wenn mehrere Richtlinientypen betroffen sind, finden Sie unter Bewertungslogik für Richtlinien im IAM-Benutzerhandbuch.

Weitere Informationen finden Sie auch unter

- the section called “Wie AWS IoT Greengrass funktioniert mit IAM”
- the section called “Beispiele für identitätsbasierte Richtlinien”
- the section called “Problembehandlung bei Identitäts- und Zugriffsproblemen”

Wie AWS IoT Greengrass funktioniert mit IAM

Bevor Sie IAM verwenden, um den Zugriff auf zu verwalten AWS IoT Greengrass, sollten Sie sich mit den IAM-Funktionen vertraut machen, die Sie mit verwenden können. AWS IoT Greengrass

IAM-Feature	Von Greengrass unterstützt?
Identitätsbasierte Richtlinien mit Berechtigungen auf Ressourcenebene	Ja
Ressourcenbasierte Richtlinien	Nein
Zugriffskontrolllisten () ACLs	Nein
Auf Tags basierende Autorisierung	Ja
Temporäre Anmeldeinformationen	Ja
Service-verknüpfte Rollen	Nein
Servicerollen	Ja

Einen allgemeinen Überblick darüber, wie andere AWS Dienste mit IAM funktionieren, finden Sie im IAM-Benutzerhandbuch unter AWS Dienste, die mit IAM funktionieren.

Identitätsbasierte Richtlinien für AWS IoT Greengrass

Mit identitätsbasierten IAM-Richtlinien können Sie zulässige oder verweigerte Aktionen und Ressourcen sowie die Bedingungen angeben, unter denen Aktionen zugelassen oder verweigert werden. AWS IoT Greengrass unterstützt bestimmte Aktionen, Ressourcen und Bedingungsschlüssel. Weitere Informationen zu allen Elementen, die Sie in einer Richtlinie verwenden, finden Sie unter Referenz zu den IAM-JSON-Richtlinienelementen im IAM-Benutzerhandbuch.

Aktionen

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer Zugriff auf was hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das Element `Action` einer JSON-Richtlinie beschreibt die Aktionen, mit denen Sie den Zugriff in einer Richtlinie zulassen oder verweigern können. Nehmen Sie Aktionen in eine Richtlinie auf, um Berechtigungen zur Ausführung des zugehörigen Vorgangs zu erteilen.

Bei Richtlinienaktionen wird das `greengrass:` Präfix vor der Aktion AWS IoT Greengrass verwendet. Um beispielsweise jemandem zu ermöglichen, mithilfe des `ListCoreDevices` API-Vorgangs die wichtigsten Geräte aufzulisten AWS-Konto, nehmen Sie die `greengrass:ListCoreDevices` Aktion in seine Richtlinie auf. Richtlinienerklärungen müssen `Action` entweder ein `NotAction` Oder-Element enthalten. AWS IoT Greengrass definiert einen eigenen Satz von Aktionen, die Aufgaben beschreiben, die Sie mit diesem Dienst ausführen können.

Um mehrere Aktionen in einer einzigen Anweisung anzugeben, listen Sie sie in Klammern (`[]`) auf und trennen Sie sie wie folgt durch Kommas:

```
"Action": [
  "greengrass:action1",
  "greengrass:action2",
  "greengrass:action3"
]
```

Sie können Platzhalter (`*`) verwenden, um mehrere Aktionen anzugeben. Beispielsweise können Sie alle Aktionen festlegen, die mit dem Wort `List` beginnen, einschließlich der folgenden Aktion:

```
"Action": "greengrass:List*"
```

> **Note**
>
> Es wird empfohlen, die Verwendung von Platzhaltern zu vermeiden, um alle verfügbaren Aktionen für einen Service anzugeben. Als bewährte Methode sollten Sie in einer Richtlinie die geringsten Berechtigungen mit eng begrenztem Umfang gewähren. Weitere Informationen finden Sie unter the section called “Erteilen von Mindestberechtigungen”.

Die vollständige Liste der AWS IoT Greengrass Aktionen finden Sie AWS IoT Greengrass im IAM-Benutzerhandbuch unter Definierte Aktionen von.

Ressourcen

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer Zugriff auf was hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das JSON-Richtlinienelement `Resource` gibt die Objekte an, auf welche die Aktion angewendet wird. Als bewährte Methode geben Sie eine Ressource mit dem zugehörigen Amazon-Ressourcennamen (ARN) an. Verwenden Sie für Aktionen, die keine Berechtigungen auf Ressourcenebene unterstützen, einen Platzhalter (*), um anzugeben, dass die Anweisung für alle Ressourcen gilt.

```
"Resource": "*"
```

Die folgende Tabelle enthält die AWS IoT Greengrass Ressource ARNs , die im `Resource` Element einer Richtlinienerklärung verwendet werden kann. Eine Zuordnung der unterstützten Berechtigungen für AWS IoT Greengrass Aktionen auf Ressourcenebene finden Sie unter Actions Defined by AWS IoT Greengrass im IAM-Benutzerhandbuch.

Einige AWS IoT Greengrass Aktionen (z. B. einige Listenoperationen) können nicht für eine bestimmte Ressource ausgeführt werden. In diesen Fällen müssen Sie allein den Platzhalter verwenden.

```
"Resource": "*"
```

Um mehrere Ressourcen ARNs in einer Anweisung anzugeben, listen Sie sie in Klammern (`[]`) auf und trennen Sie sie wie folgt durch Kommas:

```
"Resource": [
  "resource-arn1",
  "resource-arn2",
  "resource-arn3"
]
```

Weitere Informationen zu ARN-Formaten finden Sie unter Amazon Resource Names (ARNs) und AWS Service-Namespaces in der. Allgemeine Amazon Web Services-Referenz

Bedingungsschlüssel

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer Zugriff auf was hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das `Condition`-Element legt fest, ob Anweisungen auf der Grundlage definierter Kriterien ausgeführt werden. Sie können bedingte Ausdrücke erstellen, die Bedingungsoperatoren verwenden, z. B. ist gleich oder kleiner als, damit die Bedingung in der Richtlinie mit Werten in der Anforderung übereinstimmt. Eine Übersicht aller AWS globalen Bedingungsschlüssel finden Sie unter Kontextschlüssel für AWS globale Bedingungen im IAM-Benutzerhandbuch.

Beispiele

Beispiele für AWS IoT Greengrass identitätsbasierte Richtlinien finden Sie unter. the section called “Beispiele für identitätsbasierte Richtlinien”

Ressourcenbasierte Richtlinien für AWS IoT Greengrass

AWS IoT Greengrass unterstützt keine ressourcenbasierten Richtlinien.

Zugriffskontrolllisten () ACLs

AWS IoT Greengrass unterstützt nicht ACLs.

Autorisierung auf der Grundlage von AWS IoT Greengrass Tags

Sie können Tags an unterstützte AWS IoT Greengrass Ressourcen anhängen oder Tags in einer Anfrage an übergeben AWS IoT Greengrass. Um den Zugriff basierend auf Tags zu steuern,

stellen Sie Tag-Informationen im Bedingungselement einer Richtlinie unter Verwendung der Bedingungsschlüssel `aws:ResourceTag/${TagKey}`, `aws:RequestTag/${TagKey}` oder `aws:TagKeys` bereit. Weitere Informationen finden Sie unter Markieren Ihrer -Ressourcen mit Tags (Markierungen).

IAM-Rollen für AWS 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 bei einem Verbund anzumelden, eine IAM-Rolle anzunehmen oder eine kontoübergreifende Rolle anzunehmen. Sie erhalten temporäre Sicherheitsanmeldedaten, indem Sie AWS STS API-Operationen wie AssumeRoleoder aufrufen. GetFederationToken

Auf dem Greengrass-Kern werden den Greengrass-Komponenten temporäre Anmeldeinformationen für die Geräterolle zur Verfügung gestellt. Wenn Ihre Komponenten das AWS SDK verwenden, müssen Sie keine Logik hinzufügen, um die Anmeldeinformationen zu erhalten, da das AWS SDK dies für Sie erledigt.

Service-verknüpfte Rollen

AWS IoT Greengrass unterstützt keine dienstbezogenen Rollen.

Servicerollen

Dieses Feature ermöglicht einem Service das Annehmen einer Servicerolle in Ihrem Namen. Diese Rolle gewährt dem Service Zugriff auf Ressourcen in anderen Diensten, um eine Aktion in Ihrem Namen auszuführen. Servicerollen werden in Ihrem IAM-Konto angezeigt und gehören zum Konto. Dies bedeutet, dass ein IAM-Administrator die Berechtigungen für diese Rolle ändern kann. Dies kann jedoch die Funktionalität des Dienstes beeinträchtigen.

AWS IoT Greengrass Kerngeräte verwenden eine Servicerolle, damit Greengrass-Komponenten und Lambda-Funktionen in Ihrem Namen auf einige Ihrer AWS Ressourcen zugreifen können. Weitere Informationen finden Sie unter the section called “Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS”.

AWS IoT Greengrass verwendet eine Servicerolle, um in Ihrem Namen auf einige Ihrer AWS Ressourcen zuzugreifen. Weitere Informationen finden Sie unter Servicerolle bei Greengrass.

Beispiele für identitätsbasierte Richtlinien für AWS IoT Greengrass

IAM-Benutzer besitzen keine Berechtigungen zum Erstellen oder Ändern von AWS IoT Greengrass -Ressourcen. Sie können auch keine Aufgaben mithilfe der API AWS-Managementkonsole AWS CLI, oder AWS ausführen. Ein IAM-Administrator muss IAM-Richtlinien erstellen, die Benutzern und Rollen die Berechtigung zum Ausführen bestimmter API-Operationen für die angegebenen Ressourcen gewähren, die diese benötigen. Der Administrator muss diese Richtlinien anschließend den IAM-Benutzern oder -Gruppen anfügen, die diese Berechtigungen benötigen.

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. Beachten Sie beim Erstellen oder Bearbeiten identitätsbasierter Richtlinien die folgenden Richtlinien und Empfehlungen:

- Erste Schritte mit AWS verwalteten Richtlinien und Umstellung auf Berechtigungen mit den geringsten Rechten — Verwenden Sie die AWS verwalteten Richtlinien, die Berechtigungen für viele gängige Anwendungsfälle gewähren, um damit zu beginnen, Ihren Benutzern und Workloads Berechtigungen zu gewähren. Sie sind in Ihrem verfügbar. AWS-Konto Wir empfehlen Ihnen, die Berechtigungen weiter zu reduzieren, indem Sie vom AWS Kunden verwaltete Richtlinien definieren, die speziell auf Ihre Anwendungsfälle zugeschnitten sind. Weitere Informationen finden Sie unter AWS -verwaltete Richtlinien oder AWS -verwaltete Richtlinien für Auftrags-Funktionen im IAM-Benutzerhandbuch.
- Anwendung von Berechtigungen mit den geringsten Rechten – Wenn Sie mit IAM-Richtlinien Berechtigungen festlegen, gewähren Sie nur die Berechtigungen, die für die Durchführung einer Aufgabe erforderlich sind. Sie tun dies, indem Sie die Aktionen definieren, die für bestimmte Ressourcen unter bestimmten Bedingungen durchgeführt werden können, auch bekannt als die geringsten Berechtigungen. Weitere Informationen zur Verwendung von IAM zum Anwenden von Berechtigungen finden Sie unter Richtlinien und Berechtigungen in IAM im IAM-Benutzerhandbuch.
- Verwenden von Bedingungen in IAM-Richtlinien zur weiteren Einschränkung des Zugriffs – Sie können Ihren Richtlinien eine Bedingung hinzufügen, um den Zugriff auf Aktionen und Ressourcen zu beschränken. Sie können beispielsweise eine Richtlinienbedingung schreiben, um festzulegen, dass alle Anforderungen mithilfe von SSL gesendet werden müssen. Sie können auch Bedingungen verwenden, um Zugriff auf Serviceaktionen zu gewähren, wenn diese für einen

bestimmten Zweck verwendet werden AWS-Service, z. CloudFormation B. Weitere Informationen finden Sie unter IAM-JSON-Richtlinienelemente: Bedingung im IAM-Benutzerhandbuch.

- Verwenden von IAM Access Analyzer zur Validierung Ihrer IAM-Richtlinien, um sichere und funktionale Berechtigungen zu gewährleisten – IAM Access Analyzer validiert neue und vorhandene Richtlinien, damit die Richtlinien der IAM-Richtliniensprache (JSON) und den bewährten IAM-Methoden entsprechen. IAM Access Analyzer stellt mehr als 100 Richtlinienprüfungen und umsetzbare Empfehlungen zur Verfügung, damit Sie sichere und funktionale Richtlinien erstellen können. Weitere Informationen finden Sie unter Richtlinienvalidierung mit IAM Access Analyzer im IAM-Benutzerhandbuch.
- Multi-Faktor-Authentifizierung (MFA) erforderlich — Wenn Sie ein Szenario haben, das IAM-Benutzer oder einen Root-Benutzer in Ihrem System erfordert AWS-Konto, aktivieren Sie MFA für zusätzliche Sicherheit. Um MFA beim Aufrufen von API-Vorgängen anzufordern, fügen Sie Ihren Richtlinien MFA-Bedingungen hinzu. Weitere Informationen finden Sie unter Sicherer API-Zugriff mit MFA im IAM-Benutzerhandbuch.

Weitere Informationen zu bewährten Methoden in IAM finden Sie unter 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 umfasst Berechtigungen zum Ausführen dieser Aktion auf der Konsole oder programmgesteuert mithilfe der API oder. AWS CLI AWS

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

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 unter Autorisieren von direkten Aufrufen von AWS Diensten im AWS IoT Core Entwicklerhandbuch.

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 Rollenalias: `GreengrassV2TokenExchangeRoleAlias`

  Dieser Rollenalias bezieht sich auf die IAM-Rolle.

Weitere Informationen finden Sie unter Schritt 3: Installieren Sie die AWS IoT Greengrass Core-Software.

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 Sie mit AWS Diensten.

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

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
```

```
}
```

v2.4.x

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:DescribeCertificate",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

Earlier than v2.4.0

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:DescribeCertificate",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
```

```
        "iot:Connect",
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

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 amzn-s3-demo-bucket durch den Namen des S3-Buckets, damit das Kerngerät darauf zugreifen kann.

   JSON

   ```
   {
     "Version":"2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": [
           "s3:GetObject"
         ],
         "Resource": "arn:aws:s3:::amzn-s3-demo-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. *GreengrassV2TokenExchangeRole*Ersetzen Sie es 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. *`account-id`*Ersetzen Sie es durch Ihre AWS-Konto ID und *`GreengrassV2TokenExchangeRole`* ersetzen Sie es durch den Namen der Token-Austauschrolle, die Sie mit dem `--tes-role-name` Installer-Argument angeben.

> **Note**
>
> Die `DeployDevTools` Richtlinienanweisung ist nur erforderlich, wenn Sie das `--deploy-dev-tools` Argument für das Installationsprogramm angeben.

Greengrass nucleus v2.5.0 and later

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "CreateTokenExchangeRole",
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy",
                "iam:CreatePolicy",
                "iam:CreateRole",
                "iam:GetPolicy",
                "iam:GetRole",
                "iam:PassRole"
            ],
            "Resource": [

 "arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole",

 "arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess",
                "arn:aws:iam::aws:policy/GreengrassV2TokenExchangeRoleAccess"
            ]
        },
        {
            "Sid": "CreateIoTResources",
            "Effect": "Allow",
            "Action": [
                "iot:AddThingToThingGroup",
                "iot:AttachPolicy",
                "iot:AttachThingPrincipal",
                "iot:CreateKeysAndCertificate",
                "iot:CreatePolicy",
                "iot:CreateRoleAlias",
                "iot:CreateThing",
                "iot:CreateThingGroup",
                "iot:DescribeEndpoint",
                "iot:DescribeRoleAlias",
                "iot:DescribeThingGroup",
                "iot:GetPolicy"
            ],
```

```
            "Resource": "*"
        },
        {
            "Sid": "DeployDevTools",
            "Effect": "Allow",
            "Action": [
                "greengrass:CreateDeployment",
                "iot:CancelJob",
                "iot:CreateJob",
                "iot:DeleteThingShadow",
                "iot:DescribeJob",
                "iot:DescribeThing",
                "iot:DescribeThingGroup",
                "iot:GetThingShadow",
                "iot:UpdateJob",
                "iot:UpdateThingShadow"
            ],
            "Resource": "*"
        }
    ]
}
```

Earlier than v2.5.0

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "CreateTokenExchangeRole",
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy",
                "iam:CreatePolicy",
                "iam:CreateRole",
                "iam:GetPolicy",
                "iam:GetRole",
                "iam:PassRole"
            ],
            "Resource": [
            "arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole",
```

```
    "arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess",
                "arn:aws:iam::aws:policy/GreengrassV2TokenExchangeRoleAccess"
            ]
        },
        {
            "Sid": "CreateIoTResources",
            "Effect": "Allow",
            "Action": [
                "iot:AddThingToThingGroup",
                "iot:AttachPolicy",
                "iot:AttachThingPrincipal",
                "iot:CreateKeysAndCertificate",
                "iot:CreatePolicy",
                "iot:CreateRoleAlias",
                "iot:CreateThing",
                "iot:CreateThingGroup",
                "iot:DescribeEndpoint",
                "iot:DescribeRoleAlias",
                "iot:DescribeThingGroup",
                "iot:GetPolicy"
            ],
            "Resource": "*"
        },
        {
            "Sid": "DeployDevTools",
            "Effect": "Allow",
            "Action": [
                "greengrass:CreateDeployment",
                "iot:CancelJob",
                "iot:CreateJob",
                "iot:DeleteThingShadow",
                "iot:DescribeJob",
                "iot:DescribeThing",
                "iot:DescribeThingGroup",
                "iot:GetThingShadow",
                "iot:UpdateJob",
                "iot:UpdateThingShadow"
            ],
            "Resource": "*"
        }
    ]
}
```

Servicerolle bei Greengrass

Die Greengrass-Servicerolle ist eine AWS Identity and Access Management (IAM) -Servicerolle, die den AWS IoT Greengrass Zugriff auf Ressourcen von AWS Diensten in Ihrem Namen autorisiert. Diese Rolle ermöglicht es, die Identität von Client-Geräten AWS IoT Greengrass zu überprüfen und die Verbindungsinformationen der Kerngeräte zu verwalten.

> **Note**
>
> AWS IoT Greengrass V1 verwendet diese Rolle auch, um wichtige Aufgaben auszuführen. Weitere Informationen finden Sie unter Greengrass service role im AWS IoT Greengrass V1 Developer Guide.

Um auf Ihre Ressourcen zugreifen AWS IoT Greengrass zu können, muss die Greengrass-Servicerolle mit Ihrer verknüpft AWS-Konto und AWS IoT Greengrass als vertrauenswürdige Entität angegeben werden. Die Rolle muss die AWSGreengrassResourceAccessRolePolicyverwaltete Richtlinie oder eine benutzerdefinierte Richtlinie enthalten, die entsprechende 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 AWS Ressourcen AWS IoT Greengrass verwendet werden. Weitere Informationen finden Sie unter AWS verwaltete Richtlinie: AWSGreengrass ResourceAccessRolePolicy.

Sie können dieselbe Greengrass-Servicerolle überall wiederverwenden AWS-Regionen, müssen sie jedoch an jedem AWS-Region Ort, an dem Sie sie verwenden AWS IoT Greengrass, Ihrem Konto zuordnen. Wenn die Servicerolle in der aktuellen Version nicht konfiguriert ist AWS-Region, können die Kerngeräte die Client-Geräte nicht verifizieren und die Verbindungsinformationen nicht aktualisieren.

In den folgenden Abschnitten wird beschrieben, wie Sie die Greengrass-Servicerolle mit dem AWS-Managementkonsole oder AWS CLI erstellen und verwalten.

Themen

- Die Greengrass-Servicerolle verwalten (Konsole)
- Die Greengrass-Servicerolle (CLI) verwalten
- Weitere Informationen finden Sie auch unter

> **Note**
>
> Zusätzlich zu der Servicerolle, die den Zugriff auf Service-Ebene autorisiert, weisen Sie Greengrass-Kerngeräten eine Token-Austauschrolle zu. Die Token-Exchange-Rolle ist eine separate IAM-Rolle, die steuert, wie Greengrass-Komponenten und Lambda-Funktionen auf dem Kerngerät auf Dienste zugreifen können. AWS Weitere Informationen finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

Die Greengrass-Servicerolle verwalten (Konsole)

Die AWS IoT Konsole macht es einfach, Ihre Greengrass-Servicerolle zu verwalten. Wenn Sie beispielsweise die Client-Geräteerkennung für ein Core-Gerät konfigurieren, prüft die Konsole, ob Ihr Gerät derzeit AWS-Region mit einer Greengrass-Servicerolle verknüpft AWS-Konto ist. Andernfalls kann die Konsole eine Servicerolle für Sie erstellen und konfigurieren. Weitere Informationen finden Sie unter the section called “Erstellen der Greengrass-Servicerolle”.

Sie können die Konsole für die folgenden Rollenverwaltungsaufgaben 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 Version AWS IoT Greengrass verwendet wird AWS-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 Sie keine Servicerolle sehen, 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 Geräte-Setup-Skript 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 AWS-Region Version eine Greengrass-Servicerolle zugeordnet ist. Falls nicht, werden Sie von der Konsole aufgefordert, in Ihrem Namen Lese- und Schreibzugriff auf AWS Dienste zuzulassen AWS IoT Greengrass .

Wenn Sie die Erlaubnis erteilen, überprüft die Konsole, ob eine Rolle mit dem Namen in Ihrer `Greengrass_ServiceRole` AWS-Konto vorhanden ist.

- Wenn die Rolle vorhanden ist, ordnet die Konsole die Servicerolle Ihrer AWS-Konto in der aktuellen AWS-Region Version zu.
- Wenn die Rolle nicht existiert, erstellt die Konsole eine standardmäßige Greengrass-Servicerolle und fügt sie Ihrer aktuellen Rolle AWS-Konto hinzu. AWS-Region

> **Note**
>
> Wenn Sie eine Servicerolle mit benutzerdefinierten Rollenrichtlinien erstellen möchten, verwenden Sie die IAM-Konsole, um die Rolle zu erstellen oder zu ändern. Weitere Informationen finden Sie unter Erstellen einer Rolle zum Delegieren von Berechtigungen für einen AWS Dienst oder Ändern einer Rolle im IAM-Benutzerhandbuch. Stellen Sie sicher, dass die Rolle Berechtigungen erteilt, die der verwalteten Richtlinie `AWSGreengrassResourceAccessRolePolicy` für die verwendeten Funktionen und Ressourcen entsprechen. Wir empfehlen, dass Sie auch die Kontextschlüssel `aws:SourceArn` und die `aws:SourceAccount` globalen Bedingungsschlüssel in Ihre Vertrauensrichtlinie aufnehmen, um das Sicherheitsproblem „Confused Deputy“ zu vermeiden. Die Bedingungskontextschlüssel schränken den Zugriff so ein, dass nur Anfragen zugelassen werden, die vom angegebenen Konto und Greengrass-Workspace stammen. Weitere Informationen zum Confused-Deputy-Problem finden Sie Serviceübergreifende Confused-Deputy-Prävention.
> Wenn Sie eine Servicerolle erstellen, kehren Sie zur AWS IoT Konsole zurück und fügen Sie die Rolle Ihrer AWS-Konto hinzu. Sie können dies unter der Greengrass-Servicerolle auf der Seite Einstellungen tun.

Ändern der Greengrass-Servicerolle (Konsole)

Gehen Sie wie folgt vor, um eine andere Greengrass-Servicerolle auszuwählen, die Sie Ihrer AWS-Region aktuell AWS-Konto in der Konsole ausgewählten zuordnen möchten.

1. 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-Serverolle aktualisieren“ wird geöffnet und zeigt die IAM-Rollen in Ihrem System an AWS-Konto , die AWS IoT Greengrass als vertrauenswürdige Entität definiert sind.
4. Wählen Sie die Greengrass-Servicerolle aus, die Sie zuordnen möchten.
5. Wählen Sie Rolle anhängen.

Trennen der Greengrass-Servicerolle (Konsole)

Gehen Sie wie folgt vor, um die Greengrass-Servicerolle von Ihrem aktuellen AWS Konto zu trennen. AWS-Region Dadurch werden die Berechtigungen für den Zugriff AWS IoT Greengrass auf AWS Dienste in der aktuellen Version aufgehoben. AWS-Region

> **Important**
>
> Durch das Trennen der Servicerolle können aktive Operationen unterbrochen werden.

1. 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 ermöglichen möglicherweise AWS IoT Greengrass den Zugriff auf Ihre Ressourcen. Um alle Rollen zu finden, mit denen AWS IoT Greengrass Sie in Ihrem Namen Berechtigungen übernehmen können, suchen Sie in der IAM-Konsole auf der Seite Rollen in der Spalte Vertrauenswürdige Entitäten nach Rollen, die AWS service: greengrass enthalten.

Die Greengrass-Servicerolle (CLI) verwalten

Bei den folgenden Verfahren gehen wir davon aus, dass der für die Verwendung Ihres AWS-Konto installiert und konfiguriert AWS Command Line Interface ist. Weitere Informationen finden Sie unter Installation, Aktualisierung und Deinstallation von AWS CLI und Konfiguration von AWS CLI im AWS Command Line Interface Benutzerhandbuch.

Sie können das AWS CLI für die folgenden Rollenverwaltungsaufgaben verwenden:

Themen

- Abrufen der Greengrass-Servicerolle (CLI)
- Erstellen der Greengrass-Servicerolle (CLI)
- Entfernen der Greengrass-Servicerolle (CLI)

Abrufen der Greengrass-Servicerolle (CLI)

Verwenden Sie das folgende Verfahren, um herauszufinden, ob eine Greengrass-Servicerolle mit Ihrer AWS-Konto in einem AWS-Region verknüpft ist.

- Rufen Sie die Servicerolle ab. *region*Ersetzen Sie durch Ihre AWS-Region (zum Beispiel`us-west-2`).

  ```
  aws greengrassv2 get-service-role-for-account --region region
  ```

  Wenn Ihrem Konto bereits eine Greengrass-Servicerolle zugeordnet ist, gibt die Anfrage die folgenden Rollenmetadaten zurück.

  ```
  {
    "associatedAt": "timestamp",
    "roleArn": "arn:aws:iam::account-id:role/path/role-name"
  }
  ```

  Wenn die Anfrage keine Rollenmetadaten zurückgibt, müssen Sie die Servicerolle erstellen (falls sie nicht existiert) und sie Ihrem Konto in der AWS-Region zuordnen.

Erstellen der Greengrass-Servicerolle (CLI)

Gehen Sie wie folgt vor, um eine Rolle zu erstellen und sie Ihrer Rolle zuzuordnen AWS-Konto.

Um die Servicerolle mit IAM zu erstellen

1. Erstellen Sie eine Rolle mit einer Vertrauensrichtlinie, die es ermöglicht, die Rolle AWS IoT Greengrass zu übernehmen. In diesem Beispiel wird eine Rolle namens `Greengrass_ServiceRole` erstellt, aber Sie können einen anderen Namen verwenden. Wir empfehlen, dass Sie auch die Kontextschlüssel `aws:SourceArn` und die `aws:SourceAccount` globalen Bedingungsschlüssel in Ihre Vertrauensrichtlinie aufnehmen, um das Sicherheitsproblem „Confused Deputy“ zu vermeiden. Die Bedingungskontextschlüssel schränken den Zugriff so ein, dass nur Anfragen zugelassen werden, die vom angegebenen

Konto und Greengrass-Workspace stammen. Weitere Informationen zum Confused-Deputy-Problem finden Sie Serviceübergreifende Confused-Deputy-Prävention.

Linux 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
```

Um die Servicerolle mit Ihrem zu verknüpfen AWS-Konto

- Weisen Sie die Rolle Ihrem Konto zu. *role-arn*Ersetzen Sie durch die Servicerolle *region* ARN und 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 Anfrage die folgende Antwort zurück.

```
{
  "associatedAt": "timestamp"
}
```

Entfernen der Greengrass-Servicerolle (CLI)

Gehen Sie wie folgt vor, um die Greengrass-Servicerolle von Ihrer zu trennen. AWS-Konto

- Haben Sie die Zuordnung der Servicerolle bei Ihrem Konto auf. *region*Ersetzen Sie durch Ihre AWS-Region (zum Beispiel). `us-west-2`

  ```
  aws 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 in keiner verwenden AWS-Region. Verwenden Sie zuerst delete-role-policy, um die verwaltete `AWSGreengrassResourceAccessRolePolicy`-Richtlinie von der Rolle zu lösen, und verwenden Sie dann delete-role, um die Rolle zu löschen. Weitere Informationen zum Löschen von Rollen finden Sie unter Löschen von Rollen oder Instance-Profilen im IAM-Benutzerhandbuch.

Weitere Informationen finden Sie auch unter

- Erstellen einer Rolle zum Delegieren von Berechtigungen für einen AWS Dienst im IAM-Benutzerhandbuch
- Ändern einer Rolle im IAM-Benutzerhandbuch
- Löschen von Rollen oder Instanzprofilen im IAM-Benutzerhandbuch
- AWS IoT Greengrass Befehle in der AWS CLI Befehlsreferenz
  - associate-service-role-to-Konto
  - disassociate-service-role-from-konto
  - get-service-role-for-konto
- IAM-Befehle in der Befehlsreferenz AWS CLI

- attach-role-policy
- create-role
- delete-role
- delete-role-policy

AWS verwaltete Richtlinien für AWS IoT Greengrass

Eine AWS verwaltete Richtlinie ist eine eigenständige Richtlinie, die von erstellt und verwaltet wird AWS. AWS Verwaltete Richtlinien dienen dazu, Berechtigungen für viele gängige Anwendungsfälle bereitzustellen, sodass Sie damit beginnen können, Benutzern, Gruppen und Rollen Berechtigungen zuzuweisen.

Beachten Sie, dass AWS verwaltete Richtlinien für Ihre speziellen Anwendungsfälle möglicherweise keine Berechtigungen mit den geringsten Rechten gewähren, da sie allen AWS Kunden zur Verfügung stehen. Wir empfehlen Ihnen, die Berechtigungen weiter zu reduzieren, indem Sie vom Kunden verwaltete Richtlinien definieren, die speziell auf Ihre Anwendungsfälle zugeschnitten sind.

Sie können die in AWS verwalteten Richtlinien definierten Berechtigungen nicht ändern. Wenn die in einer AWS verwalteten Richtlinie definierten Berechtigungen AWS aktualisiert werden, wirkt sich das Update auf alle Prinzidentitäten (Benutzer, Gruppen und Rollen) aus, denen die Richtlinie zugeordnet ist. AWS aktualisiert eine AWS verwaltete Richtlinie höchstwahrscheinlich, wenn eine neue Richtlinie eingeführt AWS-Service wird oder neue API-Operationen für bestehende Dienste verfügbar werden.

Weitere Informationen finden Sie unter Von AWS verwaltete Richtlinien im IAM-Benutzerhandbuch.

Themen

- AWS verwaltete Richtlinie: AWSGreengrass FullAccess
- AWS verwaltete Richtlinie: AWSGreengrass ReadOnlyAccess
- AWS verwaltete Richtlinie: AWSGreengrass ResourceAccessRolePolicy
- AWS IoT Greengrass Aktualisierungen AWS verwalteter Richtlinien

AWS verwaltete Richtlinie: AWSGreengrass FullAccess

Sie können die `AWSGreengrassFullAccess`-Richtlinie an Ihre IAM-Identitäten anfügen.

Diese Richtlinie gewährt Administratorberechtigungen, die einem Principal vollen Zugriff auf alle AWS IoT Greengrass Aktionen gewähren.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen:

- `greengrass`— Ermöglicht Principals vollen Zugriff auf alle AWS IoT Greengrass Aktionen.

JSON

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

AWS verwaltete Richtlinie: AWSGreengrass ReadOnlyAccess

Sie können die `AWSGreengrassReadOnlyAccess`-Richtlinie an Ihre IAM-Identitäten anfügen.

Diese Richtlinie gewährt nur Leseberechtigungen, die es einem Prinzipal ermöglichen, Informationen darin anzuzeigen, aber nicht zu ändern. AWS IoT Greengrass Principals mit diesen Berechtigungen können beispielsweise die Liste der auf einem Greengrass-Core-Gerät bereitgestellten Komponenten einsehen, aber sie können keine Bereitstellung erstellen, um die Komponenten zu ändern, die auf diesem Gerät ausgeführt werden.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen:

- `greengrass`— Ermöglicht es Prinzipalen, Aktionen durchzuführen, bei denen entweder eine Liste von Elementen oder Details zu einem Element zurückgegeben wird. Dazu gehören API-Operationen, die mit `List` oder `Get` beginnen.

JSON

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

AWS verwaltete Richtlinie: AWSGreengrass ResourceAccessRolePolicy

Sie können die `AWSGreengrassResourceAccessRolePolicy` Richtlinie an Ihre IAM-Entitäten anhängen. AWS IoT Greengrass ordnet diese Richtlinie auch einer Servicerolle zu, mit der Sie Aktionen AWS IoT Greengrass in Ihrem Namen ausführen können. Weitere Informationen finden Sie unter Servicerolle bei Greengrass.

Diese Richtlinie gewährt Administratorberechtigungen, mit denen AWS IoT Greengrass Sie wichtige Aufgaben ausführen können, z. B. das Abrufen Ihrer Lambda-Funktionen, AWS IoT das Verwalten von Geräteschatten und das Überprüfen von Greengrass-Client-Geräten.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen.

- `greengrass`— Verwalte Greengrass-Ressourcen.
- `iot(*Shadow)` — AWS IoT Verwaltet Schatten, deren Namen die folgenden speziellen Identifikatoren enthalten. Diese Berechtigungen sind erforderlich, um mit Kerngeräten kommunizieren zu AWS IoT Greengrass können.
  - `*-gci`— AWS IoT Greengrass verwendet diesen Shadow, um Verbindungsinformationen zu Kerngeräten zu speichern, sodass Client-Geräte Kerngeräte erkennen und eine Verbindung zu ihnen herstellen können.

  - `*-gcm`— AWS IoT Greengrass V1 verwendet diesen Shadow, um das Kerngerät darüber zu informieren, dass das Zertifikat der Greengrass-Gruppe (CA) rotiert wurde.
  - `*-gda`— AWS IoT Greengrass V1 verwendet diesen Shadow, um das Kerngerät über eine Bereitstellung zu informieren.
  - `GG_*`— Unbenutzt.
- `iot`(`DescribeThing`und`DescribeCertificate`) — Informationen über AWS IoT Dinge und Zertifikate abrufen. Diese Berechtigungen sind erforderlich, damit Client-Geräte verifiziert werden AWS IoT Greengrass können, die eine Verbindung zu einem Core-Gerät herstellen. Weitere Informationen finden Sie unter Interagieren Sie mit lokalen IoT-Geräten.
- `lambda`— Ruft Informationen über AWS Lambda Funktionen ab. Diese Berechtigung ist erforderlich, damit Lambda-Funktionen auf Greengrass-Kernen bereitgestellt werden AWS IoT Greengrass V1 können. Weitere Informationen finden Sie im AWS IoT Greengrass V1 Entwicklerhandbuch unter Lambda-Funktion auf dem AWS IoT Greengrass Kern ausführen.
- `secretsmanager`— Ruft den Wert von AWS Secrets Manager Geheimnissen ab, deren Namen mit `greengrass-` beginnen. Diese Berechtigung ist erforderlich, damit Secrets AWS IoT Greengrass V1 Manager-Secrets auf Greengrass-Kernen bereitgestellt werden können. Weitere Informationen finden Sie unter Deploy Secrets to the AWS IoT Greengrass Core im AWS IoT Greengrass V1 Developer Guide.
- `s3`— Ruft Dateiobjekte aus S3-Buckets ab, deren Namen `greengrass` oder `sagemaker` enthalten. Diese Berechtigungen sind erforderlich, damit Sie Ressourcen für maschinelles Lernen bereitstellen AWS IoT Greengrass V1 können, die Sie in S3-Buckets speichern. Weitere Informationen finden Sie unter Ressourcen für maschinelles Lernen im AWS IoT Greengrass V1 Entwicklerhandbuch.
- `sagemaker`— Rufen Sie Informationen über Amazon SageMaker AI-Inferenzmodelle für maschinelles Lernen ab. Diese Berechtigung ist erforderlich, damit ML-Modelle auf Greengrass-Kernen bereitgestellt werden AWS IoT Greengrass V1 können. Weitere Informationen finden Sie unter Perform Machine Learning-Inferenz im AWS IoT Greengrass V1 Entwicklerhandbuch.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowGreengrassAccessToShadows",
```

```
            "Action": [
                "iot:DeleteThingShadow",
                "iot:GetThingShadow",
                "iot:UpdateThingShadow"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iot:*:*:thing/GG_*",
                "arn:aws:iot:*:*:thing/*-gcm",
                "arn:aws:iot:*:*:thing/*-gda",
                "arn:aws:iot:*:*:thing/*-gci"
            ]
        },
        {
            "Sid": "AllowGreengrassToDescribeThings",
            "Action": [
                "iot:DescribeThing"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iot:*:*:thing/*"
        },
        {
            "Sid": "AllowGreengrassToDescribeCertificates",
            "Action": [
                "iot:DescribeCertificate"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iot:*:*:cert/*"
        },
        {
            "Sid": "AllowGreengrassToCallGreengrassServices",
            "Action": [
                "greengrass:*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Sid": "AllowGreengrassToGetLambdaFunctions",
            "Action": [
                "lambda:GetFunction",
                "lambda:GetFunctionConfiguration"
            ],
            "Effect": "Allow",
```

```
            "Resource": "*"
        },
        {
            "Sid": "AllowGreengrassToGetGreengrassSecrets",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:secretsmanager:*:*:secret:greengrass-*"
        },
        {
            "Sid": "AllowGreengrassAccessToS3Objects",
            "Action": [
                "s3:GetObject"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::*Greengrass*",
                "arn:aws:s3:::*GreenGrass*",
                "arn:aws:s3:::*greengrass*",
                "arn:aws:s3:::*Sagemaker*",
                "arn:aws:s3:::*SageMaker*",
                "arn:aws:s3:::*sagemaker*"
            ]
        },
        {
            "Sid": "AllowGreengrassAccessToS3BucketLocation",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Sid": "AllowGreengrassAccessToSageMakerTrainingJobs",
            "Action": [
                "sagemaker:DescribeTrainingJob"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:sagemaker:*:*:training-job/*"
            ]
        }
    ]
```

```
}
```

AWS IoT Greengrass Aktualisierungen AWS verwalteter Richtlinien

Sie können Details zu Aktualisierungen AWS verwalteter Richtlinien AWS IoT Greengrass ab dem Zeitpunkt einsehen, zu dem dieser Dienst mit der Erfassung dieser Änderungen begann. Abonnieren Sie den RSS-Feed auf der Seite mit dem AWS IoT Greengrass V2 Dokumentenverlauf, um automatische Benachrichtigungen über Änderungen an dieser Seite zu erhalten.

Änderungen	Beschreibung	Date
AWS IoT Greengrass hat begonnen, Änderungen zu verfolgen	AWS IoT Greengrass hat begonnen, Änderungen für die AWS verwalteten Richtlinien zu verfolgen.	2. Juli 2021

Serviceübergreifende Confused-Deputy-Prävention

Das Problem des verwirrten Stellvertreters ist ein Sicherheitsproblem, bei dem eine Entität, die keine Berechtigung zur Durchführung einer Aktion hat, eine privilegiertere Entität zur Durchführung der Aktion zwingen kann. In AWS kann ein dienstübergreifendes Identitätswechsels zu einem Problem mit dem verwirrten Stellvertreter führen. Ein dienstübergreifender Identitätswechsel kann auftreten, wenn ein Dienst (der Anruf-Dienst) einen anderen Dienst anruft (den aufgerufenen Dienst). Der aufrufende Service kann manipuliert werden, um seine Berechtigungen zu verwenden, um Aktionen auf die Ressourcen eines anderen Kunden auszuführen, für die er sonst keine Zugriffsberechtigung haben sollte. Um dies zu verhindern, bietet AWS Tools, mit denen Sie Ihre Daten für alle Services mit Serviceprinzipalen schützen können, die Zugriff auf Ressourcen in Ihrem Konto erhalten haben.

Wir empfehlen, die Kontextschlüssel `aws:SourceArn`und die `aws:SourceAccount`globalen Bedingungsschlüssel in Ressourcenrichtlinien zu verwenden, um die Berechtigungen einzuschränken, die der AWS IoT Greengrass Ressource einen anderen Dienst gewähren. Wenn Sie beide globalen Bedingungskontextschlüssel verwenden, müssen der `aws:SourceAccount`-Wert und das Konto im `aws:SourceArn`-Wert dieselbe Konto-ID verwenden, wenn sie in derselben Richtlinienanweisung verwendet werden.

Der Wert von `aws:SourceArn` muss die Greengrass-Kundenressource sein, die der `sts:AssumeRole` Anfrage zugeordnet ist.

Der effektivste Weg, um sich vor dem Confused-Deputy-Problem zu schützen, ist die Verwendung des globalen Bedingungskontext-Schlüssels `aws:SourceArn` mit dem vollständigen ARN der Ressource. Wenn Sie den vollständigen ARN der Ressource nicht kennen oder wenn Sie mehrere Ressourcen angeben, verwenden Sie den globalen Bedingungskontext-Schlüssel `aws:SourceArn` mit Platzhaltern (`*`) für die unbekannten Teile des ARN. Beispiel, `arn:aws:greengrass::account-id:*`.

Ein Beispiel für eine Richtlinie, die die Kontextschlüssel `aws:SourceArn` und die `aws:SourceAccount` globalen Bedingungsschlüssel verwendet, finden Sie unterErstellen der Greengrass-Servicerolle.

Problembehandlung bei Identitäts- und Zugriffsproblemen für AWS IoT Greengrass

Verwenden Sie die folgenden Informationen, um häufig auftretende Probleme zu diagnostizieren und zu beheben, die bei der Arbeit mit AWS IoT Greengrass und IAM auftreten können.

Problembereiche

- Ich bin nicht berechtigt, eine Aktion durchzuführen in AWS IoT Greengrass
- Ich bin nicht berechtigt, IAM auszuführen: PassRole
- Ich bin Administrator und möchte anderen Zugriff gewähren AWS IoT Greengrass
- Ich möchte Personen außerhalb von mir den Zugriff auf meine Ressourcen ermöglichen AWS-KontoAWS IoT Greengrass

Hilfe zur allgemeinen Problembehandlung finden Sie unter Fehlerbehebung.

Ich bin nicht berechtigt, eine Aktion durchzuführen in AWS IoT Greengrass

Wenn Sie die Fehlermeldung erhalten, dass Sie nicht zur Ausführung einer Aktion autorisiert sind, müssen Sie sich an Ihren Administrator wenden, um Unterstützung zu erhalten. Ihr Administrator ist die Person, die Ihnen Ihren Benutzernamen und Ihr Passwort bereitgestellt hat.

Der folgende Beispielfehler tritt auf, wenn der `mateojackson` IAM-Benutzer versucht, Details zu einem Kerngerät anzuzeigen, aber nicht über die `greengrass:GetCoreDevice` entsprechenden Berechtigungen verfügt.

```
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 aufgeführt, die bei der Arbeit mit auftreten können. AWS IoT Greengrass

Ich bin nicht berechtigt, IAM auszuführen: PassRole

Wenn Sie die Fehlermeldung erhalten, dass Sie nicht zum Durchführen der `iam:PassRole`-Aktion autorisiert sind, müssen Ihre Richtlinien aktualisiert werden, um eine Rolle an AWS IoT Greengrassübergeben zu können.

Einige AWS-Services ermöglichen es Ihnen, eine bestehende Rolle an diesen Dienst zu übergeben, anstatt eine neue Servicerolle oder eine dienstverknüpfte Rolle zu erstellen. Hierzu benötigen Sie Berechtigungen für die Übergabe der Rolle an den Dienst.

Der folgende Beispielfehler tritt auf, wenn ein IAM-Benutzer mit dem Namen `marymajor` versucht, die Konsole zu verwenden, um eine Aktion in AWS IoT Greengrass auszuführen. Die Aktion erfordert jedoch, dass der Service über Berechtigungen verfügt, die durch eine Servicerolle gewährt werden. Mary besitzt keine Berechtigungen für die Übergabe der Rolle an den Dienst.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In diesem Fall müssen die Richtlinien von Mary aktualisiert werden, um die Aktion `iam:PassRole` ausführen zu können.

Wenn Sie Hilfe benötigen, wenden Sie sich an Ihren AWS Administrator. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen zur Verfügung gestellt.

Ich bin Administrator und möchte anderen Zugriff gewähren AWS IoT Greengrass

Um anderen den Zugriff zu ermöglichen AWS IoT Greengrass, müssen Sie den Personen oder Anwendungen, die Zugriff benötigen, die entsprechenden Berechtigungen erteilen. Wenn Sie Personen und Anwendungen verwalten, weisen Sie Benutzern oder Gruppen Berechtigungssätze

zu, um deren Zugriffsebene zu definieren. AWS IAM Identity Center Mit Berechtigungssätzen werden automatisch IAM-Richtlinien erstellt und den IAM-Rollen zugewiesen, die der Person oder Anwendung zugeordnet sind. Weitere Informationen finden Sie im AWS IAM Identity Center Benutzerhandbuch unter Berechtigungssätze.

Wenn Sie IAM Identity Center nicht verwenden, müssen Sie IAM-Entitäten (Benutzer oder Rollen) für die Personen oder Anwendungen erstellen, die Zugriff benötigen. Anschließend müssen Sie der Entität eine Richtlinie anfügen, die dieser die korrekten Berechtigungen in AWS IoT Greengrass gewährt. Nachdem die Berechtigungen erteilt wurden, stellen Sie dem Benutzer oder Anwendungsentwickler die Anmeldeinformationen zur Verfügung. Sie werden diese Anmeldeinformationen für den Zugriff verwenden AWS. Weitere Informationen zum Erstellen von IAM-Benutzern, -Gruppen, -Richtlinien und -Berechtigungen finden Sie im IAM-Benutzerhandbuch unter IAM-Identitäten sowie Richtlinien und Berechtigungen in IAM.

Ich möchte Personen außerhalb von mir den Zugriff auf meine Ressourcen ermöglichen AWS-KontoAWS IoT Greengrass

Sie können eine IAM-Rolle erstellen, mit der Benutzer in anderen Konten oder Personen außerhalb Ihrer Organisation auf Ihre AWS Ressourcen zugreifen können. Sie können angeben, welchen Personen vertraut werden soll, so dass diese die Rolle übernehmen können. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter AWS-Konto Gewähren des Zugriffs für einen IAM-Benutzer in einem anderen AWS-Konto , das Ihnen gehört, und Bereitstellen des Zugriffs für Drittanbieter.

AWS IoT Greengrass unterstützt keinen kontenübergreifenden Zugriff auf der Grundlage ressourcenbasierter Richtlinien oder Zugriffskontrolllisten (). ACLs

Geräteverkehr über einen Proxy oder eine Firewall zulassen

Greengrass-Kerngeräte und Greengrass-Komponenten führen ausgehende Anfragen an AWS Dienste und andere Websites durch. Aus Sicherheitsgründen können Sie den ausgehenden Datenverkehr auf eine kleine Anzahl von Endpunkten und Ports beschränken. Sie können die folgenden Informationen zu Endpunkten und Ports verwenden, um den Geräteverkehr über einen Proxy, eine Firewall oder eine Amazon VPC-Sicherheitsgruppe zu begrenzen. Weitere Informationen zur Konfiguration eines Core-Geräts für die Verwendung eines Proxys finden Sie unter. Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy

Themen

- Endpunkte für den Basisbetrieb
- Endpunkte für die Installation mit automatischer Bereitstellung
- Endpunkte für AWS von -bereitgestellte Komponenten

Endpunkte für den Basisbetrieb

Greengrass-Core-Geräte verwenden die folgenden Endpunkte und Ports für den Basisbetrieb.

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"
   }
   ```

Endpunkt	Port	Erforderlich	Beschreibung
`greengrass-ats.iot. region.amazonaws.com`	8443 oder 443	Ja	Wird für Datenebenenoperationen verwendet, z. B. für die Installation von Bereitstellungen und die Arbeit mit Client-Geräten.
`device-data-prefix -ats.iot. region.amazonaws.com`	MQTT: 8883 oder 443 HTTPS: 8443 oder 443	Ja	Wird für Datenebenenoperationen zur Gerätеverwaltung verwendet, z. B. für MQTT-Kommunikation und Shadow-Synchronisierung mit. AWS IoT Core

Endpunkt	Port	Erforderlich	Beschreibung
device-credentials-prefix .credentials.iot. *region*.amazonaws.com	443	Ja	Wird zum Abrufen von AWS Anmeldeinformationen verwendet, die das Kerngerät verwendet, um Komponentenartefakte von Amazon S3 herunterzuladen und andere Operationen 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 beinhaltet 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 auf dem Kerngerät eine Version von Greengrass Nucleus vor v2.4.0 ausgeführt wird und es für die Verwendung eines Netzwerk-Proxys konfiguriert ist. Das Kerngerät verwendet diesen Endpunkt für die MQTT-Kommunikation mit einem Proxy AWS IoT Core . Weitere Informationen

Endpunkt	Port	Erforderlich	Beschreibung
			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 Ressourcen abzurufen. AWS IoT

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 abzurufen. AWS-Konto

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 Kerngerät bereitzustellen.

Endpunkte für AWS von -bereitgestellte Komponenten

Greengrass-Core-Geräte verwenden zusätzliche Endpunkte, je nachdem, welche Softwarekomponenten sie ausführen. Die Endpunkte, die für jede AWS bereitgestellte Komponente erforderlich sind, finden Sie im Abschnitt „Anforderungen“ auf der jeweiligen Komponentenseite in diesem Entwicklerhandbuch. Weitere Informationen finden Sie unter AWS-mitgelieferte 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. Weitere Informationen zu Ihrer Verantwortung für die Einhaltung der Vorschriften bei der Nutzung AWS-Services finden Sie in der AWS Sicherheitsdokumentation.

FIPS-Endpunkte

AWS IoT Greengrass unterstützt die Verwendung von FIPS-Endpunkten (Federal Information Processing Standard (FIPS) 140-2). Wenn der FIPS-Modus aktiviert ist, sollten alle Datenübertragungen, einschließlich der HTTP- und MQTT-Protokolle, zu AWS Cloud Diensten Verbindungen mit den entsprechenden FIPS-konformen Endpunkten (FIPS — Amazon Web Services ()) aufrufen und herstellen.AWS

MQTT-Kommunikation zur AWS IoT Nutzung des FIPS-Endpunkts der IoT-Datenebene (Verbindung zu FIPS-Endpunkten - AWS IoT Core) und der von AWS IoT ihm entwickelten FIPS-konformen kryptografischen Bibliothek aws-lc. AWS

Für HTTP-Kommunikation in Greengrass:

- Für Nucleus- und Plugin-Komponenten werden alle SDK-HTTP-Clients mit FIPS-Endpunkten konfiguriert, indem die Systemeigenschaft AWS_USE_FIPS_ENDPOINT auf true gesetzt wird;
- Bei generischen Komponenten beginnen alle Komponenten damit, dass die Systemeigenschaft auf true AWS_USE_FIPS_ENDPOINT gesetzt ist. Dieser Prozess stellt sicher, dass die SDK-HTTP-Clients, die von diesen generischen Komponenten verwendet werden, Anfragen an FIPS-konforme Endpunkte senden.

> **Note**
>
> Im Fall von Stream Manager übergibt Nucleus die Umgebungsvariable _FIPS_MODE. AWS_GG Diese Umgebungsvariable ermöglicht es den im Stream Manager verwendeten HTTP-Clients, den entsprechenden FIPS-konformen Endpunkt zu identifizieren und eine Verbindung zu diesem herzustellen.

AWS IoT Greengrass bietet zwei Methoden zur Aktivierung des FIPS-Modus: Bereitstellung und Bereitstellung. Um den FIPS-Modus zu aktivieren, müssen Sie den Konfigurationsparameter `fipsMode` auf true setzen. Nucleus setzt dann die Systemeigenschaft AWS_USE_FIPS_ENDPOINT auf true und gibt sie als Umgebungsvariable an alle anderen Komponenten weiter. Lädt außerdem ein Root-CA-Zertifikat (CA3) herunter und hängt es an die bestehende RootCA.pem (oder .pem) - Datei an. AWS IoT Greengrass AmazonRoot CA1 Wenn Sie FIPS über eine neue Bereitstellung aktivieren, wird Nucleus neu gestartet, um sicherzustellen, dass die Systemeigenschaft nach der Aktivierung des FIPS-Modus wirksam wird.

Neben der Konfiguration des `fipsMode` Parameters müssen Sie auch die `iotDataEndpoint` Parameter `iotCredEndpoint` und `greengrassDataEndpoint` konfigurieren. Weitere Informationen finden Sie im entsprechenden Dokument unten.

Aktivieren Sie FIPS-Endpunkte bei der Bereitstellung

Besorgen Sie sich die AWS IoT Endgeräte für Sie und speichern Sie sie AWS-Konto, um sie später zu verwenden. Ihr Gerät verwendet diese Endpunkte, um eine Verbindung herzustellen. AWS IoT Es sind zwei Endpunkte erforderlich, der `iotDataEndpoint` und der. `iotCredEndpoint` Gehen Sie wie folgt vor:

1. Rufen Sie den FIPS-Datenendpunkt für Ihre Region in den AWS IoT Core FIPS-Datenebenen-Endpunkten ab. Der FIPS-Datenendpunkt für Sie AWS-Konto sollte wie folgt aussehen: *`data.iot-fips.us-west-2.amazonaws.com`*
2. Rufen Sie den Endpunkt der FIPS-Anmeldeinformationen für Ihre Region auf den Endpunkten der AWS IoT Core FIPS-Datenebene ab. Der Endpunkt der FIPS-Anmeldeinformationen für Sie AWS-Konto sollte wie folgt aussehen: *`data.credentials.iot-fips.us-west-2.amazonaws.com`*

Um FIPS dann bei einer Bereitstellung zu aktivieren, müssen Sie die folgende Konfiguration auf Nucleus anwenden. Die Konfiguration, die in der Bereitstellung zusammengeführt werden soll, lautet wie folgt.

Console

Konfiguration, die zusammengeführt werden soll

```
{
  "fipsMode": "true",
```

```
  "iotDataEndpoint": "data.iot-fips.us-west-2.amazonaws.com",
  "greengrassDataPlaneEndpoint": "iotData",
  "iotCredEndpoint": "data.credentials.iot-fips.us-west-2.amazonaws.com"
}
```

AWS CLI

Der folgende Befehl erstellt eine Bereitstellung auf einem Kerngerät.

```
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": {
    "aws.greengrass.Nucleus": {
      "componentVersion": "2.13.0",
      "configurationUpdate": {
        "merge":{\"fipsMode\":\"true\",\"iotDataEndpoint\":\"data.iot-fips.us-
west-2.amazonaws.com\",\"greengrassDataPlaneEndpoint\":\"iotData\",\"iotCredEndpoint
\":\"data.credentials.iot-fips.us-west-2.amazonaws.com\"}"
      }
    }
  }
}
```

Greengrass CLI

Der folgende Greengrass-CLI-Befehl erstellt eine lokale Bereitstellung auf einem Core-Gerät.

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "aws.greengrass.Nucleus=2.13.0" \
  --update-config dashboard-configuration.json
```

Die `dashboard-configuration.json` Datei enthält das folgende JSON-Dokument.

```
{
```

```
  "aws.greengrass.Nucleus": {
    "MERGE": {
       "fipsMode": "true",
       "iotDataEndpoint": "data.iot-fips.us-west-2.amazonaws.com",
       "greengrassDataPlaneEndpoint": "iotData",
       "iotCredEndpoint": "data.credentials.iot-fips.us-west-2.amazonaws.com"

    }
  }
}
```

Installieren Sie Nucleus mit FIPS-Endpunkten mit manueller Ressourcenbereitstellung

Manuelles Bereitstellen von AWS Ressourcen für AWS IoT Greengrass V2 Kerngeräte mit FIPS-Endpunkten

> **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

- Endpunkte abrufen AWS IoT
- Erstelle ein Ding AWS IoT
- Erstellen Sie das Ding-Zertifikat
- Erstellen Sie eine Token-Austausch-Rolle
- Laden Sie Zertifikate auf das Gerät herunter
- Richten Sie die Geräteumgebung ein
- Laden Sie die AWS IoT Greengrass Core-Software herunter
- Installieren Sie die Core-Software AWS IoT Greengrass

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 Es sind zwei Endpunkte erforderlich, der `iotDataEndpoint` und der. `iotCredEndpoint` Gehen Sie wie folgt vor:

1. Rufen Sie den FIPS-Datenendpunkt für Ihre Region in den AWS IoT Core FIPS-Datenebenen-Endpunkten ab. Der FIPS-Datenendpunkt für Sie AWS-Konto sollte wie folgt aussehen: *`data.iot-fips.us-west-2.amazonaws.com`*
2. Rufen Sie den Endpunkt der FIPS-Anmeldeinformationen für Ihre Region auf den Endpunkten der AWS IoT Core FIPS-Datenebene ab. Der Endpunkt der FIPS-Anmeldeinformationen für Sie AWS-Konto sollte wie folgt aussehen: *`data.credentials.iot-fips.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 AWS IoT etwas zu erschaffen

1. Erstelle AWS IoT etwas für dein Gerät. Führen Sie auf Ihrem Entwicklungscomputer den folgenden Befehl aus.

   - *`MyGreengrassCore`*Ersetzen Sie es durch den Namen des zu verwendenden Dings. 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 ähnlich wie im folgenden Beispiel aus, 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 es *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 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-----",
  "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 Objekte unterstützt IDs, 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 bei der Initialisierung des Tokens und der Generierung des privaten Schlüssels angeben. 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. `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. *`device-ip-address`*Ersetzen

Sie es durch die IP-Adresse Ihres Geräts und *~/iotdevicekey.csr* ersetzen Sie es 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 AWS IoT Ding herunterladen.

```
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.

Als Nächstes konfigurieren Sie das Ding-Zertifikat. Weitere Informationen finden Sie unter Konfigurieren Sie das Ding-Zertifikat.

Erstellen Sie eine Token-Austausch-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.

- *GreengrassV2TokenExchangeRole*Ersetzen Sie es 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.

- *GreengrassV2TokenExchangeRoleAccess*Ersetzen Sie es 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 *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 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.

- *GreengrassCoreTokenExchangeRoleAliasPolicy*Ersetzen Sie es 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.

- *GreengrassCoreTokenExchangeRoleAliasPolicy*Ersetzen Sie es durch den Namen der AWS IoT Rollenalias-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. *`device-ip-address`*Ersetzen Sie es 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.

   > **Note**
   >
   > Windows hat eine Pfadlängenbegrenzung von 260 Zeichen. Wenn Sie Windows verwenden, verwenden Sie einen Stammordner wie `C:\greengrass\v2` oder, `D:\greengrass\v2` um die Greengrass-Komponentenpfade unter der Obergrenze von 260 Zeichen zu halten.

   Linux or Unix

   - Ersetzen Sie es */greengrass/v2* durch den Ordner, den Sie verwenden möchten.

   ```
   sudo mkdir -p /greengrass/v2
   ```

   Windows Command Prompt

   - *`C:\greengrass\v2`*Ersetzen Sie durch den zu verwendenden Ordner.

   ```
   mkdir C:\greengrass\v2
   ```

   PowerShell

   - *`C:\greengrass\v2`*Ersetzen Sie durch den zu verwendenden Ordner.

```
mkdir C:\greengrass\v2
```

3. (Nur Linux) Legen Sie die Berechtigungen des übergeordneten Elements des Greengrass-Stammordners fest.

   - */greengrass*Ersetzen Sie es 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 es *C:\greengrass\v2* durch den Ordner, den Sie verwenden möchten.

   ```
   robocopy %USERPROFILE%\greengrass-v2-certs C:\greengrass\v2 /E
   ```

   PowerShell

   - *C:\greengrass\v2*Ersetzen Sie durch den zu verwendenden Ordner.

   ```
   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. Laden Sie das CA1 Zertifikat und das CA3Zertifikat herunter.

Linux or Unix

- Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Greengrass-Stammordner.

```
sudo curl -o /greengrass/v2/AmazonRootCA1.pem https://www.amazontrust.com/
repository/AmazonRootCA1.pem
sudo curl -o - https://www.amazontrust.com/repository/AmazonRootCA3.pem >> /
greengrass/v2/AmazonRootCA1.pem
```

Windows Command Prompt (CMD)

```
curl -o C:\greengrass\v2\\AmazonRootCA1.pem https://www.amazontrust.com/
repository/AmazonRootCA1.pem
```

PowerShell

```
iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile C:
\greengrass\v2\\AmazonRootCA1.pem
```

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. *device-ip-address*Ersetzen Sie es 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.

> **Note**
>
> Windows hat eine Pfadlängenbegrenzung von 260 Zeichen. Wenn Sie Windows verwenden, verwenden Sie einen Stammordner wie `C:\greengrass\v2` oder, `D:\greengrass\v2` um die Greengrass-Komponentenpfade unter der Obergrenze von 260 Zeichen zu halten.

Linux or Unix

- Ersetzen Sie es */greengrass/v2* durch den Ordner, den Sie verwenden möchten.

```
sudo mkdir -p /greengrass/v2
```

Windows Command Prompt

- *C:\greengrass\v2*Ersetzen Sie durch den zu verwendenden Ordner.

```
mkdir C:\greengrass\v2
```

PowerShell

- *C:\greengrass\v2*Ersetzen Sie durch den zu verwendenden Ordner.

```
mkdir C:\greengrass\v2
```

3. (Nur Linux) Legen Sie die Berechtigungen des übergeordneten Elements des Greengrass-Stammordners fest.

- */greengrass*Ersetzen Sie es 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. Laden Sie sowohl das als CA1 auch das CA3Zertifikat herunter.

   Linux or Unix

   - Ersetzen Sie */greengrass/v2* oder *C:\greengrass\v2* durch den Greengrass-Stammordner.

   ```
   sudo curl -o /greengrass/v2/AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   sudo curl -o - https://www.amazontrust.com/repository/AmazonRootCA3.pem >> /
   greengrass/v2/AmazonRootCA1.pem
   ```

   Windows Command Prompt (CMD)

   ```
   curl -o C:\greengrass\v2\\AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   ```

PowerShell

```
iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile C:
\greengrass\v2\\AmazonRootCA1.pem
```

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 mithilfe des `--component-default-user` Installer-Arguments erstellt. Weitere Informationen finden Sie unter Argumente des Installers.

```
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 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. *`password`*Durch ein sicheres Passwort ersetzen.

```
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. *password*Ersetzen Sie es durch das zuvor festgelegte Benutzerkennwort.

   ```
   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 es *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 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 unterArgumente des Installers.

> **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: "2.16.1"
       configuration:
         awsRegion: "us-west-2"
         iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
         fipsMode: "true"
         iotDataEndpoint: "data.iot-fips.us-west-2.amazonaws.com"
         greengrassDataPlaneEndpoint: "iotData"
         iotCredEndpoint: "data.credentials.iot-fips.us-west-2.amazonaws.com"
   ```

Führen Sie dann die folgenden Schritte aus:

- Ersetzen Sie jede Instanz von */greengrass/v2* durch den Greengrass-Stammordner.
- Ersetze es *MyGreengrassCore* durch den Namen der AWS IoT Sache.
- *2.16.1*Ersetzen Sie durch die Version der AWS IoT Greengrass Core-Software.
- *us-west-2*Ersetzen Sie durch den AWS-Region Ort, an dem Sie die Ressourcen erstellt haben.
- *GreengrassCoreTokenExchangeRoleAlias*Ersetzen Sie es durch den Namen des Alias der Token-Exchange-Rolle.
- Ersetzen Sie den *iotDataEndpoint* durch Ihren AWS IoT Datenendpunkt.
- Ersetzen Sie den *iotCredEndpoint* durch den Endpunkt Ihrer AWS IoT Anmeldeinformationen.

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 -Provider-Plugin in eine Datei mit dem Namen herunter. `aws.greengrass.crypto.Pkcs11Provider.jar` *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: "/greengrass/v2/device.pem.crt"
     privateKeyPath: "/greengrass/v2/private.pem.key"
     rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
     rootpath: "/greengrass/v2"
     thingName: "MyGreengrassCore"
   services:
     aws.greengrass.Nucleus:
       componentType: "NUCLEUS"
       version: "2.16.1"
       configuration:
         awsRegion: "us-west-2"
         iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
         fipsMode: "true"
         iotDataEndpoint: "data.iot-fips.us-west-2.amazonaws.com"
         greengrassDataPlaneEndpoint: "iotData"
   ```

```
      iotCredEndpoint: "data.credentials.iot-fips.us-west-2.amazonaws.com"
```

Führen Sie dann die folgenden Schritte aus:

- Ersetzen Sie jede Instanz von *iotdevicekey* in 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.
- Ersetze es *MyGreengrassCore* durch den Namen der AWS IoT Sache.
- *2.16.1*Ersetzen Sie durch die Version der AWS IoT Greengrass Core-Software.
- *us-west-2*Ersetzen Sie durch den AWS-Region Ort, an dem Sie die Ressourcen erstellt haben.
- *GreengrassCoreTokenExchangeRoleAlias*Ersetzen Sie es durch den Namen des Alias der Token-Exchange-Rolle.
- Ersetzen Sie den `iotDataEndpoint` durch Ihren AWS IoT Datenendpunkt.
- Ersetzen Sie den `iotCredEndpoint` durch den Endpunkt Ihrer AWS IoT Anmeldeinformationen.
- Ersetzen Sie die Konfigurationsparameter für die `aws.greengrass.crypto.Pkcs11Provider` Komponente durch die Werte für die HSM-Konfiguration auf dem Kerngerät.

4. Führen Sie das Installationsprogramm aus 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.

```
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 Führen Sie die AWS IoT Greengrass Core-Software aus.

Weitere Informationen zur Konfiguration und Verwendung der Software finden Sie unter: AWS IoT Greengrass

- Konfiguration der AWS IoT Greengrass Core-Software
- AWS IoT Greengrass Komponenten entwickeln
- AWS IoT Greengrass Komponenten auf Geräten bereitstellen
- Greengrass-Befehlszeilenschnittstelle

Installieren Sie FIPS-Endpunkte mit Flottenbereitstellung

Diese Funktion ist für Version 2.4.0 und höher der Greengrass Nucleus-Komponente verfügbar.

Installieren Sie FIPS-Endpunkte auf Ihrer AWS IoT Greengrass Core-Software mit AWS IoT Flottenbereitstellung für Ihre Kerngeräte.

> **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 AWS IoT Flottenbereitstellung für Greengrass-Kerngeräte einrichten.

> **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
- Endpunkte abrufen AWS IoT
- Laden Sie Zertifikate auf das Gerät herunter
- Richten Sie die Geräteumgebung ein
- Laden Sie die AWS IoT Greengrass Core-Software herunter
- Laden Sie das AWS IoT Fleet Provisioning Plugin herunter
- Installieren Sie die AWS IoT Greengrass Core-Software.

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 FIPS-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. Rufen Sie den FIPS-Datenendpunkt für Ihre Region in den AWS IoT Core FIPS-Datenebenen-Endpunkten ab. Der FIPS-Datenendpunkt für Sie AWS-Konto sollte wie folgt aussehen: *`data.iot-fips.us-west-2.amazonaws.com`*
2. Rufen Sie den Endpunkt der FIPS-Anmeldeinformationen für Ihre Region auf den Endpunkten der AWS IoT Core FIPS-Datenebene ab. Der Endpunkt der FIPS-Anmeldeinformationen für Sie AWS-Konto sollte wie folgt aussehen: *`data.credentials.iot-fips.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. *`device-ip-address`*Ersetzen Sie es 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.

> **Note**
>
> Windows hat eine Pfadlängenbegrenzung von 260 Zeichen. Wenn Sie Windows verwenden, verwenden Sie einen Stammordner wie `C:\greengrass\v2` oder, `D:\greengrass\v2` um die Greengrass-Komponentenpfade unter der Obergrenze von 260 Zeichen zu halten.

Linux or Unix

- Ersetzen Sie es */greengrass/v2* durch den Ordner, den Sie verwenden möchten.

```
sudo mkdir -p /greengrass/v2
```

Windows Command Prompt

- *C:\greengrass\v2*Ersetzen Sie durch den zu verwendenden Ordner.

```
mkdir C:\greengrass\v2
```

PowerShell

- *C:\greengrass\v2*Ersetzen Sie durch den zu verwendenden Ordner.

```
mkdir C:\greengrass\v2
```

3. (Nur Linux) Legen Sie die Berechtigungen des übergeordneten Elements des Greengrass-Stammordners fest.

- */greengrass*Ersetzen Sie es 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 sowohl das CA1 Zertifikat als auch das CA3Zertifikat herunter.

   Linux or Unix

   ```
   sudo curl -o - https://www.amazontrust.com/repository/AmazonRootCA3.pem >> /
   greengrass/v2/AmazonRootCA1.pem
   ```

   Windows Command Prompt (CMD)

   ```
   curl -o C:\greengrass\v2\\AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   ```

   PowerShell

   ```
   iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile C:
   \greengrass\v2\\AmazonRootCA1.pem
   ```

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 mithilfe des `--component-default-user` Installer-Arguments erstellt. Weitere Informationen finden Sie unter Argumente des Installers.

   ```
   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 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. *`password`*Durch ein sicheres Passwort ersetzen.

```
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. *`password`*Ersetzen Sie es durch das zuvor festgelegte Benutzerkennwort.

```
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 es *`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/fleetprovisioningbyclaim-latest.jar

> **Note**
>
> Sie können eine bestimmte Version des AWS IoT Fleet Provisioning-Plug-ins von der folgenden Adresse herunterladen. Ersetzen Sie es *version* durch die Version, die Sie herunterladen möchten. 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 unterArgumente des Installers.

> **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 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 Sie das AWS IoT Flottenbereitstellungs-Plugin.

   Linux or Unix

   ```
   ---
   services:
     aws.greengrass.Nucleus:
       version: "2.16.1"
       configuration:
         fipsMode: "true"
         greengrassDataPlaneEndpoint: "iotData"
     aws.greengrass.FleetProvisioningByClaim:
       configuration:
         rootPath: "/greengrass/v2"
         awsRegion: "us-west-2"
         iotDataEndpoint: "data.iot-fips.us-west-2.amazonaws.com"
         iotCredEndpoint: "data.credentials.iot-fips.us-west-2.amazonaws.com"
         iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
         provisioningTemplate: "GreengrassFleetProvisioningTemplate"
         claimCertificatePath: "/greengrass/v2/claim-certs/claim.pem.crt"
         claimCertificatePrivateKeyPath: "/greengrass/v2/claim-certs/
   claim.private.pem.key"
         rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
         templateParameters:
           ThingName: "MyGreengrassCore"
           ThingGroupName: "MyGreengrassCoreGroup"
   ```

Windows

```
---
services:
  aws.greengrass.Nucleus:
    version: "2.16.1"
  aws.greengrass.FleetProvisioningByClaim:
    configuration:
      rootPath: "C:\\greengrass\\v2"
      awsRegion: "us-west-2"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredentialEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "C:\\greengrass\\v2\\claim-certs\\claim.pem.crt"
      claimCertificatePrivateKeyPath: "C:\\greengrass\\v2\\claim-certs\
\claim.private.pem.key"
      rootCaPath: "C:\\greengrass\\v2\\AmazonRootCA1.pem"
      templateParameters:
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
```

Führen Sie dann die folgenden Schritte aus:

- *2.16.1*Ersetzen Sie es 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`
- *us-west-2*Ersetzen Sie durch die AWS Region, in der Sie die Provisioning-Vorlage und andere Ressourcen erstellt haben.
- Ersetzen Sie das `iotDataEndpoint` durch Ihren AWS IoT Datenendpunkt.
- Ersetzen Sie den `iotCredentialEndpoint` durch den Endpunkt Ihrer AWS IoT Anmeldeinformationen.

- *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`.

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. `config`, `packages` 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 Führen Sie die AWS IoT Greengrass Core-Software aus.

Weitere Informationen zur Konfiguration und Verwendung der Software finden Sie unter: AWS IoT Greengrass

- Konfiguration der AWS IoT Greengrass Core-Software
- AWS IoT Greengrass Komponenten entwickeln
- AWS IoT Greengrass Komponenten auf Geräten bereitstellen
- Greengrass-Befehlszeilenschnittstelle

Installieren Sie FIPS-Endpunkte mit auto Ressourcenbereitstellung

Die AWS IoT Greengrass Core-Software enthält ein Installationsprogramm, das Ihr Gerät als Greengrass-Core-Gerät einrichtet. Um ein Gerät schnell einzurichten, kann das Installationsprogramm das AWS IoT AWS IoT Ding, die Dinggruppe, die AWS IoT IAM-Rolle und den Rollenalias bereitstellen, die das Kerngerät für den Betrieb benötigt. Das Installationsprogramm kann auch die lokalen Entwicklungstools auf dem Kerngerät 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 keine AWS Anmeldeinformationen für das Gerät angeben können, können Sie die AWS Ressourcen bereitstellen, die das Kerngerät für den Betrieb benötigt. Sie können die Entwicklungstools auch auf einem Kerngerät bereitstellen, um es als Entwicklungsgerät zu verwenden. Auf diese Weise können Sie dem Gerät weniger Berechtigungen zuweisen, 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 Kerngerät die Anforderungen für die Installation und Ausführung der AWS IoT Greengrass Core-Software v2.0 erfüllt.

Themen

- Richten Sie die Geräteumgebung ein
- Geben Sie AWS Anmeldeinformationen für das Gerät ein
- Laden Sie die AWS IoT Greengrass Core-Software herunter
- Installieren Sie die Core-Software AWS IoT Greengrass

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 mithilfe des `--component-default-user` Installer-Arguments erstellt. Weitere Informationen finden Sie unter Argumente des Installers.

```
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 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. *`password`*Durch ein sicheres Passwort ersetzen.

   ```
   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. *password*Ersetzen Sie es durch das zuvor festgelegte Benutzerkennwort.

```
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

Geben Sie AWS Anmeldeinformationen für das Gerät ein

Geben Sie Ihre AWS Anmeldeinformationen für Ihr Gerät ein, 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.

Um AWS Anmeldeinformationen für das Gerät bereitzustellen

- Geben Sie Ihre AWS Anmeldeinformationen für das Gerät ein, damit das Installationsprogramm die AWS IoT und IAM-Ressourcen für Ihr Kerngerät bereitstellen kann. Um die Sicherheit zu erhöhen, empfehlen wir Ihnen, temporäre Anmeldeinformationen für eine IAM-Rolle zu erwerben, die nur die für die Bereitstellung erforderlichen Mindestberechtigungen gewährt. 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 die Anmeldeinformationen abzurufen und sie dem AWS IoT Greengrass Core-Software-Installationsprogramm zur Verfügung zu stellen:

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

    a. Geben Sie die Zugriffsschlüssel-ID, den geheimen Zugriffsschlüssel und das Sitzungstoken aus dem IAM Identity Center ein. Weitere Informationen finden Sie unter Manuelles Aktualisieren von Anmeldeinformationen im Abschnitt Temporäre Anmeldeinformationen abrufen und aktualisieren 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 Sicherheitsanmeldedaten aus einer IAM-Rolle:

  a. Geben Sie die Zugriffsschlüssel-ID, den geheimen Zugriffsschlüssel und das Sitzungstoken aus einer IAM-Rolle an, die Sie annehmen. Weitere Informationen zum Abrufen dieser Anmeldeinformationen finden Sie unter Temporäre Sicherheitsanmeldeinformationen anfordern 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 zur Verfügung gestellt werden soll, finden Sie unter. Minimale IAM-Richtlinie für das Installationsprogramm zur Bereitstellung von Ressourcen Weitere Informationen zum Abrufen langfristiger Anmeldeinformationen finden Sie unter Verwaltung 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 für die 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, sodass sie nicht mehr gültig sind. Weitere Informationen finden Sie im AWS Identity and Access Management Benutzerhandbuch unter Aktualisieren von Zugriffsschlüsseln.

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 es *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 Folgendes angeben:

- Erstellen Sie die AWS Ressourcen, die das Kerngerät für den Betrieb benötigt.
- 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.

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 Hinweise zu den Argumenten, die Sie angeben können, finden Sie unterArgumente des Installers.

> **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. 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.

   ```
   ---
   services:
     aws.greengrass.Nucleus:
       configuration:
         fipsMode: "true"
         iotDataEndpoint: "data.iot-fips.us-west-2.amazonaws.com"
         iotCredEndpoint: "data.credentials.iot-fips.us-west-2.amazonaws.com"
         greengrassDataPlaneEndpoint: "iotData"
   ```

   - *us-west-2*Ersetzen Sie durch den AWS-Region Ort, an dem Sie die Ressourcen erstellt haben.
   - Ersetzen Sie das *iotDataEndpoint* durch Ihren AWS IoT Datenendpunkt.

- Ersetzen Sie den *iotCredEndpoint* durch den Endpunkt Ihrer AWS IoT Anmeldeinformationen.

2. Führen Sie das AWS IoT Greengrass Core-Installationsprogramm aus. Ersetzen Sie die Argumentwerte in Ihrem Befehl wie folgt.

> **Note**
>
> Windows hat eine Pfadlängenbegrenzung von 260 Zeichen. Wenn Sie Windows verwenden, verwenden Sie einen Stammordner wie `C:\greengrass\v2` oder, `D:\greengrass\v2` um die Greengrass-Komponentenpfade unter der Obergrenze von 260 Zeichen zu halten.

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

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

   c. *region*. Das, AWS-Region in dem Ressourcen gefunden oder erstellt werden sollen.

   d. *MyGreengrassCore*. Der Name der AWS IoT Sache für Ihr Greengrass-Core-Gerät. Wenn das Ding nicht existiert, erstellt es das Installationsprogramm. Das Installationsprogramm lädt die Zertifikate herunter, um sich als das AWS IoT Ding zu authentifizieren. Weitere Informationen finden Sie unter Geräteauthentifizierung und Autorisierung für AWS IoT Greengrass.

   > **Note**
   >
   > Der Name der Sache darf keine Doppelpunkte (`:`) enthalten.

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

> **Note**
>
> Der Name der Dinggruppe darf keine Doppelpunkte (`:`) enthalten.

f. *`GreengrassV2IoTThingPolicy`*. Der Name der AWS IoT Richtlinie, die es den Greengrass-Core-Geräten ermöglicht, mit AWS IoT und AWS IoT Greengrass zu kommunizieren. Wenn die AWS IoT Richtlinie nicht existiert, erstellt das Installationsprogramm eine zulässige 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 V2 Kerngeräte.

g. *`GreengrassV2TokenExchangeRole`*. Der Name der IAM-Rolle, die es dem Greengrass-Core-Gerät ermöglicht, temporäre AWS Anmeldeinformationen abzurufen. Wenn die Rolle nicht existiert, erstellt das Installationsprogramm sie und erstellt und fügt eine Richtlinie mit dem Namen hinzu. *`GreengrassV2TokenExchangeRole`*`Access` Weitere Informationen finden Sie unter Autorisieren Sie Kerngeräte für die Interaktion mit Diensten AWS.

h. *`GreengrassCoreTokenExchangeRoleAlias`*. Der Alias für die IAM-Rolle, der es dem Greengrass-Core-Gerät ermöglicht, später temporäre Anmeldeinformationen abzurufen. Wenn der Rollenalias nicht existiert, erstellt ihn das Installationsprogramm 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 \
  --init-config ./GreengrassInstaller/config.yaml \
  --setup-system-service true
```

Windows Command Prompt (CMD)

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

PowerShell

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

Important

Auf Windows Core-Geräten müssen Sie angeben, `--setup-system-service true` dass die AWS IoT Greengrass Core-Software als Systemdienst eingerichtet werden soll.

Das Installationsprogramm gibt die folgenden Meldungen aus, wenn es erfolgreich ist:

- Wenn Sie angeben`--provision`, druckt das Installationsprogramm, `Successfully configured Nucleus with provisioned resource details` ob es die Ressourcen erfolgreich konfiguriert hat.

- Wenn Sie dies angeben`--deploy-dev-tools`, druckt das Installationsprogramm, `Configured Nucleus to deploy aws.greengrass.Cli component` ob es die Einrichtung erfolgreich erstellt hat.
- 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 Dienst eingerichtet und ausgeführt hat.
- Wenn Sie nichts angeben`--setup-system-service true`, gibt das Installationsprogramm aus, `Launched Nucleus successfully` ob es erfolgreich war und die Software ausgeführt hat.

3. 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 AWS IoT Greengrass Core-Software-Stammordner festzulegen. */greengrass/v2*Ersetzen Sie es durch den Stammordner, den Sie in Ihrem Installationsbefehl angegeben haben, und */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 Systemdienst installiert haben, führt das Installationsprogramm die Software für Sie aus. Andernfalls müssen Sie die Software manuell ausführen. Weitere Informationen finden Sie unter Führen Sie die AWS IoT Greengrass Core-Software aus.

> **Note**
>
> Standardmäßig erlaubt die IAM-Rolle, die das Installationsprogramm erstellt, 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.

Weitere Informationen zur Konfiguration und Verwendung der Software finden Sie unter: AWS IoT Greengrass

- Konfiguration der AWS IoT Greengrass Core-Software
- AWS IoT Greengrass Komponenten entwickeln
- AWS IoT Greengrass Komponenten auf Geräten bereitstellen
- Greengrass-Befehlszeilenschnittstelle

Komponenten von Erstanbietern zur FIPS-Konformität

aws.greengrass.Nucleus	data.iot-fips. *us-east-1* .amazonaws.com
	greengrass-fips. *us-east-1* .amazonaws.com
	data.credentials.iot-fips. *us-east-1* .amazonaws.com
aws.greengrass.TokenExchangeService	data.credentials.iot-fips. *us-east-1* .amazonaws.com
aws.greengrass.Cli	
aws.greengrass.StreamManager	• kinesis-fips. *us-east-1* .amazonaws.com • data.iotsitewise-fips. *us-east-1* .amazonaws.com • s3-fips.*us-east-1* .amazonaws.com

	Note Stream Manager unterstützt keinen AWS IoT Analytics FIPS-Endpunkt
`aws.greengrass.LogManager`	logs-fips. *`us-east-1`* .amazonaws.com
`aws.greengrass.crypto.Pkcs11Provider`	
`aws.greengrass.ShadowManager`	
`aws.greengrass.DockerApplicationManager`	ecr-fips. *`us-east-1`* .amazonaws.com
`aws.greengrass.SecretManager`	SecretsManager-FIPS. *`us-east-1`* .amazonaws.com
`aws.greengrass.telemetry.NucleusEmitter`	
`aws.greengrass.clientdevices.IPDetector`	
`aws.greengrass.DiskSpooler`	

Resilienz in AWS IoT Greengrass

Die AWS globale Infrastruktur basiert auf den Regionen und Availability Zones von Amazon Web Services. Jede AWS-Region bietet mehrere physisch getrennte und isolierte Availability Zones, die über Netzwerke mit niedriger Latenz, hohem Durchsatz und hoher Redundanz miteinander verbunden sind. Mithilfe von Availability Zones können Sie Anwendungen und Datenbanken erstellen und ausführen, die automatisch Failover zwischen Zonen ausführen, ohne dass es zu Unterbrechungen kommt. Availability Zones sind besser verfügbar, fehlertoleranter und skalierbarer als herkömmliche Infrastrukturen mit einem oder mehreren Rechenzentren.

Weitere Informationen finden Sie unter AWS -Globale Infrastruktur.

Zusätzlich zur AWS globalen Infrastruktur AWS IoT Greengrass bietet es mehrere Funktionen, die Sie bei der Erfüllung Ihrer Datenausfallsicherheit und Ihrer Backup-Anforderungen unterstützen.

- Sie können ein Greengrass-Core-Gerät so konfigurieren, dass es Protokolle in das lokale Dateisystem und in CloudWatch Logs schreibt. Wenn das Core-Gerät die Konnektivität verliert, kann es weiterhin Nachrichten im Dateisystem protokollieren. Wenn es wieder eine Verbindung herstellt, schreibt es die Protokollnachrichten in CloudWatch Logs. Weitere Informationen finden Sie unter AWS IoT Greengrass Protokolle überwachen.
- Wenn ein Core-Gerät während einer Bereitstellung die Stromversorgung verliert, nimmt es die Bereitstellung wieder auf, nachdem die AWS IoT Greengrass Core-Software erneut gestartet wurde.
- Wenn ein Core-Gerät die Internetverbindung verliert, können Greengrass-Client-Geräte weiterhin über das lokale Netzwerk kommunizieren.
- Sie können Greengrass-Komponenten erstellen, die Stream-Manager-Streams lesen und die Daten an lokale Speicherziele senden.

Infrastruktursicherheit in AWS IoT Greengrass

Als verwalteter Service AWS IoT Greengrass ist er durch die AWS globalen Netzwerksicherheitsverfahren geschützt, die im Whitepaper Amazon Web Services: Sicherheitsprozesse im Überblick beschrieben sind.

Sie verwenden AWS veröffentlichte API-Aufrufe, um AWS IoT Greengrass über das Netzwerk darauf 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 Systeme 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 sich mit dem zu verbinden und zu authentifizieren. AWS Cloud Weitere Informationen finden Sie unter the section called “Geräteauthentifizierung und -autorisierung”.

Konfiguration und Schwachstellenanalyse in AWS IoT Greengrass

IoT-Umgebungen können aus einer großen Anzahl von Geräten mit unterschiedlichsten Funktionen bestehen und sind langlebig und geografisch verteilt. Aufgrund dieser Merkmale ist die Geräteeinrichtung komplex und fehleranfällig. Und da Geräte bezüglich Rechenleistung, Arbeitsspeicher und Speicherkapazitäten eingeschränkt sind, können Verschlüsselung und andere Formen der Sicherheit auf den Geräten selbst nur limitiert eingesetzt werden. Außerdem verwenden Geräte häufig Software mit bekannten Schwachstellen. Diese Faktoren machen IoT-Geräte zu einem attraktiven Ziel für Hacker und erschweren die kontinuierliche Sicherung.

AWS IoT Device Defender begegnet diesen Herausforderungen durch die Bereitstellung von Tools zur Identifizierung von Sicherheitsproblemen und Abweichungen von bewährten Verfahren. Sie können AWS IoT Device Defender damit verbundene Geräte analysieren, prüfen und überwachen, um ungewöhnliches Verhalten zu erkennen und Sicherheitsrisiken zu minimieren. AWS IoT Device Defender kann Geräte überprüfen, um sicherzustellen, dass sie den bewährten Sicherheitsmethoden entsprechen, und abnormales Verhalten auf Geräten erkennen. Auf diese Weise können Sie einheitliche Sicherheitsrichtlinien für alle Ihre Geräte durchsetzen und schnell reagieren, wenn Geräte kompromittiert werden. IForWeitere Informationen finden Sie in den folgenden Themen:

- Die Device Defender-Komponente
- AWS IoT Device Defender im Entwicklerhandbuch für AWS IoT Core .

In AWS IoT Greengrass Umgebungen sollten Sie sich der folgenden Überlegungen bewusst sein:

- Es liegt in Ihrer Verantwortung, Ihre physischen Geräte, das Dateisystem auf Ihren Geräten und das lokale Netzwerk zu sichern.
- AWS IoT Greengrass erzwingt keine Netzwerkisolierung für benutzerdefinierte Greengrass-Komponenten, unabhängig davon, ob sie in einem Greengrass-Container ausgeführt werden oder nicht. Daher ist es für Greengrass-Komponenten möglich, mit jedem anderen Prozess zu kommunizieren, der im System oder außerhalb über das Netzwerk ausgeführt wird.

Codeintegrität in AWS IoT Greengrass V2

AWS IoT Greengrass stellt Softwarekomponenten von den beiden Geräten bereit AWS Cloud , auf denen die AWS IoT Greengrass Core-Software ausgeführt wird. Zu diesen Softwarekomponenten AWS gehören bereitgestellte Komponenten und benutzerdefinierte Komponenten, die Sie auf

Ihre hochladen. AWS-Konto Jede Komponente besteht aus einem Rezept. Das Rezept definiert die Metadaten der Komponente und eine beliebige Anzahl von Artefakten, bei denen es sich um Binärdateien der Komponenten handelt, wie z. B. kompilierter Code und statische Ressourcen. Komponentenartefakte werden in Amazon S3 gespeichert.

Bei der Entwicklung und Bereitstellung von Greengrass-Komponenten folgen Sie diesen grundlegenden Schritten, die mit Komponentenartefakten auf Ihren AWS-Konto und auf Ihren Geräten funktionieren:

1. Erstellen Sie Artefakte und laden Sie sie in S3-Buckets hoch.
2. Erstellen Sie eine Komponente aus einem Rezept und Artefakten im AWS IoT Greengrass Service, der für jedes Artefakt einen kryptografischen Hash berechnet.
3. Stellen Sie eine Komponente auf Greengrass-Core-Geräten bereit, die jedes Artefakt herunterladen und dessen Integrität überprüfen.

AWS ist dafür verantwortlich, die Integrität von Artefakten aufrechtzuerhalten, nachdem Sie Artefakte in S3-Buckets hochgeladen haben, auch wenn Sie Komponenten auf Greengrass-Kerngeräten bereitstellen. 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 Ihren Ressourcen zu sichern AWS-Konto, einschließlich der S3-Buckets, in die Sie Komponentenartefakte hochladen.

> **Note**
>
> Amazon S3 bietet eine Funktion namens S3 Object Lock, mit der Sie sich vor Änderungen an Komponentenartefakten in Ihren AWS-Konto S3-Buckets schützen können. Sie können S3 Object Lock verwenden, um zu verhindern, dass Komponentenartefakte gelöscht oder überschrieben werden. Weitere Informationen finden Sie unter Verwenden von S3 Object Lock im Amazon Simple Storage Service-Benutzerhandbuch.

Wenn eine öffentliche Komponente AWS veröffentlicht wird und wenn Sie eine benutzerdefinierte Komponente hochladen, AWS IoT Greengrass berechnet für jedes Komponentenartefakt einen kryptografischen Digest. AWS IoT Greengrass aktualisiert das Komponentenrezept so, dass es den Digest jedes Artefakts und den Hash-Algorithmus, der zur Berechnung dieses Digest verwendet wurde, enthält. Dieser Digest garantiert die Integrität des Artefakts, denn wenn sich das Artefakt beim AWS Cloud oder während des Herunterladens ändert, stimmt sein Datei-Digest nicht mit dem Digest

überein, der im Komponentenrezept gespeichert ist. AWS IoT Greengrass Weitere Informationen finden Sie in der Rezeptreferenz für Komponenten unter Artefakte.

Wenn Sie eine Komponente auf einem Kerngerät bereitstellen, lädt die AWS IoT Greengrass Core-Software das Komponentenrezept und jedes Komponentenartefakt herunter, das durch das Rezept definiert wird. Die AWS IoT Greengrass Core-Software berechnet den Digest jeder heruntergeladenen Artefaktdatei und vergleicht ihn mit dem Digest dieses Artefakts im Rezept. Wenn die Digests nicht übereinstimmen, schlägt die Bereitstellung fehl und die AWS IoT Greengrass Core-Software löscht die heruntergeladenen Artefakte aus dem Dateisystem des Geräts. Weitere Informationen darüber, wie Verbindungen zwischen den Kerngeräten und E gesichert AWS IoT Greengrass werden, finden Sie unter. Verschlüsselung während der Übertragung

Sie sind für die Sicherung von Komponentenartefaktdateien auf den Dateisystemen Ihrer Kerngeräte verantwortlich. Die AWS IoT Greengrass Core-Software speichert Artefakte in dem `packages` Ordner im Greengrass-Stammordner. Sie können AWS IoT Device Defender sie zur Analyse, Prüfung und Überwachung von Kerngeräten verwenden. Weitere Informationen finden Sie unter Konfiguration 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 Steuerungsebene herstellen, indem Sie einen VPC-Schnittstellen-Endpunkt erstellen. Sie können diesen Endpunkt verwenden, um Komponenten, Bereitstellungen und Kerngeräte im Service zu verwalten. AWS IoT Greengrass Schnittstellenendpunkte werden mit einer Technologie betrieben AWS PrivateLink, mit der Sie AWS IoT Greengrass APIs privat ohne Internet-Gateway, NAT-Gerät, VPN-Verbindung oder AWS Direct Connect-Verbindung zugreifen können. Instances in Ihrer VPC benötigen keine öffentlichen IP-Adressen für die Kommunikation. AWS IoT Greengrass APIs Datenverkehr zwischen Ihrer VPC und AWS IoT Greengrass verlässt das Amazon-Netzwerk nicht.

Jeder Schnittstellenendpunkt wird durch eine oder mehrere Elastic-Network-Schnittstellen in Ihren Subnetzen dargestellt.

Weitere Informationen finden Sie unter Schnittstellen-VPC-Endpunkte (AWS PrivateLink) im Amazon-VPC-Benutzerhandbuch.

Themen

- Überlegungen zu AWS IoT Greengrass VPC-Endpunkten

- Erstellen Sie einen VPC-Schnittstellen-Endpunkt für den Betrieb auf AWS IoT Greengrass Steuerungsebene
- Erstellen einer VPC-Endpunktrichtlinie für AWS IoT Greengrass
- Betreiben Sie ein AWS IoT Greengrass Kerngerät in VPC

Überlegungen zu AWS IoT Greengrass VPC-Endpunkten

Bevor Sie einen Schnittstellen-VPC-Endpunkt für einrichten AWS IoT Greengrass, lesen Sie die Eigenschaften und Einschränkungen des Schnittstellenendpunkts im Amazon VPC-Benutzerhandbuch. Beachten Sie außerdem die folgenden Überlegungen:

- AWS IoT Greengrass unterstützt Aufrufe aller API-Aktionen auf der Kontrollebene von Ihrer VPC aus. Die Steuerungsebene umfasst Operationen wie CreateDeploymentund ListEffectiveDeployments. Die Steuerungsebene umfasst keine Operationen wie ResolveComponentCandidatesund Discover, bei denen es sich um Operationen auf der Datenebene handelt.
- VPC-Endpunkte für AWS IoT Greengrass werden derzeit in AWS China Regionen nicht unterstützt.

Erstellen Sie einen VPC-Schnittstellen-Endpunkt für den Betrieb auf AWS IoT Greengrass Steuerungsebene

Sie können einen VPC-Endpunkt für die AWS IoT Greengrass Kontrollebene entweder mit der Amazon VPC-Konsole oder mit AWS Command Line Interface ()AWS CLI erstellen. Weitere Informationen finden Sie unter Erstellung eines Schnittstellenendpunkts im Benutzerhandbuch für Amazon VPC.

Erstellen Sie einen VPC-Endpunkt für die AWS IoT Greengrass Verwendung des folgenden Dienstnamens:

- com.amazonaws. *region*. grünes Gras

Wenn Sie privates DNS für den Endpunkt aktivieren, können Sie API-Anfragen an die AWS IoT Greengrass Verwendung des Standard-DNS-Namens für die Region stellen, z. B. `greengrass.us-east-1.amazonaws.com` Die Option für ein privates DNS ist standardmäßig aktiviert.

Weitere Informationen finden Sie unter Zugriff auf einen Service über einen Schnittstellenendpunkt im Benutzerhandbuch für Amazon VPC.

Erstellen einer VPC-Endpunktrichtlinie für AWS IoT Greengrass

Sie können Ihrem VPC-Endpunkt eine Endpunktrichtlinie hinzufügen, die den Zugriff auf den Betrieb der AWS IoT Greengrass Kontrollebene steuert. Die Richtlinie gibt die folgenden Informationen an:

- Prinzipal, der die Aktionen ausführen kann
- Die Aktionen, die der Prinzipal ausführen kann.
- Die Ressourcen, mit denen der Principal Aktionen ausführen kann.

Weitere Informationen finden Sie unter Steuerung des Zugriffs auf Services mit VPC-Endpunkten im Amazon-VPC-Benutzerhandbuch.

Example Beispiel: VPC-Endpunktrichtlinie für Aktionen AWS IoT Greengrass

Im Folgenden finden Sie ein Beispiel für eine Endpunktrichtlinie für AWS IoT Greengrass. Wenn diese Richtlinie an einen Endpunkt angehängt ist, gewährt sie allen Prinzipalen auf allen Ressourcen Zugriff auf die aufgelisteten AWS IoT Greengrass Aktionen.

```
{
    "Statement": [
        {
            "Principal": "*",
            "Effect": "Allow",
            "Action": [
                "greengrass:CreateDeployment",
                "greengrass:ListEffectiveDeployments"
            ],
            "Resource": "*"
        }
    ]
}
```

Betreiben Sie ein AWS IoT Greengrass Kerngerät in 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-Aliasen einrichten. Weitere Informationen zum Erstellen und Verwenden von VPC-Endpoints finden Sie unter Erstellen eines VPC-Endpoints im Amazon VPC-Benutzerhandbuch.

> **Note**
>
> Die VPC-Funktion zum automatischen Erstellen eines DNS-Eintrags ist für AWS IoT data und AWS IoT Credentials deaktiviert. Um diese Endpunkte zu verbinden, müssen Sie manuell einen privaten DNS-Eintrag 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 Regionen China und nicht unterstützt. AWS GovCloud (US) Regions
- Weitere Informationen zu den Einschränkungen von VPC-Endpunkten AWS IoT data und den VPC-Endpunkten von AWS IoT Anmeldeinformationsanbietern finden Sie unter Einschränkungen.

Richten Sie Ihr Greengrass-Core-Gerät für den Betrieb in VPC ein

1. 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:

   a. 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"
}
```

b. 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"
}
```

2. Erstellen Sie eine Amazon VPC-Schnittstelle für AWS IoT data Endpoints und AWS IoT Anmeldeinformationen:

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-Service s 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 AWS IoT Core Datenebenen-API-Endpunkt 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 Credential Provider erstellen, wählen Sie den AWS IoT Core Credential Provider-Endpunkt 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 haben`cn.com.amazonaws.region.iot.data`. 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 erstellen 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 S3-Endpunkt. Weitere Informationen finden Sie unter Erstellen eines VPC-Endpunkts für Amazon S3.
5. Wenn Sie von Greengrass AWS bereitgestellte Komponenten verwenden, sind möglicherweise zusätzliche Endpunkte und Konfigurationen erforderlich. Um die Anforderungen der Endgeräte einzusehen, wählen Sie die Komponente aus der Liste der AWS bereitgestellten Komponenten aus und schauen Sie sich den Abschnitt Anforderungen an. In den Anforderungen für die Log Manager-Komponente wird beispielsweise angegeben, 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-Nukleus-Konfiguration `greengrassDataPlaneEndpoint` muss auf eingestellt sein**`iotdata`**. Weitere Informationen finden Sie unter Greengrass Nucleus-Konfiguration.

7. Wenn Sie sich in der `us-east-1` Region befinden, setzen Sie den Konfigurationsparameter `s3EndpointType` **`REGIONAL`** in der Greengrass Nucleus-Konfiguration auf. Diese Funktion ist für Greengrass Nucleus-Versionen 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-Aliasen.

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`*`.a` `s.com`	Der private DNS-Eintrag sollte mit dem AWS IoT data Endpunkt Ihres

Service	Name des VPC-Endpunkt-Service	VPC-Endpunkttyp	Benutzerdefinierter privater DNS-Alias	Hinweise
				Kontos übereinstimmen:`aws iot describe-endpoint --endpoint-type iot:Data-ATS .`

Service	Name des VPC-Endpunkt-Service	VPC-Endpunkttyp	Benutzerdefinierter privater DNS-Alias	Hinweise
AWS IoT Erweitern Sie im angezeigten Detailbereich die Option	`com.amazonaws.` *region* `.iot.credentials`	Schnittstelle	*prefix* `.credentials.iot.us.com`	Der private DNS-Eintrag sollte 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-Eintrag wird automatisch erstellt.

Bewährte Sicherheitsmethoden für AWS IoT Greengrass

Dieses Thema enthält bewährte Sicherheitsmethoden für AWS IoT Greengrass.

Erteilen von Mindestberechtigungen

Folgen Sie dem Prinzip der geringsten Rechte für Ihre Komponenten, indem Sie sie als Benutzer ohne Zugriffsrechte 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 des * Platzhalters für die `Resource` Eigenschaften `Action` und in Ihren IAM-Richtlinien. Deklarieren Sie stattdessen, wenn möglich, eine endliche Menge von Aktionen und Ressourcen. Weitere Informationen zu den geringsten Berechtigungen und anderen bewährten Methoden für Richtlinien finden Sie unter the section called “Bewährte Methoden für Richtlinien”.

Die bewährte Methode mit den geringsten Rechten gilt auch für AWS IoT Policen, die Sie mit Ihrem Greengrass-Kern verknüpfen.

Anmeldeinformationen in Greengrass-Komponenten nicht hartcodieren

Kodieren Sie Anmeldeinformationen nicht fest in Ihren benutzerdefinierten Greengrass-Komponenten. So schützen Sie Ihre Anmeldeinformationen besser:

- Um mit AWS Diensten zu interagieren, definieren Sie Berechtigungen für bestimmte Aktionen und Ressourcen in der zentralen Gerätedienstrolle von Greengrass.
- Verwenden Sie die Secret Manager-Komponente, um Ihre Anmeldeinformationen zu speichern. Oder, wenn die Funktion das AWS SDK verwendet, verwenden Sie Anmeldeinformationen aus der standardmäßigen Anmeldeinformationsanbieterkette.

Keine Protokollierung sensibler Informationen

Sie sollten die Protokollierung von Anmeldeinformationen und anderen persönlich identifizierbaren Informationen (PII) verhindern. Es wird empfohlen, die folgenden Sicherheitsvorkehrungen zu implementieren, auch wenn für den Zugriff auf lokale Protokolle auf einem Kerngerät Root-Rechte und für den Zugriff auf CloudWatch Protokolle IAM-Berechtigungen erforderlich sind.

- Verwenden Sie keine sensiblen Informationen in MQTT-Themenpfaden.

- Verwenden Sie keine 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 in den Ressourcen IDs von Greengrass:
  - 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 in der Java-Kryptografie-Konfiguration.

Weitere Informationen finden Sie auch unter

- Bewährte Sicherheitsmethoden finden Sie AWS IoT Core im AWS IoT Entwicklerhandbuch
- Zehn goldene Sicherheitsregeln für industrielle IoT-Lösungen im Internet der Dinge im AWS offiziellen Blog

AWS IoT Device Tester Für AWS IoT Greengrass V2 verwenden

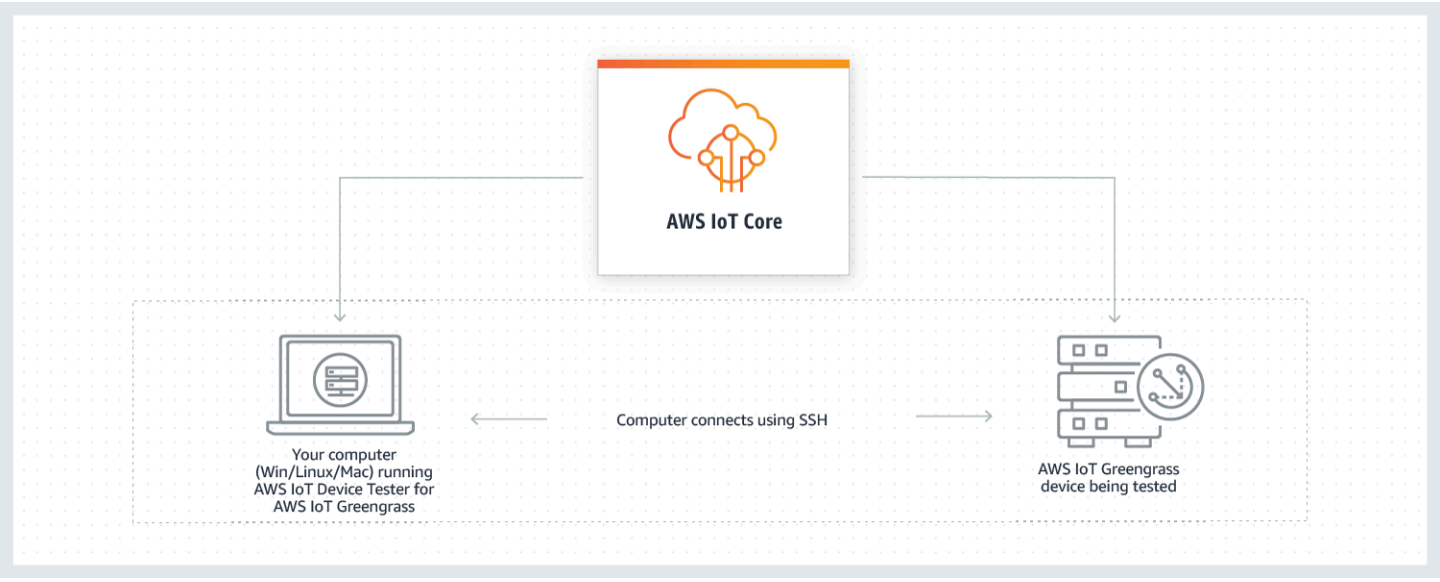
AWS IoT Device Tester (IDT) ist ein herunterladbares Test-Framework, mit dem Sie IoT-Geräte validieren können. Sie können IDT verwenden, AWS IoT Greengrass um die AWS IoT Greengrass Qualifizierungssuite auszuführen und benutzerdefinierte Testsuiten für Ihre Geräte zu erstellen und auszuführen.

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 Testsuite Q_2.5.4 für Linux GGV2
- IDT v4.9.4 mit Testsuite GGV2 Q_2.5.4 für macOS
- IDT v4.9.4 mit Testsuite Q_2.5.4 für Windows GGV2

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 Testsuite Q_2.5.3 für Linux GGV2
- IDT v4.9.3 mit Testsuite GGV2 Q_2.5.3 für macOS
- IDT v4.9.3 mit Testsuite Q_2.5.3 für Windows GGV2

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 “Unterstützungsrichtlinie 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, um ein ECR-basiertes Docker-Image zu verwenden.
- 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-Testorchestrator.
- Zusätzliche kleinere Bugfixes.

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

- Laden Sie IDT manuell herunter
- Laden Sie IDT programmgesteuert herunter

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 Testsuite Q_2.5.4 für Linux GGV2
- IDT v4.9.4 mit Testsuite GGV2 Q_2.5.4 für macOS
- IDT v4.9.4 mit Testsuite Q_2.5.4 für Windows GGV2

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 aufrufen zu können, müssen Sie über die Berechtigung zum Ausführen der **`iot-device-tester:LatestIdt`** Aktion verfügen. 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 sich auf die folgenden Beispiele beziehen, 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

- Ersetze es *`us-west-2`* durch dein AWS-Region. Eine Liste der Regionscodes finden Sie unter Regionale Endpunkte.
- *`linux`*Ersetzen Sie es durch das Betriebssystem Ihres Host-Computers.
- *`2.5.3`*Ersetzen Sie es 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.

- *`linux`*Ersetzen Sie es durch das Betriebssystem Ihres Host-Computers.

- *Timestamp*Ersetzen Sie durch Datum und Uhrzeit, z. **20220210T004606Z** B.
- *Date*Ersetzen Sie es durch das Datum, z. **20220210** B.
- Ersetze *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.
- *linux*Ersetzen Sie es 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 “Unterstützungsrichtlinie 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 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.

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. *`version`*Ersetzen Sie Ihre IDT-Version durch eine unterstützte Nucleus-Komponentenversion.

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/die Anmeldung.
2. Folgen Sie den Online-Anweisungen.

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

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

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	Beachtung der Anweisungen unter Erste Schritte im AWS IAM Identity Center - Benutzerhandbuch.	Konfigurieren Sie den programmatischen Zugriff, indem Sie AWS CLI die Konfiguration für die Verwendung AWS IAM Identity Center im AWS Command Line Interface Benutzerhandbuch vornehmen.

Wählen Sie eine Möglichkeit zur Verwaltung Ihres Administrators aus.	Bis	Von	Sie können auch
	Informationen zu 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.	Folgen Sie den Anleitungen unter IAM-Benutzer für den Notfallzugriff erstellen im IAM-Benutzerhandbuch.	Sie konfigurieren den programmgesteuerten Zugriff unter Verwendung der Informationen unter Verwalten der Zugriffsschlüssel für IAM-Benutzer im IAM-Benutzerhandbuch.

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-Managementkonsoleoder 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 Richtlinien und dann 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 das *`passRole`* und *`roleAlias`* in der `roleAliasResources` Anweisung und der `passRoleForResources` Anweisung durch die 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` Erklärung 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 IDTGreengrassIAMPermissionsRichtlinie, die Sie im vorherigen Schritt erstellt haben. Markieren Sie das Kontrollkästchen.

c. Wählen Sie Weiter: Tags aus.

d. Wählen Sie Next: Review (Weiter: Überprüfen), um eine Zusammenfassung Ihrer Auswahlmöglichkeiten anzuzeigen.

e. Wählen Sie Create user (Benutzer erstellen) aus.

f. Um die Zugriffsschlüssel des Benutzers (Zugriffsschlüssel IDs und geheime Zugriffsschlüssel) anzuzeigen, wählen Sie neben dem Passwort und dem Zugriffsschlüssel die Option Anzeigen aus. Zum Speichern der Zugriffsschlüssel wählen Sie Download .csv (CSV-Datei herunterladen) aus und speichern die Datei an einem sicheren Speicherort. Sie verwenden diese Informationen später, um Ihre AWS Anmeldeinformationsdatei zu konfigurieren.

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 das *`passRole`* und *`roleAlias`* in der `roleAliasResources` Anweisung und der `passRoleForResources` Anweisung durch die 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` Erklärung 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 Ihnen, wie Sie SSH-KEYGEN oder Pu TTYgen (für Windows) verwenden. Wenn Sie eine andere SSL-Implementierung verwenden, lesen Sie die Dokumentation zu dieser Implementierung.

IDT verwendet SSH-Schlüssel, um sich bei Ihrem zu testenden Gerät zu authentifizieren.

So erstellen Sie einen SSH-Schlüssel mit SSH-KEYGEN

1. Erstellen Sie einen SSH-Schlüssel.

   Sie können mit dem Open SSH-Befehl ssh-keygen ein SSH-Schlüsselpaar erstellen. Wenn Sie bereits ein SSH-Schlüsselpaar auf Ihrem Host-Computer haben, ist es eine bewährte Methode, ein SSH-Schlüsselpaar speziell für IDT zu erstellen. Auf diese Weise kann Ihr Host-Computer nach Abschluss des Tests keine Verbindung mehr zu Ihrem Gerät herstellen, ohne ein Passwort einzugeben. Außerdem können Sie den Zugriff auf das Remote-Gerät auf die Personen beschränken, die tatsächlich Zugriff benötigen.

   > **Note**
   >
   > Windows verfügt nicht über einen installierten SSH-Client. Weitere Informationen zur Installation eines SSH-Clients unter Windows finden Sie unter Herunterladen der SSH-Client-Software.

   Der Befehl ssh-keygen fordert Sie auf, einen Namen und Pfad zum Speichern des Schlüsselpaars einzugeben. Standardmäßig werden die Schlüsselpaardateien `id_rsa` (privater Schlüssel) und `id_rsa.pub` (öffentlicher Schlüssel) genannt. Unter macOS und Linux ist der Standard-Speicherort dieser Dateien `~/.ssh/`. Unter Windows ist der Standard-Speicherort `C:\Users\<user-name>\.ssh`.

   Wenn Sie dazu aufgefordert werden, geben Sie eine Schlüsselphrase zum Schutz Ihres SSH-Schlüssels ein. Weitere Informationen finden Sie unter Generieren eines neuen SSH-Schlüssels.

2. Fügen Sie autorisierte SSH-Schlüssel zu Ihrem zu testenden Gerät hinzu.

   IDT muss Ihren privaten SSH-Schlüssel für die Anmeldung bei Ihrem zu testenden Gerät verwenden. Um Ihren privaten SSH-Schlüssel für die Anmeldung bei Ihrem zu testenden Gerät zu autorisieren, verwenden Sie den Befehl ssh-copy-id von Ihrem Host-Computer. Dieser Befehl fügt Ihren öffentlichen Schlüssel zur Datei `~/.ssh/authorized_keys` auf dem zu testenden Gerät hinzu. Beispiel:

   $ ssh-copy-id *<remote-ssh-user>*@*<remote-device-ip>*

   Wo *remote-ssh-user* wird der Benutzername verwendet, um sich auf Ihrem zu testenden Gerät anzumelden, und *remote-device-ip* ist die IP-Adresse des zu testenden Geräts, mit dem Tests durchgeführt werden sollen. Beispiel:

ssh-copy-id pi@192.168.1.5

Wenn Sie dazu aufgefordert werden, geben Sie das Passwort für den im Befehl ssh-copy-id angegebenen Benutzernamen ein.

ssh-copy-id geht davon aus, dass der öffentliche Schlüssel `id_rsa.pub` heißt und am Standard-Speicherort gespeichert ist (`~/.ssh/` in macOS und Linux und `C:\Users\`*`<user-name>`*`\.ssh` in Windows). Wenn Sie dem öffentlichen Schlüssel einen anderen Namen gegeben oder ihn an einem anderen Ort gespeichert haben, müssen Sie den vollqualifizierten Pfad zu Ihrem öffentlichen SSH-Schlüssel mit der Option -i zu ssh-copy-id (z. B. ssh-copy-id -i ~/my/path/myKey.pub) angeben. Weitere Informationen zum Erstellen von SSH-Schlüsseln und Kopieren von öffentlichen Schlüsseln finden Sie unter SSH-COPY-ID.

Um einen SSH-Schlüssel mit Pu zu erstellen TTYgen (nur Windows)

1. Stellen Sie sicher, dass der OpenSSH-Server und -Client auf Ihrem zu testenden Gerät installiert sind. Weitere Informationen finden Sie unter OpenSSH.
2. Installieren Sie Pu TTYgen auf Ihrem zu testenden Gerät.
3. Öffnen Sie PuTTYgen.
4. Wählen Sie Generate (Generieren) aus und bewegen Sie den Mauszeiger in das Feld, um einen privaten Schlüssel zu generieren.
5. Wählen Sie im Menü Conversions (Konvertierungen) die Option Export OpenSSH key (OpenSSH-Schlüssel exportieren) aus und speichern Sie den privaten Schlüssel mit der Dateierweiterung `.pem`.
6. Fügen Sie den öffentlichen Schlüssel der Datei `/home/`*`<user>`*`/.ssh/authorized_keys` auf dem zu testenden Gerät hinzu.

   a. Kopieren Sie den Text des öffentlichen Schlüssels aus dem TTYgen Pu-Fenster.

   b. Verwenden Sie PuTTY, um eine Sitzung auf Ihrem zu testenden Gerät zu erstellen.

      i. Führen Sie in einer Eingabeaufforderung oder einem Windows Powershell-Fenster den folgenden Befehl aus:

      C:/***<path-to-putty>***/putty.exe -ssh ***<user>***@***<dut-ip-address>***

      ii. Wenn Sie dazu aufgefordert werden, geben Sie das Passwort Ihres Geräts ein.

iii. Verwenden Sie vi oder einen anderen Texteditor, um den öffentlichen Schlüssel an die Datei `/home/<user>/.ssh/authorized_keys` auf Ihrem zu testenden Gerät anzuhängen.

7. Aktualisieren Sie die Datei `device.json` mit Ihrem Benutzernamen, der IP-Adresse und dem Pfad zur Datei mit dem privaten Schlüssel, die Sie gerade auf Ihrem Host-Computer für jedes zu testende Gerät gespeichert haben. Weitere Informationen finden Sie unter the section called “Konfigurieren von device.json”. Stellen Sie sicher, dass Sie den vollständigen Pfad und Dateinamen für den privaten Schlüssel angeben und Schrägstriche ('/') verwenden. Verwenden Sie beispielsweise für den Windows-Pfad `C:\DT\privatekey.pem` die Angabe `C:/DT/privatekey.pem` in der Datei `device.json`.

Konfigurieren 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 *`user-name`* durch`ggc_user`. *`password`*Durch ein sicheres Passwort ersetzen.

   ```
   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 *user-name* durch`ggc_user`. *password*Ersetzen Sie es 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:

  JSON

  ```
  {
     "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:

  JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "iot:DescribeCertificate",
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:PutLogEvents",
            "logs:DescribeLogStreams",
            "iot:Connect",
            "iot:Publish",
            "iot:Subscribe",
            "iot:Receive",
            "iot:ListThingPrincipals",
            "iot:GetThingShadow",
            "iot:UpdateThingShadow",
            "s3:GetBucketLocation",
            "s3:GetObject",
            "s3:PutObject",
            "s3:AbortMultipartUpload",
            "s3:ListMultipartUploadParts"
         ],
         "Resource":"*"
      }
   ]
}
```

- 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 Anmeldeinformationsdatei
- 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 ist`yes`, 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 Sie die 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, mit der IDT die AWS IoT Greengrass Core-Software beim HSM authentifiziert.

> **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 von der AWS CLI aus 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. Zu diesem Zweck 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.

Führen Sie die AWS IoT Greengrass Qualifizierungssuite aus

Nachdem Sie die gewünschte Konfiguration festgelegt haben, können Sie die Tests starten. Die Laufzeit der vollständigen Testsuite hängt von Ihrer Hardware ab. Zur Referenz: Es dauert etwa 30 Minuten, die vollständige Testsuite auf einem Raspberry Pi 3B auszuführen.

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 eine Gruppe 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 Qualification Suite

Die folgenden Befehlszeilenbeispiele zeigen Ihnen, wie Sie die Qualifikationstests 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, dass Sie die vollständige Qualifizierungstestsuite ausführen, die Testgruppenabhängigkeiten in der richtigen Reihenfolge ausführt. Wenn Sie sich dafür entscheiden, bestimmte Testgruppen auszuführen, empfehlen wir, zuerst die Testgruppe für die Abhängigkeitsprüfung auszuführen, um sicherzustellen, dass alle Greengrass-Abhängigkeiten installiert sind, bevor Sie verwandte Testgruppen ausführen. Zum Beispiel:

- Führen Sie vor Ausführung von Testgruppen für die Core-Qualifizierung `coredependencies` aus.

IDT für V2-Befehle AWS IoT Greengrass

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, in diesem Fall AWS IoT Greengrass Versionen, und Testsuite-Versionen für die aktuelle IDT-Version auf.

`list-test-cases`

Listet die Testfälle in einer bestimmten Testgruppe auf. Die folgende Option wird unterstützt:

- `group-id`. Die Testgruppe, nach der gesucht werden soll. Diese Option ist erforderlich und muss eine einzelne Gruppe angeben.

`run-suite`

Führt eine Reihe von Tests in einem Pool von Geräten aus. Im Folgenden finden Sie einige unterstützte Optionen:

- `suite-id`. Die auszuführende Version der Testsuite. Wenn nicht angegeben, verwendet IDT die neueste Version im `tests`-Ordner.
- `group-id`. Die auszuführenden Testgruppen als kommagetrennte Liste. Wenn nicht angegeben, führt IDT alle entsprechenden Testgruppen in der Testsuite aus, abhängig von den konfigurierten Einstellungen in. `device.json` IDT führt aufgrund Ihrer konfigurierten Einstellungen keine Testgruppen aus, die das Gerät nicht unterstützt, auch 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 läuft. Verwenden Sie diese Option zusammen`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 auf die Aufforderung zur Aktualisierung 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 über Testartefaktpfade enthält. Diese Option ist für den `run-suite` Befehl erforderlich. Die `userdata.json` Datei muss sich im Verzeichnis *`devicetester_extract_location`* / devicetester_ggv2_ /configs/ *`[win|mac|linux]`* 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 unterProblembehandlung IDT für V2 AWS IoT Greengrass.

Anzeigen der Ergebnisse

Während der Ausführung schreibt IDT Fehler in die Konsole, Protokolldateien und Testberichte. Nachdem IDT die Qualifikations-Testsuite abgeschlossen hat, erstellt er zwei Testberichte. Diese Berichte befinden sich in*<device-tester-extract-location>*/results/*<execution-id>*/. Beide Berichte enthalten die Ergebnisse der Ausführung der Qualifizierungstestsuite.

Dies `awsiotdevicetester_report.xml` ist der Qualifizierungstestbericht, den Sie einreichen AWS , um Ihr Gerät im AWS 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 `GGV2Q_Result.xml` Bericht ist im JUnit XML-Format. 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.
- Aufschlüsselung der Testergebnisse nach den getesteten AWS IoT Greengrass Funktionen.

Interpretation der AWS 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. Zum Beispiel:

```
<testsuites name="GGQ results" time="2299" tests="28" failures="0" errors="0"
 disabled="0">
```

Im **`<testsuites>`**-Tag verwendete Attribute

`name`

Name der Testsuite

`time`

Die Zeit in Sekunden, die für die Ausführung der Qualification Suite benötigt wurde.

`tests`

Die Anzahl der Tests, die ausgeführt 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 keine Testfehler oder Fehler bei den erforderlichen Funktionen vorliegen, erfüllt Ihr Gerät die technischen Voraussetzungen für den Betrieb AWS IoT Greengrass und kann mit AWS IoT Diensten zusammenarbeiten. Wenn Sie Ihr Gerät im AWS Partner 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. 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. In jedem `<testsuite>` XML-Tag befinden sich `<testcase>` Tags für jeden Test, der für eine Testgruppe ausgeführt wurde. Zum Beispiel:

```
<testcase classname="Security Combination (IPD + DCM) Test Context" name="Security
 Combination IP Change Tests sec4_test_1: Should rotate server cert when IPD disabled
 and following changes are made:Add CIS conn info and Add another CIS conn info"
 attempts="1"></testcase>>
```

Im **`<testcase>`**-Tag verwendete Attribute

`name`

Der Name des Tests

`attempts`

Gibt an, wie oft IDT den Testfall ausgeführt hat.

Wenn ein Testfall fehlschlägt oder ein Fehler auftritt, werden `<failure>`- oder `<error>`-Tags hinzugefügt, um das `<testcase>`-Tag mit Informationen für die Fehlerbehebung zu versehen. Zum Beispiel:

```
<testcase classname="mcu.Full_MQTT" name="AFQP_MQTT_Connect_HappyCase" attempts="1">
 <failure type="Failure">Reason for the test failure</failure>
 <error>Reason for the test execution error</error>
</testcase>
```

Anzeigen von -Protokollen

IDT generiert Protokolle von Testläufen in. *<devicetester-extract-location>*/results/*<execution-id>*/logs Es werden zwei Protokollgruppen generiert:

`test_manager.log`

Von der Test Manager-Komponente von generierte Protokolle AWS IoT Device Tester (z. B. Protokolle zur Konfiguration, Testsequenzierung und Berichtserstellung).

<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 Verzeichnisses. *<devicetester-extract-location>*/results/*<execution-id>*/logs/*<test-group-id>/*

Verwenden Sie IDT, um Ihre eigenen Testsuiten zu entwickeln und auszuführen

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.

Verwenden Sie IDT, um benutzerdefinierte Testsuiten wie folgt zu entwickeln und auszuführen:

Um benutzerdefinierte Testsuiten zu entwickeln

- Erstellen Sie Testsuiten mit benutzerdefinierter Testlogik für das Greengrass-Gerät, das Sie testen möchten.
- Stellen Sie IDT Ihre benutzerdefinierten Testsuiten für Testläufer zur Verfügung. Fügen Sie Informationen zu bestimmten Einstellungskonfigurationen für Ihre Testsuiten hinzu.

Um benutzerdefinierte Testsuiten auszuführen

- Richten Sie das Gerät ein, das Sie testen möchten.
- Implementieren Sie die Einstellungskonfigurationen entsprechend den Anforderungen der Testsuiten, die Sie verwenden möchten.
- Verwenden Sie IDT, um Ihre benutzerdefinierten Testsuiten auszuführen.
- Sehen Sie sich die Testergebnisse und Ausführungsprotokolle für die von IDT ausgeführten Tests an.

Laden Sie die neueste Version von 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.

Arbeitsablauf bei der Erstellung einer Testsuite

Testsuiten bestehen aus drei Arten von Dateien:

- Konfigurationsdateien, die IDT Informationen zur Ausführung der Testsuite liefern.
- Testen Sie ausführbare Dateien, die IDT zum Ausführen von Testfällen verwendet.
- Zusätzliche Dateien, die zum Ausführen von Tests erforderlich sind.

Führen Sie die folgenden grundlegenden Schritte aus, um benutzerdefinierte IDT-Tests zu erstellen:

1. Erstellen Sie Konfigurationsdateien für Ihre Testsuite.
2. Erstellen Sie ausführbare Testfalldateien, die die Testlogik für Ihre Testsuite enthalten.
3. Überprüfen und dokumentieren Sie die Konfigurationsinformationen, die Testläufer benötigen, um die Testsuite auszuführen.
4. Stellen Sie sicher, dass IDT Ihre Testsuite ausführen und die Testergebnisse erwartungsgemäß liefern kann.

Folgen Sie den Anweisungen unter, um schnell eine benutzerdefinierte Beispielsuite zu erstellen und auszuführen. Tutorial: IDT-Beispiel-Testsuite erstellen und ausführen

Erste Schritte zum Erstellen einer benutzerdefinierten Testsuite in Python finden Sie unterTutorial: Entwickeln Sie eine einfache IDT-Testsuite.

Tutorial: IDT-Beispiel-Testsuite erstellen und ausführen

Der AWS IoT Device Tester Download enthält den Quellcode für eine Beispiel-Testsuite. Sie können dieses Tutorial abschließen, um die Beispieltestsuite zu erstellen und auszuführen, um zu verstehen, wie Sie IDT für AWS IoT Greengrass die Ausführung benutzerdefinierter Testsuiten verwenden können.

In diesem Tutorial werden Sie die folgenden Schritte ausführen:

1. Erstellen Sie die Beispiel-Testsuite
2. Verwenden Sie IDT, um die Beispieltestsuite auszuführen

Voraussetzungen

Zum Durcharbeiten dieses Tutorials ist Folgendes erforderlich:

- Anforderungen an den Host-Computer
  - Aktuelle Version von AWS IoT Device Tester

- Python 3.7 oder höher

  Führen Sie den folgenden Befehl aus, um die auf Ihrem Computer installierte Version von Python zu überprüfen:

  ```
  python3 --version
  ```

  Wenn die Verwendung dieses Befehls unter Windows einen Fehler zurückgibt, verwenden Sie `python --version` stattdessen. Wenn die zurückgegebene Versionsnummer 3.7 oder höher ist, führen Sie den folgenden Befehl in einem Powershell-Terminal aus, um ihn `python3` als Alias für Ihren `python` Befehl festzulegen.

  ```
  Set-Alias -Name "python3" -Value "python"
  ```

  Wenn keine Versionsinformationen zurückgegeben werden oder wenn die Versionsnummer kleiner als 3.7 ist, folgen Sie den Anweisungen unter Python herunterladen, um Python 3.7+ zu installieren. Weitere Informationen finden Sie in der Python-Dokumentation.

- urllib3

  Führen Sie den folgenden Befehl aus, um zu überprüfen, ob die Installation korrekt `urllib3` ist:

  ```
  python3 -c 'import urllib3'
  ```

  Wenn `urllib3` es nicht installiert ist, führen Sie den folgenden Befehl aus, um es zu installieren:

  ```
  python3 -m pip install urllib3
  ```

- Anforderungen an Speichergeräte

  - Ein Gerät mit einem Linux-Betriebssystem und einer Netzwerkverbindung zu demselben Netzwerk wie Ihr Host-Computer.

    Wir empfehlen Ihnen, einen Raspberry Pi mit Raspberry Pi OS zu verwenden. Stellen Sie sicher, dass Sie SSH auf Ihrem Raspberry Pi eingerichtet haben, um eine Remoteverbindung herzustellen.

Geräteinformationen für IDT konfigurieren

Konfigurieren Sie Ihre Geräteinformationen für IDT, um den Test auszuführen. Sie müssen die `device.json` Vorlage im *<device-tester-extract-location>*`/configs` Ordner mit den folgenden Informationen aktualisieren.

```
[
  {
    "id": "pool",
    "sku": "N/A",
    "devices": [
      {
        "id": "<device-id>",
        "connectivity": {
          "protocol": "ssh",
          "ip": "<ip-address>",
          "port": "<port>",
          "auth": {
            "method": "pki | password",
            "credentials": {
              "user": "<user-name>",
              "privKeyPath": "/path/to/private/key",
              "password": "<password>"
            }
          }
        }
      }
    ]
  }
]
```

Geben Sie im `devices` Objekt die folgenden Informationen ein:

`id`

Eine benutzerdefinierte eindeutige Kennung für Ihr Gerät.

`connectivity.ip`

Die IP-Adresse Ihres Geräts.

`connectivity.port`

Optional. Die Portnummer, die für SSH-Verbindungen zu Ihrem Gerät verwendet werden soll.

`connectivity.auth`

Authentifizierungsinformationen für die Verbindung.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.auth.method`

Die Authentifizierungsmethode, die für den Zugriff über ein bestimmtes Verbindungsprotokoll auf ein Gerät verwendet wird.

Unterstützte Werte sind:

- `pki`
- `password`

`connectivity.auth.credentials`

Die für die Authentifizierung verwendeten Anmeldeinformationen.

`connectivity.auth.credentials.user`

Der Benutzername, mit dem Sie sich auf Ihrem Gerät angemeldet haben.

`connectivity.auth.credentials.privKeyPath`

Der vollständige Pfad zu dem privaten Schlüssel, mit dem Sie sich auf Ihrem Gerät angemeldet haben.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `pki` festgelegt ist.

`devices.connectivity.auth.credentials.password`

Das Passwort, mit dem Sie sich auf Ihrem Gerät angemeldet haben.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `password` festgelegt ist.

> **Note**
>
> Geben Sie `privKeyPath` nur an, wenn `method` auf `pki` gesetzt ist.
> Geben Sie `password` nur an, wenn `method` auf `password` gesetzt ist.

Erstellen Sie die Beispiel-Testsuite

Der *<device-tester-extract-location>*`/samples/python` Ordner enthält Beispielkonfigurationsdateien, Quellcode und das IDT Client SDK, die Sie mithilfe der bereitgestellten Build-Skripten zu einer Testsuite kombinieren können. Die folgende Verzeichnisstruktur zeigt den Speicherort dieser Beispieldateien:

```
<device-tester-extract-location>
### ...
### tests
### samples
#   ### ...
#   ### python
#       ### configuration
#       ### src
#       ### build-scripts
#           ### build.sh
#           ### build.ps1
### sdks
    ### ...
    ### python
        ### idt_client
```

Um die Testsuite zu erstellen, führen Sie die folgenden Befehle auf Ihrem Host-Computer aus:

Windows

```
cd <device-tester-extract-location>/samples/python/build-scripts
./build.ps1
```

Linux, macOS, or UNIX

```
cd <device-tester-extract-location>/samples/python/build-scripts
./build.sh
```

Dadurch wird die Beispiel-Testsuite in dem `IDTSampleSuitePython_1.0.0` Ordner innerhalb des *<device-tester-extract-location>*`/tests` Ordners erstellt. Sehen Sie sich die Dateien im `IDTSampleSuitePython_1.0.0` Ordner an, um zu verstehen, wie die Beispieltestsuite strukturiert ist, und um verschiedene Beispiele für ausführbare Testfälle und JSON-Dateien für Testkonfigurationen zu sehen.

> **Note**
>
> Die Beispieltestsuite enthält Python-Quellcode. Nehmen Sie keine vertraulichen Informationen in Ihren Testsuite-Code auf.

Nächster Schritt: Verwenden Sie IDT, um die von Ihnen erstellte Beispiel-Testsuite auszuführen.

Verwenden Sie IDT, um die Beispieltestsuite auszuführen

Führen Sie die folgenden Befehle auf Ihrem Host-Computer aus, um die Beispiel-Testsuite auszuführen:

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id IDTSampleSuitePython
```

IDT führt die Beispieltestsuite aus und streamt die Ergebnisse an die Konsole. Wenn der Test abgeschlossen ist, sehen Sie die folgenden Informationen:

```
========== Test Summary ==========
Execution Time:         5s
Tests Completed:        4
Tests Passed:           4
Tests Failed:           0
Tests Skipped:          0
----------------------------------
Test Groups:
    sample_group:       PASSED
----------------------------------
Path to IoT Device Tester Report: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/logs
Path to Aggregated JUnit Report: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/IDTSampleSuitePython_Report.xml
```

Fehlerbehebung

Verwenden Sie die folgenden Informationen, um Probleme beim Abschließen des Tutorials zu lösen.

Der Testfall wird nicht erfolgreich ausgeführt

Wenn der Test nicht erfolgreich ausgeführt wird, streamt IDT die Fehlerprotokolle an die Konsole, die Ihnen bei der Problembehandlung des Testlaufs helfen kann. Stellen Sie sicher, dass Sie alle Voraussetzungen für dieses Tutorial erfüllen.

Es kann keine Verbindung zu dem zu testenden Gerät hergestellt werden

Überprüfen Sie Folgendes:

- Ihre `device.json` Datei enthält die richtige IP-Adresse, den richtigen Port und die richtigen Authentifizierungsinformationen.
- Sie können von Ihrem Host-Computer aus eine Verbindung zu Ihrem Gerät über SSH herstellen.

Tutorial: Entwickeln Sie eine einfache IDT-Testsuite

Eine Testsuite kombiniert Folgendes:

- Testdateien, die die Testlogik enthalten
- Konfigurationsdateien, die die Testsuite beschreiben

Dieses Tutorial zeigt Ihnen, wie Sie IDT for verwenden AWS IoT Greengrass , um eine Python-Testsuite zu entwickeln, die einen einzigen Testfall enthält. In diesem Tutorial werden Sie die folgenden Schritte ausführen:

1. Erstellen Sie ein Testsuite-Verzeichnis
2. Erstellen Sie Konfigurationsdateien
3. Erstellen Sie die ausführbare Testfalldatei
4. Führen Sie die Testsuite aus

Voraussetzungen

Zum Durcharbeiten dieses Tutorials ist Folgendes erforderlich:

- Anforderungen an den Host-Computer
  - Aktuelle Version von AWS IoT Device Tester
  - Python 3.7 oder höher

Führen Sie den folgenden Befehl aus, um die auf Ihrem Computer installierte Version von Python zu überprüfen:

```
python3 --version
```

Wenn die Verwendung dieses Befehls unter Windows einen Fehler zurückgibt, verwenden Sie `python --version` stattdessen. Wenn die zurückgegebene Versionsnummer 3.7 oder höher ist, führen Sie den folgenden Befehl in einem Powershell-Terminal aus, um ihn `python3` als Alias für Ihren `python` Befehl festzulegen.

```
Set-Alias -Name "python3" -Value "python"
```

Wenn keine Versionsinformationen zurückgegeben werden oder wenn die Versionsnummer kleiner als 3.7 ist, folgen Sie den Anweisungen unter Python herunterladen, um Python 3.7+ zu installieren. Weitere Informationen finden Sie in der Python-Dokumentation.

- urllib3

  Führen Sie den folgenden Befehl aus, um zu überprüfen, ob die Installation korrekt `urllib3` ist:

  ```
  python3 -c 'import urllib3'
  ```

  Wenn `urllib3` es nicht installiert ist, führen Sie den folgenden Befehl aus, um es zu installieren:

  ```
  python3 -m pip install urllib3
  ```

- Anforderungen an Speichergeräte
  - Ein Gerät mit einem Linux-Betriebssystem und einer Netzwerkverbindung zu demselben Netzwerk wie Ihr Host-Computer.

    Wir empfehlen Ihnen, einen Raspberry Pi mit Raspberry Pi OS zu verwenden. Stellen Sie sicher, dass Sie SSH auf Ihrem Raspberry Pi eingerichtet haben, um eine Remoteverbindung herzustellen.

Erstellen Sie ein Testsuite-Verzeichnis

IDT unterteilt Testfälle innerhalb jeder Testsuite logisch in Testgruppen. Jeder Testfall muss sich innerhalb einer Testgruppe befinden. Erstellen Sie für dieses Tutorial einen Ordner mit dem Namen `MyTestSuite_1.0.0` und erstellen Sie in diesem Ordner den folgenden Verzeichnisbaum:

```
MyTestSuite_1.0.0
### suite
    ### myTestGroup
        ### myTestCase
```

Erstellen Sie Konfigurationsdateien

Ihre Testsuite muss die folgenden erforderlichen Konfigurationsdateien enthalten:

Erforderliche Konfigurationsdateien

`suite.json`

Enthält Informationen über die Testsuite. Siehe Konfigurieren Sie suite.json.

`group.json`

Enthält Informationen über eine Testgruppe. Sie müssen für jede Testgruppe in Ihrer Testsuite eine `group.json` Datei erstellen. Siehe Konfigurieren Sie group.json.

`test.json`

Enthält Informationen zu einem Testfall. Sie müssen für jeden Testfall in Ihrer Testsuite eine `test.json` Datei erstellen. Siehe Konfigurieren Sie test.json.

1. Erstellen Sie in dem `MyTestSuite_1.0.0/suite` Ordner eine `suite.json` Datei mit der folgenden Struktur:

   ```
   {
       "id": "MyTestSuite_1.0.0",
       "title": "My Test Suite",
       "details": "This is my test suite.",
       "userDataRequired": false
   }
   ```

2. Erstellen Sie in dem `MyTestSuite_1.0.0/myTestGroup` Ordner eine `group.json` Datei mit der folgenden Struktur:

```
{
    "id": "MyTestGroup",
    "title": "My Test Group",
    "details": "This is my test group.",
    "optional": false
}
```

3. Erstellen Sie in dem `MyTestSuite_1.0.0/myTestGroup/myTestCase` Ordner eine `test.json` Datei mit der folgenden Struktur:

```
{
    "id": "MyTestCase",
    "title": "My Test Case",
    "details": "This is my test case.",
    "execution": {
        "timeout": 300000,
        "linux": {
            "cmd": "python3",
            "args": [
                "myTestCase.py"
            ]
        },
        "mac": {
            "cmd": "python3",
            "args": [
                "myTestCase.py"
            ]
        },
        "win": {
            "cmd": "python3",
            "args": [
                "myTestCase.py"
            ]
        }
    }
}
```

Der Verzeichnisbaum für Ihren `MyTestSuite_1.0.0` Ordner sollte jetzt wie folgt aussehen:

```
MyTestSuite_1.0.0
### suite
```

```
    ### suite.json
    ### myTestGroup
        ### group.json
        ### myTestCase
            ### test.json
```

Holen Sie sich das IDT-Client-SDK

Sie verwenden das IDT-Client-SDK, damit IDT mit dem zu testenden Gerät interagieren und Testergebnisse melden kann. Für dieses Tutorial verwenden Sie die Python-Version des SDK.

Kopieren Sie den *<device-tester-extract-location>*`/sdks/python/` Ordner aus dem `idt_client` Ordner in Ihren `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase` Ordner.

Führen Sie den folgenden Befehl aus, um zu überprüfen, ob das SDK erfolgreich kopiert wurde.

```
cd MyTestSuite_1.0.0/suite/myTestGroup/myTestCase
python3 -c 'import idt_client'
```

Erstellen Sie die ausführbare Testfalldatei

Die ausführbaren Testfalldateien enthalten die Testlogik, die Sie ausführen möchten. Eine Testsuite kann mehrere ausführbare Testfalldateien enthalten. Für dieses Tutorial erstellen Sie nur eine ausführbare Testfalldatei.

1. Erstellen Sie die Testsuite-Datei.

   Erstellen Sie in dem `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase` Ordner eine `myTestCase.py` Datei mit dem folgenden Inhalt:

   ```
   from idt_client import *

   def main():
       # Use the client SDK to communicate with IDT
       client = Client()

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

2. Verwenden Sie die Funktionen des Client-SDK, um Ihrer `myTestCase.py` Datei die folgende Testlogik hinzuzufügen:

a. Führen Sie auf dem zu testenden Gerät einen SSH-Befehl aus.

```
from idt_client import *

def main():
    # Use the client SDK to communicate with IDT
    client = Client()

    # Create an execute on device request
    exec_req = ExecuteOnDeviceRequest(ExecuteOnDeviceCommand("echo 'hello
 world'"))

    # Run the command
    exec_resp = client.execute_on_device(exec_req)

    # Print the standard output
    print(exec_resp.stdout)

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

b. Senden Sie das Testergebnis an IDT.

```
from idt_client import *

def main():
    # Use the client SDK to communicate with IDT
    client = Client()

    # Create an execute on device request
    exec_req = ExecuteOnDeviceRequest(ExecuteOnDeviceCommand("echo 'hello
 world'"))

    # Run the command
    exec_resp = client.execute_on_device(exec_req)

    # Print the standard output
    print(exec_resp.stdout)

    # Create a send result request
    sr_req = SendResultRequest(TestResult(passed=True))
```

```
    # Send the result
    client.send_result(sr_req)

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

Geräteinformationen für IDT konfigurieren

Konfigurieren Sie Ihre Geräteinformationen für IDT, um den Test auszuführen. Sie müssen die `device.json` Vorlage im *<device-tester-extract-location>*`/configs` Ordner mit den folgenden Informationen aktualisieren.

```
[
  {
    "id": "pool",
    "sku": "N/A",
    "devices": [
      {
        "id": "<device-id>",
        "connectivity": {
          "protocol": "ssh",
          "ip": "<ip-address>",
          "port": "<port>",
          "auth": {
            "method": "pki | password",
            "credentials": {
              "user": "<user-name>",
              "privKeyPath": "/path/to/private/key",
              "password": "<password>"
            }
          }
        }
      }
    ]
  }
]
```

Geben Sie im `devices` Objekt die folgenden Informationen ein:

`id`

Eine benutzerdefinierte eindeutige Kennung für Ihr Gerät.

`connectivity.ip`

Die IP-Adresse Ihres Geräts.

`connectivity.port`

Optional. Die Portnummer, die für SSH-Verbindungen zu Ihrem Gerät verwendet werden soll.

`connectivity.auth`

Authentifizierungsinformationen für die Verbindung.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.auth.method`

Die Authentifizierungsmethode, die für den Zugriff über ein bestimmtes Verbindungsprotokoll auf ein Gerät verwendet wird.

Unterstützte Werte sind:

- `pki`
- `password`

`connectivity.auth.credentials`

Die für die Authentifizierung verwendeten Anmeldeinformationen.

`connectivity.auth.credentials.user`

Der Benutzername, mit dem Sie sich auf Ihrem Gerät angemeldet haben.

`connectivity.auth.credentials.privKeyPath`

Der vollständige Pfad zu dem privaten Schlüssel, mit dem Sie sich auf Ihrem Gerät angemeldet haben.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `pki` festgelegt ist.

`devices.connectivity.auth.credentials.password`

Das Passwort, mit dem Sie sich auf Ihrem Gerät angemeldet haben.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `password` festgelegt ist.

> **Note**
>
> Geben Sie `privKeyPath` nur an, wenn `method` auf `pki` gesetzt ist.

Geben Sie `password` nur an, wenn `method` auf `password` gesetzt ist.

Führen Sie die Testsuite aus

Nachdem Sie Ihre Testsuite erstellt haben, möchten Sie sicherstellen, dass sie wie erwartet funktioniert. Führen Sie dazu die folgenden Schritte aus, um die Testsuite mit Ihrem vorhandenen Gerätepool auszuführen.

1. Kopieren Sie Ihren `MyTestSuite_1.0.0` Ordner in*<device-tester-extract-location>*/`tests`.
2. Führen Sie die folgenden Befehle aus:

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id MyTestSuite
```

IDT führt Ihre Testsuite aus und streamt die Ergebnisse an die Konsole. Wenn der Test abgeschlossen ist, sehen Sie die folgenden Informationen:

```
time="2020-10-19T09:24:47-07:00" level=info msg=Using pool: pool
time="2020-10-19T09:24:47-07:00" level=info msg=Using test suite "MyTestSuite_1.0.0"
 for execution
time="2020-10-19T09:24:47-07:00" level=info msg=b'hello world\n'
 suiteId=MyTestSuite groupId=myTestGroup testCaseId=myTestCase deviceId=my-device
 executionId=9a52f362-1227-11eb-86c9-8c8590419f30
time="2020-10-19T09:24:47-07:00" level=info msg=All tests finished.
 executionId=9a52f362-1227-11eb-86c9-8c8590419f30
time="2020-10-19T09:24:48-07:00" level=info msg=

========== Test Summary ==========
Execution Time:         1s
Tests Completed:        1
Tests Passed:           1
Tests Failed:           0
Tests Skipped:          0
----------------------------------
Test Groups:
    myTestGroup:        PASSED
----------------------------------
```

```
Path to IoT Device Tester Report: /path/to/devicetester/
results/9a52f362-1227-11eb-86c9-8c8590419f30/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/devicetester/
results/9a52f362-1227-11eb-86c9-8c8590419f30/logs
Path to Aggregated JUnit Report: /path/to/devicetester/
results/9a52f362-1227-11eb-86c9-8c8590419f30/MyTestSuite_Report.xml
```

Fehlerbehebung

Verwenden Sie die folgenden Informationen, um Probleme beim Abschließen des Tutorials zu lösen.

Der Testfall wird nicht erfolgreich ausgeführt

Wenn der Test nicht erfolgreich ausgeführt wird, streamt IDT die Fehlerprotokolle an die Konsole, die Ihnen bei der Problembehandlung des Testlaufs helfen kann. Bevor Sie die Fehlerprotokolle überprüfen, überprüfen Sie Folgendes:

- Das IDT-Client-SDK befindet sich im richtigen Ordner, wie in diesem Schritt beschrieben.
- Sie erfüllen alle Voraussetzungen für dieses Tutorial.

Es kann keine Verbindung zu dem zu testenden Gerät hergestellt werden

Überprüfen Sie Folgendes:

- Ihre `device.json` Datei enthält die richtige IP-Adresse, den richtigen Port und die richtigen Authentifizierungsinformationen.
- Sie können von Ihrem Host-Computer aus eine Verbindung zu Ihrem Gerät über SSH herstellen.

Erstellen Sie Konfigurationsdateien für die IDT-Testsuite

In diesem Abschnitt werden die Formate beschrieben, in denen Sie Konfigurationsdateien erstellen, die Sie beim Schreiben einer benutzerdefinierten Testsuite einbeziehen.

Erforderliche Konfigurationsdateien

`suite.json`

Enthält Informationen über die Testsuite. Siehe Konfigurieren Sie suite.json.

`group.json`

Enthält Informationen über eine Testgruppe. Sie müssen für jede Testgruppe in Ihrer Testsuite eine `group.json` Datei erstellen. Siehe Konfigurieren Sie group.json.

`test.json`

Enthält Informationen zu einem Testfall. Sie müssen für jeden Testfall in Ihrer Testsuite eine `test.json` Datei erstellen. Siehe Konfigurieren Sie test.json.

Optionale Konfigurationsdateien

`test_orchestrator.yaml` oder `state_machine.json`

Definiert, wie Tests ausgeführt werden, wenn IDT die Testsuite ausführt. SSe Konfigurieren Sie 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 Datei. `state_machine.json` Informationen zur Zustandsmaschine finden Sie unterKonfigurieren Sie die IDT-Zustandsmaschine.

`userdata_schema.json`

Definiert das Schema für die `userdata.json`Datei, die Testläufer in ihre Einstellungskonfiguration aufnehmen können. Die `userdata.json` Datei wird für alle zusätzlichen Konfigurationsinformationen verwendet, die für die Ausführung des Tests erforderlich sind, aber nicht in der `device.json` Datei enthalten sind. Siehe Konfigurieren Sie userdata_schema.json.

Die 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
```

Konfigurieren Sie suite.json

Die `suite.json` Datei legt Umgebungsvariablen fest und bestimmt, ob Benutzerdaten für die Ausführung der Testsuite erforderlich sind. Verwenden Sie die folgende Vorlage, um Ihre *<custom-test-suite-folder>*`/suite/suite.json` Datei zu konfigurieren:

```
{
    "id": "<suite-name>_<suite-version>",
    "title": "<suite-title>",
    "details": "<suite-details>",
    "userDataRequired": true | false,
    "environmentVariables": [
        {
            "key": "<name>",
            "value": "<value>",
        },
        ...
        {
            "key": "<name>",
            "value": "<value>",
        }
    ]
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`id`

Eine eindeutige benutzerdefinierte ID für die Testsuite. Der Wert von `id` muss mit dem Namen des Testsuite-Ordners übereinstimmen, in dem sich die `suite.json` Datei befindet. Der Name der Suite und die Suite-Version müssen außerdem die folgenden Anforderungen erfüllen:

- *<suite-name>*darf keine Unterstriche enthalten.
- *<suite-version>*wird bezeichnet als*x.x.x*, wo x ist eine Zahl.

Die ID wird in IDT-generierten Testberichten angezeigt.

`title`

Ein benutzerdefinierter Name für das Produkt oder die Funktion, das von dieser Testsuite getestet wird. Der Name wird in der IDT-CLI für Testläufer angezeigt.

`details`

Eine kurze Beschreibung des Zwecks der Testsuite.

`userDataRequired`

Definiert, ob Testläufer benutzerdefinierte Informationen in eine `userdata.json` Datei aufnehmen müssen. Wenn Sie diesen Wert auf setzen`true`, müssen Sie die `userdata_schema.json`Datei auch in Ihren Testsuite-Ordner aufnehmen.

`environmentVariables`

Optional. Ein Array von Umgebungsvariablen, die für diese Testsuite festgelegt werden sollen.

`environmentVariables.key`

Der Name der Umgebungsvariable.

`environmentVariables.value`

Der Wert der Umgebungsvariable.

Konfigurieren Sie group.json

Die `group.json` Datei definiert, ob eine Testgruppe erforderlich oder optional ist. Verwenden Sie die folgende Vorlage, um Ihre *<custom-test-suite-folder>*`/suite/`*<test-group>*`/group.json` Datei zu konfigurieren:

```
{
    "id": "<group-id>",
    "title": "<group-title>",
    "details": "<group-details>",
    "optional": true | false,
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`id`

Eine eindeutige benutzerdefinierte ID für die Testgruppe. Der Wert von `id` muss mit dem Namen des Testgruppenordners übereinstimmen, in dem sich die `group.json` Datei befindet, und darf keine Unterstriche () _ enthalten. Die ID wird in IDT-generierten Testberichten verwendet.

`title`

Ein beschreibender Name für die Testgruppe. Der Name wird in der IDT-CLI für Testläufer angezeigt.

`details`

Eine kurze Beschreibung des Zwecks der Testgruppe.

`optional`

Optional. Stellen Sie diese Option ein`true`, um diese Testgruppe als optionale Gruppe anzuzeigen, nachdem IDT die Ausführung der erforderlichen Tests abgeschlossen hat. Der Standardwert ist `false`.

Konfigurieren Sie test.json

Die `test.json` Datei bestimmt die ausführbaren Testfalldateien und die Umgebungsvariablen, die von einem Testfall verwendet werden. Weitere Hinweise zum Erstellen von ausführbaren Testfalldateien finden Sie unter. Ausführbare IDT-Testfalldateien erstellen

Verwenden Sie die folgende Vorlage, um Ihre *<custom-test-suite-folder>*/suite/*<test-group>*/*<test-case>*/test.json Datei zu konfigurieren:

```
{
    "id": "<test-id>",
    "title": "<test-title>",
    "details": "<test-details>",
    "requireDUT": true | false,
    "requiredResources": [
        {
            "name": "<resource-name>",
            "features": [
                {
                    "name": "<feature-name>",
                    "version": "<feature-version>",
                    "jobSlots": <job-slots>
```

```
                }
            ]
        }
    ],
    "execution": {
        "timeout": <timeout>,
        "mac": {
            "cmd": "/path/to/executable",
            "args": [
                "<argument>"
            ],
        },
        "linux": {
            "cmd": "/path/to/executable",
            "args": [
                "<argument>"
            ],
        },
        "win": {
            "cmd": "/path/to/executable",
            "args": [
                "<argument>"
            ]
        }
    },
    "environmentVariables": [
        {
            "key": "<name>",
            "value": "<value>",
        }
    ]
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`id`

Eine eindeutige benutzerdefinierte ID für den Testfall. Der Wert von `id` muss mit dem Namen des Testfallordners übereinstimmen, in dem sich die `test.json` Datei befindet, und darf keine Unterstriche () _ enthalten. Die ID wird in IDT-generierten Testberichten verwendet.

`title`

Ein beschreibender Name für den Testfall. Der Name wird in der IDT-CLI für Testläufer angezeigt.

`details`

Eine kurze Beschreibung des Zwecks des Testfalls.

`requireDUT`

Optional. `true`Ist auf eingestellt, wenn ein Gerät für die Ausführung dieses Tests erforderlich ist, andernfalls auf`false`. Der Standardwert ist `true`. Testläufer konfigurieren die Geräte, mit denen sie den Test ausführen, in ihrer `device.json` Datei.

`requiredResources`

Optional. Ein Array, das Informationen über Ressourcengeräte bereitstellt, die für die Ausführung dieses Tests benötigt werden.

`requiredResources.name`

Der eindeutige Name, der dem Ressourcengerät gegeben werden soll, wenn dieser Test ausgeführt wird.

`requiredResources.features`

Eine Reihe von benutzerdefinierten Funktionen für Ressourcengeräte.

`requiredResources.features.name`

Der Name der Funktion. Die Gerätefunktion, für die Sie dieses Gerät verwenden möchten. Dieser Name wird mit dem Funktionsnamen abgeglichen, den der Testrunner in der `resource.json` Datei angegeben hat.

`requiredResources.features.version`

Optional. Die Version der Funktion. Dieser Wert wird mit der vom Test-Runner in der `resource.json` Datei bereitgestellten Feature-Version abgeglichen. Wenn keine Version bereitgestellt wird, wird die Funktion nicht überprüft. Wenn für die Funktion keine Versionsnummer erforderlich ist, lassen Sie dieses Feld leer.

`requiredResources.features.jobSlots`

Optional. Die Anzahl der gleichzeitigen Tests, die diese Funktion unterstützen kann. Der Standardwert ist 1. Wenn Sie möchten, dass IDT unterschiedliche Geräte für einzelne Funktionen verwendet, empfehlen wir Ihnen, diesen Wert auf 1 festzulegen.

`execution.timeout`

Die Zeit (in Millisekunden), die IDT wartet, bis der Test abgeschlossen ist. Weitere Informationen zum Einstellen dieses Werts finden Sie unter. Ausführbare IDT-Testfalldateien erstellen

`execution.os`

Die ausführbaren Testfalldateien, die auf dem Betriebssystem des Host-Computers ausgeführt werden sollen, basieren auf dem Betriebssystem, auf dem IDT ausgeführt wird. Unterstützte Werte sind `linux`, `mac` und `win`.

`execution.os.cmd`

Der Pfad zu der ausführbaren Testfalldatei, die Sie für das angegebene Betriebssystem ausführen möchten. Dieser Speicherort muss sich im Systempfad befinden.

`execution.os.args`

Optional. Die Argumente, die zur Ausführung der ausführbaren Testfalldatei angegeben werden müssen.

`environmentVariables`

Optional. Ein Array von Umgebungsvariablen, die für diesen Testfall festgelegt wurden.

`environmentVariables.key`

Der Name der Umgebungsvariable.

`environmentVariables.value`

Der Wert der Umgebungsvariable.

> **Note**
>
> Wenn Sie dieselbe Umgebungsvariable in der `test.json` Datei und in der `suite.json` Datei angeben, hat der Wert in der `test.json` Datei Vorrang.

Konfigurieren Sie test_orchestrator.yaml

Ein Testorchestrator ist ein Konstrukt, das den Ausführungsablauf der Testsuite steuert. Er bestimmt den Startstatus einer Testsuite, verwaltet Zustandsübergänge auf der Grundlage benutzerdefinierter Regeln und setzt den Übergang durch diese Zustände fort, bis der Endstatus erreicht ist.

Wenn Ihre Testsuite keinen benutzerdefinierten Test-Orchestrator enthält, generiert IDT einen Test-Orchestrator für Sie.

Der Standard-Test-Orchestrator führt die folgenden Funktionen aus:

- Bietet Testläufern die Möglichkeit, anstelle der gesamten Testsuite bestimmte Testgruppen auszuwählen und auszuführen.
- Wenn keine bestimmten Testgruppen ausgewählt sind, wird jede Testgruppe in der Testsuite in zufälliger Reihenfolge ausgeführt.
- Generiert Berichte und druckt eine Konsolenübersicht aus, in der die Testergebnisse für jede Testgruppe und jeden Testfall angezeigt werden.

Weitere Informationen zur Funktionsweise des IDT Test Orchestrator finden Sie unter. Konfigurieren Sie den IDT-Testorchestrator

Konfigurieren Sie userdata_schema.json

Die `userdata_schema.json` Datei bestimmt das Schema, in dem Testläufer Benutzerdaten bereitstellen. Benutzerdaten sind erforderlich, wenn Ihre Testsuite Informationen benötigt, die nicht in der `device.json` Datei enthalten sind. Beispielsweise benötigen Ihre Tests möglicherweise Anmeldeinformationen für Wi-Fi-Netzwerke, bestimmte offene Ports oder Zertifikate, die ein Benutzer bereitstellen muss. Diese Informationen können IDT als Eingabeparameter zur Verfügung gestellt werden`userdata`, dessen Wert eine `userdata.json` Datei ist, die Benutzer in ihrem *`<device-tester-extract-location>`*`/config` Ordner erstellen. Das Format der `userdata.json` Datei basiert auf der `userdata_schema.json` Datei, die Sie in die Testsuite aufnehmen.

Um anzugeben, dass Testläufer eine `userdata.json` Datei bereitstellen müssen:

1. Stellen Sie in der `suite.json` Datei `userDataRequired` auf ein`true`.
2. Erstellen Sie in Ihrem *`<custom-test-suite-folder>`* eine `userdata_schema.json` Datei.
3. Bearbeiten Sie die `userdata_schema.json` Datei, um ein gültiges IETF-Draft v4-JSON-Schema zu erstellen.

Wenn IDT Ihre Testsuite ausführt, liest es automatisch das Schema und verwendet es, um die vom Testläufer bereitgestellte `userdata.json` Datei zu validieren. Falls gültig, ist der Inhalt der `userdata.json` Datei sowohl im IDT-Kontext als auch im Test-Orchestrator-Kontext verfügbar.

Konfigurieren Sie den IDT-Testorchestrator

Ab IDT v4.5.1 enthält IDT eine neue Test Orchestrator-Komponente. 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, auf der Grundlage benutzerdefinierter Regeln.

Wenn Ihre Testsuite keinen benutzerdefinierten Test-Orchestrator enthält, generiert IDT einen Test-Orchestrator für Sie.

Der Standard-Test-Orchestrator führt die folgenden Funktionen aus:

- Bietet Testläufern die Möglichkeit, anstelle der gesamten Testsuite bestimmte Testgruppen auszuwählen und auszuführen.
- Wenn keine bestimmten Testgruppen ausgewählt sind, wird jede Testgruppe in der Testsuite in zufälliger Reihenfolge ausgeführt.
- Generiert Berichte und druckt eine Konsolenübersicht aus, in der die Testergebnisse für jede Testgruppe und jeden Testfall angezeigt werden.

Der Test-Orchestrator ersetzt den IDT-Test-Orchestrator. Wir empfehlen dringend, anstelle des IDT-Testorchestrators den Testorchestrator für die Entwicklung Ihrer Testsuiten zu verwenden. Der Test-Orchestrator bietet die folgenden verbesserten Funktionen:

- Verwendet ein deklaratives Format im Vergleich zum imperativen Format, das die IDT-Zustandsmaschine verwendet. Auf diese Weise können Sie angeben, welche Tests Sie ausführen möchten und wann Sie sie ausführen möchten.
- Verwaltet die Bearbeitung bestimmter Gruppen, die Generierung von Berichten, die Fehlerbehandlung und die Ergebnisverfolgung, sodass Sie diese Aktionen nicht manuell verwalten müssen.
- Verwendet das YAML-Format, das Kommentare standardmäßig unterstützt.
- Benötigt 80 Prozent weniger Festplattenspeicher 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 zirkulären Abhängigkeiten enthält.

Testen Sie das Orchestrator-Format

Sie können die folgende Vorlage verwenden, um Ihre eigene *<custom-test-suite-folder>*`/suite/test_orchestrator.yaml` Datei zu konfigurieren:

```
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 Zeichenketten, die Kontextausdrücken zugeordnet sind. Aliase ermöglichen es Ihnen, benutzerfreundliche Namen zu generieren, um Kontextausdrücke in Ihrer Test-Orchestrator-Konfiguration zu identifizieren. 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 Greifen Sie auf Daten im Kontext zu.

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 der entsprechenden Testfälle, die ausgeführt werden, wenn jede Bedingung erfüllt ist. Jede Bedingung kann mehrere Testfälle haben. Sie können einen bestimmten Testfall jedoch 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.

Jedes Element in der `ConditionalTests` Liste enthält die folgenden Parameter:

`Condition`

Ein Kontextausdruck, der einen booleschen Wert ergibt. Wenn der ausgewertete Wert wahr ist, führt IDT die im Parameter angegebenen Testfälle aus. `Tests`

`Tests`

Die Liste der Testdeskriptoren.

Jeder Testdeskriptor verwendet die Testgruppen-ID und einen oder mehrere Testfälle, um die einzelnen Tests IDs 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 `Aliases` definieren können.

```
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` dies zutrifft, führt IDT die Tests in den Testgruppen A, B und C durch.
- Wenn `Condition2` dies zutrifft, führt IDT die Tests in den Testgruppen C und D durch.

`Order`

Optional. Die Reihenfolge, in der die Tests ausgeführt werden. Sie geben die Testreihenfolge auf der Ebene der Testgruppen an. Wenn Sie diesen Abschnitt nicht angeben, führt IDT alle zutreffenden Testgruppen in zufälliger Reihenfolge aus. Der Wert von `Order` ist eine Liste von Gruppendeskriptorlisten. Jede Testgruppe, in der Sie nicht aufgeführt sind`Order`, kann parallel zu jeder anderen aufgelisteten Testgruppe ausgeführt werden.

Jede Gruppendeskriptorliste enthält einen oder mehrere Gruppendeskriptoren und gibt die Reihenfolge an, in der die Gruppen ausgeführt werden sollen, die in den einzelnen Deskriptoren 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.
- "*"— Platzhalter. Dies entspricht der Liste aller Testgruppen, die noch nicht in der aktuellen Gruppendeskriptorliste angegeben sind.

Der Wert für `Order` muss außerdem die folgenden Anforderungen erfüllen:

- Die Testgruppe IDs , die Sie in einem Gruppendeskriptor angeben, muss in Ihrer Testsuite vorhanden sein.
- Jede Gruppendeskriptorliste muss mindestens eine Testgruppe enthalten.
- Jede Gruppendeskriptorliste muss eine eindeutige Gruppe enthalten. IDs Sie können eine Testgruppen-ID nicht innerhalb einzelner Gruppendeskriptoren wiederholen.
- Eine Gruppendeskriptorliste kann höchstens einen Platzhalter-Gruppendeskriptor enthalten. Der Platzhalter-Gruppendeskriptor muss das erste oder das 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 die Testgruppe A, dann die Testgruppe B 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 der `awsiotdevicetester_report.xml` Datei hinzufügen soll. Wenn Sie diesen Abschnitt nicht angeben, fügt IDT dem Bericht keine Produktfunktionen hinzu.

Bei einer Produktfunktion handelt es sich um benutzerdefinierte Informationen über bestimmte Kriterien, die ein Gerät möglicherweise erfüllt. Beispielsweise kann die MQTT-Produktfunktion angeben, dass das Gerät MQTT-Nachrichten ordnungsgemäß veröffentlicht. In werden Produktfunktionen als`awsiotdevicetester_report.xml`, oder als benutzerdefinierter benutzerdefinierter Wert festgelegt `supportednot-supported`, je nachdem, ob die angegebenen Tests bestanden wurden.

Jedes Element in der `Features` Liste besteht aus den folgenden Parametern:

`Name`

Der Name der Funktion.
  ```

`Value`

Optional. Der benutzerdefinierte Wert, den Sie anstelle von im Bericht verwenden möchten`supported`. Wenn dieser Wert nicht angegeben ist, legt Based IDT den Feature-Wert auf `supported` oder `not-supported` basierend auf Testergebnissen fest. Wenn Sie dasselbe Feature mit unterschiedlichen Bedingungen testen, können Sie für jede Instanz dieses Features in der `Features` Liste einen benutzerdefinierten Wert verwenden, und IDT verkettet die Feature-Werte für unterstützte Bedingungen. Weitere Informationen finden Sie unter

`Condition`

Ein Kontextausdruck, der zu einem booleschen Wert ausgewertet wird. Wenn der ausgewertete Wert wahr ist, fügt IDT die Funktion dem Testbericht hinzu, nachdem die Ausführung der Testsuite abgeschlossen ist. Wenn der ausgewertete Wert falsch ist, ist der Test nicht im Bericht enthalten.

`Tests`

Optional. Die Liste der Testdeskriptoren. Alle Tests, die in dieser Liste aufgeführt sind, müssen bestanden werden, damit die Funktion unterstützt wird.

Jeder Testdeskriptor in dieser Liste verwendet die Testgruppen-ID und einen oder mehrere Testfälle, um die einzelnen Tests IDs 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 `OneOfTests` für jedes Feature in der `Features` Liste entweder `Tests` oder angeben.

`OneOfTests`

Optional. Die Liste der Testdeskriptoren. Mindestens einer der in dieser Liste aufgeführten Tests muss bestanden werden, damit die Funktion unterstützt wird.

Jeder Testdeskriptor in dieser Liste verwendet die Testgruppen-ID und einen oder mehrere Testfälle, um die einzelnen Tests IDs 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 `OneOfTests` für jedes Feature in der `Features` Liste entweder `Tests` oder angeben.

`IsRequired`

Der boolesche Wert, der definiert, ob die Funktion im Testbericht erforderlich ist. Der Standardwert ist `false`.

Example

Testen Sie den Orchestrator-Kontext

Der Test-Orchestrator-Kontext ist ein schreibgeschütztes JSON-Dokument, das Daten enthält, die dem Test-Orchestrator während der Ausführung zur Verfügung stehen. Der Test-Orchestrator-Kontext ist nur vom Test-Orchestrator aus zugänglich und enthält Informationen, die den Testablauf bestimmen. Sie können beispielsweise Informationen verwenden, die von Testläufern in der `userdata.json` Datei konfiguriert wurden, um festzustellen, ob ein bestimmter Test ausgeführt werden muss.

Der Test-Orchestrator-Kontext 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` Datei definiert ist.

`userData`

Informationen in der `userdata.json` Datei.

`config`

Informationen in der `config.json` Datei.

Sie können den Kontext mithilfe der JSONPath Notation abfragen. Die Syntax für JSONPath Abfragen in Statusdefinitionen lautet`{{query}}`. Wenn Sie auf Daten aus dem Test Orchestrator-Kontext zugreifen, stellen Sie sicher, dass jeder Wert eine Zeichenfolge, eine Zahl oder einen booleschen Wert ergibt.

Weitere Hinweise zur Verwendung der JSONPath Notation für den Zugriff auf Daten aus dem Kontext finden Sie unter. Verwenden Sie den IDT-Kontext

Konfigurieren Sie die IDT-Zustandsmaschine

> **Important**
>
> Ab IDT v4.5.1 ist diese Zustandsmaschine veraltet. Wir empfehlen dringend, den neuen Test-Orchestrator zu verwenden. Weitere Informationen finden Sie unter Konfigurieren Sie den IDT-Testorchestrator.

Eine Zustandsmaschine ist ein Konstrukt, das den Ausführungsablauf der Testsuite steuert. Sie bestimmt den Startstatus einer Testsuite, verwaltet Zustandsübergänge auf der Grundlage benutzerdefinierter Regeln und setzt den Übergang durch diese Zustände fort, bis der Endstatus erreicht ist.

Wenn Ihre Testsuite keine benutzerdefinierte Zustandsmaschine enthält, generiert IDT eine Zustandsmaschine für Sie. Die Standard-Zustandsmaschine erfüllt die folgenden Funktionen:

- Bietet Testläufern die Möglichkeit, anstelle der gesamten Testsuite bestimmte Testgruppen auszuwählen und auszuführen.
- Wenn keine bestimmten Testgruppen ausgewählt sind, wird jede Testgruppe in der Testsuite in zufälliger Reihenfolge ausgeführt.
- Generiert Berichte und druckt eine Konsolenübersicht aus, in der die Testergebnisse für jede Testgruppe und jeden Testfall angezeigt werden.

Die Zustandsmaschine für eine IDT-Testsuite muss die folgenden Kriterien erfüllen:

- Jeder Status entspricht einer Aktion, die IDT ausführen muss, z. B. dem Ausführen einer Testgruppe oder eines Produkts oder einer Berichtsdatei.
- Beim Übergang zu einem Status wird die mit dem Status verknüpfte Aktion ausgeführt.
- Jeder Status definiert die Übergangsregel für den nächsten Status.
- Der Endstatus muss entweder `Succeed` oder sein`Fail`.

Format der Zustandsmaschine

Sie können die folgende Vorlage verwenden, um Ihre eigene *<custom-test-suite-folder>*`/suite/state_machine.json` Datei zu konfigurieren:

```
{
  "Comment": "<description>",
  "StartAt": "<state-name>",
  "States": {
    "<state-name>": {
      "Type": "<state-type>",
      // Additional state configuration
    }

    // Required states
    "Succeed": {
      "Type": "Succeed"
    },
    "Fail": {
      "Type": "Fail"
    }
  }
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`Comment`

Eine Beschreibung der Zustandsmaschine.

`StartAt`

Der Name des Status, in dem IDT mit der Ausführung der Testsuite beginnt. Der Wert von `StartAt` muss auf einen der im `States` Objekt aufgelisteten Zustände gesetzt werden.

`States`

Ein Objekt, das benutzerdefinierte Statusnamen gültigen IDT-Staaten zuordnet. Jeder Bundesstaat. *`state-name`*Objekt enthält die Definition eines gültigen Zustands, der dem zugeordnet ist*`state-name`*.

Das `States` Objekt muss die `Fail` Zustände `Succeed` und enthalten. Hinweise zu gültigen Bundesstaaten finden Sie unterGültige Staaten und Bundesstaatendefinitionen.

Gültige Staaten und Bundesstaatendefinitionen

In diesem Abschnitt werden die Zustandsdefinitionen aller gültigen Staaten beschrieben, die in der IDT-Zustandsmaschine verwendet werden können. Einige der folgenden Staaten unterstützen Konfigurationen auf Testfallebene. Wir empfehlen jedoch, Regeln für den Statusübergang auf Testgruppenebene statt auf Testfallebene zu konfigurieren, sofern dies nicht unbedingt erforderlich ist.

Definitionen von Bundesstaaten

- RunTask
- Choice
- Parallel
- AddProductFeatures
- Bericht
- LogMessage
- SelectGroup
- Fehler
- Succeed

RunTask

Der `RunTask` Staat führt Testfälle aus einer in der Testsuite definierten Testgruppe aus.

```
{
```

```
    "Type": "RunTask",
    "Next": "<state-name>",
    "TestGroup": "<group-id>",
    "TestCases": [
        "<test-id>"
    ],
    "ResultVar": "<result-name>"
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`Next`

Der Name des Status, zu dem nach der Ausführung der Aktionen im aktuellen Status übergegangen werden soll.

`TestGroup`

Optional. Die ID der Testgruppe, die ausgeführt werden soll. Wenn dieser Wert nicht angegeben ist, führt IDT die Testgruppe aus, die der Testläufer auswählt.

`TestCases`

Optional. Ein Array von Testfällen IDs aus der in `TestGroup` angegebenen Gruppe. Basierend auf den Werten von `TestGroup` und `TestCases` bestimmt IDT das Verhalten der Testausführung wie folgt:

- Wenn `TestGroup` sowohl als auch angegeben `TestCases` sind, führt IDT die angegebenen Testfälle aus der Testgruppe aus.
- Wenn `TestCases` angegeben, aber nicht angegeben `TestGroup` ist, führt IDT die angegebenen Testfälle aus.
- Wenn `TestGroup` angegeben, aber nicht angegeben `TestCases` ist, führt IDT alle Testfälle innerhalb der angegebenen Testgruppe aus.
- Wenn weder `TestGroup` oder angegeben `TestCases` ist, führt IDT alle Testfälle aus der Testgruppe aus, die der Testläufer aus der IDT-CLI auswählt. Um die Gruppenauswahl für Testläufer zu aktivieren, müssen Sie `RunTask` sowohl Status als auch `Choice` Status in Ihre `state_machine.json` Datei aufnehmen. Ein Beispiel dafür, wie das funktioniert, finden Sie unter Beispiel für eine Zustandsmaschine: Vom Benutzer ausgewählte Testgruppen ausführen.

  Weitere Informationen zur Aktivierung von IDT-CLI-Befehlen für Testläufer finden Sie unterthe section called “IDT-CLI-Befehle aktivieren”.

`ResultVar`

Der Name der Kontextvariablen, die mit den Ergebnissen des Testlaufs festgelegt werden soll. Geben Sie diesen Wert nicht an, wenn Sie keinen Wert für angegeben habenTestGroup. IDT legt den Wert der Variablen, die Sie definieren, auf `true` oder `ResultVar` auf der `false` Grundlage der folgenden Werte fest:

- Wenn der Variablenname die Form hat*text*_*text*_passed, wird der Wert darauf gesetzt, ob alle Tests in der ersten Testgruppe bestanden oder übersprungen wurden.
- In allen anderen Fällen wird der Wert darauf gesetzt, ob alle Tests in allen Testgruppen bestanden wurden oder ob sie übersprungen wurden.

In der Regel verwenden Sie `RunTask` state, um eine Testgruppen-ID ohne Angabe eines einzelnen Testfalls anzugeben IDs, sodass IDT alle Testfälle in der angegebenen Testgruppe ausführt. Alle Testfälle, die von diesem Status ausgeführt werden, werden parallel in zufälliger Reihenfolge ausgeführt. Wenn jedoch für alle Testfälle ein Gerät ausgeführt werden muss und nur ein einziges Gerät verfügbar ist, werden die Testfälle stattdessen sequentiell ausgeführt.

Fehlerbehandlung

Wenn eine der angegebenen Testgruppen oder Testfälle nicht gültig ist, IDs gibt dieser Status den `RunTaskError` Ausführungsfehler aus. Wenn im Status ein Ausführungsfehler auftritt, wird auch die `hasExecutionError` Variable im Zustandsmaschinenkontext auf gesetzt`true`.

Choice

Mit `Choice` diesem Status können Sie basierend auf benutzerdefinierten Bedingungen dynamisch den nächsten Status festlegen, zu dem der Übergang erfolgen soll.

```
{
    "Type": "Choice",
    "Default": "<state-name>",
    "FallthroughOnError": true | false,
    "Choices": [
        {
            "Expression": "<expression>",
            "Next": "<state-name>"
        }
    ]
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`Default`

Der Standardstatus, in den der Übergang erfolgen soll, wenn keiner der in definierten Ausdrücke ausgewertet werden `Choices` kann. `true`

`FallthroughOnError`

Optional. Gibt das Verhalten an, wenn der Status bei der Auswertung von Ausdrücken auf einen Fehler stößt. Legt fest, `true` ob Sie einen Ausdruck überspringen möchten, wenn die Auswertung zu einem Fehler führt. Wenn keine Ausdrücke übereinstimmen, wechselt die Zustandsmaschine in den `Default` Status. Wenn der `FallthroughOnError` Wert nicht angegeben ist, wird standardmäßig der Wert verwendet. `false`

`Choices`

Eine Reihe von Ausdrücken und Zuständen, um zu bestimmen, in welchen Status nach der Ausführung der Aktionen im aktuellen Status übergegangen werden soll.

`Choices.Expression`

Eine Ausdruckszeichenfolge, die einen booleschen Wert ergibt. Wenn der Ausdruck zu ausgewertet wird`true`, geht die Zustandsmaschine in den Zustand über, der in definiert ist. `Choices.Next` Ausdruckszeichenfolgen rufen Werte aus dem Zustandsmaschinen-Kontext ab und führen dann Operationen an ihnen durch, um einen booleschen Wert zu erhalten. Hinweise zum Zugriff auf den Zustandsmaschinenkontext finden Sie unter. Kontext der Zustandsmaschine

`Choices.Next`

Der Name des Zustands, zu dem der Übergang erfolgen soll, wenn der in definierte Ausdruck zu `Choices.Expression` ausgewertet wird. `true`

Fehlerbehandlung

In den folgenden Fällen kann für den `Choice` Status eine Fehlerbehandlung erforderlich sein:

- Einige Variablen in den Auswahlausdrücken sind im Zustandsmaschinen-Kontext nicht vorhanden.
- Das Ergebnis eines Ausdrucks ist kein boolescher Wert.
- Das Ergebnis einer JSON-Suche ist keine Zeichenfolge, Zahl oder boolescher Wert.

In diesem Status können Sie keinen `Catch` Block verwenden, um Fehler zu behandeln. Wenn Sie die Ausführung der Zustandsmaschine beenden möchten, wenn sie auf einen Fehler stößt, müssen Sie `FallthroughOnError` auf einstellen`false`. Wir empfehlen jedoch, dass Sie die Einstellung `FallthroughOnError` auf `true` festlegen und je nach Anwendungsfall eine der folgenden Aktionen ausführen:

- Wenn davon ausgegangen wird, dass eine Variable, auf die Sie zugreifen, in einigen Fällen nicht existiert, verwenden Sie den Wert von `Default` und zusätzliche `Choices` Blöcke, um den nächsten Status anzugeben.
- Wenn eine Variable, auf die Sie zugreifen, immer existieren sollte, setzen Sie den `Default` Status auf`Fail`.

Parallel

Mit dem `Parallel` Status können Sie neue Zustandsmaschinen definieren und parallel zueinander ausführen.

```
{
    "Type": "Parallel",
    "Next": "<state-name>",
    "Branches": [
        <state-machine-definition>
    ]
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`Next`

Der Name des Status, zu dem nach der Ausführung der Aktionen im aktuellen Status übergegangen werden soll.

`Branches`

Eine Reihe von Zustandsmaschinendefinitionen, die ausgeführt werden sollen. Jede Zustandsmaschinen-Definition muss ihre eigenen `StartAtSucceed`, und `Fail` -Zustände enthalten. Die Zustandsmaschinendefinitionen in diesem Array können nicht auf Zustände verweisen, die außerhalb ihrer eigenen Definition liegen.

> **Note**
>
> Da jeder Zustandsmaschine denselben Zustandsmaschinenkontext verwendet, kann das Setzen von Variablen in einem Zweig und das anschließende Lesen dieser Variablen aus einem anderen Zweig zu unerwartetem Verhalten führen.

Der `Parallel` Status wechselt erst in den nächsten Status, nachdem er alle Branch-State-Machines ausgeführt hat. Jeder Status, für den ein Gerät erforderlich ist, wartet mit der Ausführung, bis das Gerät verfügbar ist. Wenn mehrere Geräte verfügbar sind, führt dieser Status Testfälle aus mehreren Gruppen parallel aus. Wenn nicht genügend Geräte verfügbar sind, werden die Testfälle nacheinander ausgeführt. Da Testfälle in zufälliger Reihenfolge ausgeführt werden, wenn sie parallel ausgeführt werden, können verschiedene Geräte verwendet werden, um Tests derselben Testgruppe auszuführen.

Fehlerbehandlung

Stellen Sie sicher, dass sowohl die Zweigzustandsmaschine als auch die übergeordnete Zustandsmaschine in den `Fail` Status wechseln, um Ausführungsfehler zu beheben.

Da Branch-State-Maschinen keine Ausführungsfehler an den übergeordneten Zustandsmaschinen übertragen, können Sie einen `Catch` Block nicht verwenden, um Ausführungsfehler in Branch-State-Machines zu behandeln. Verwenden Sie den `hasExecutionErrors` Wert stattdessen im Kontext des Shared State Machines. Ein Beispiel dafür, wie das funktioniert, finden Sie unterBeispiel State Machine: Zwei Testgruppen parallel ausführen.

AddProductFeatures

Mit `AddProductFeatures` diesem Status können Sie der von IDT generierten `awsiotdevicetester_report.xml` Datei Produktmerkmale hinzufügen.

Bei einer Produktfunktion handelt es sich um benutzerdefinierte Informationen über bestimmte Kriterien, die ein Gerät möglicherweise erfüllt. Beispielsweise kann die `MQTT` Produktfunktion angeben, dass das Gerät MQTT-Nachrichten ordnungsgemäß veröffentlicht. Im Bericht werden Produktfunktionen als, oder als benutzerdefinierter Wert festgelegt `supportednot-supported`, je nachdem, ob die angegebenen Tests bestanden wurden.

> **Note**
>
> Der `AddProductFeatures` Staat generiert selbst keine Berichte. Dieser Status muss in den `ReportStatus` übergehen, in dem Berichte generiert werden können.

```
{
    "Type": "Parallel",
    "Next": "<state-name>",
    "Features": [
        {
            "Feature": "<feature-name>",
            "Groups": [
                "<group-id>"
            ],
            "OneOfGroups": [
                "<group-id>"
            ],
            "TestCases": [
                "<test-id>"
            ],
            "IsRequired": true | false,
            "ExecutionMethods": [
                "<execution-method>"
            ]
        }
    ]
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`Next`

Der Name des Status, zu dem nach der Ausführung der Aktionen im aktuellen Status übergegangen werden soll.

`Features`

Eine Reihe von Produktfunktionen, die in der `awsiotdevicetester_report.xml` Datei angezeigt werden sollen.

`Feature`

Der Name der Funktion

`FeatureValue`

Optional. Der benutzerdefinierte Wert, der anstelle von im Bericht verwendet werden soll`supported`. Wenn dieser Wert nicht angegeben ist, wird der Feature-Wert basierend auf den Testergebnissen auf `supported` oder gesetzt`not-supported`.

Wenn Sie einen benutzerdefinierten Wert für verwenden`FeatureValue`, können Sie dasselbe Feature mit unterschiedlichen Bedingungen testen, und IDT verkettet die Feature-Werte für die unterstützten Bedingungen. Der folgende Auszug zeigt beispielsweise das `MyFeature` Feature mit zwei separaten Feature-Werten:

```
...
{
    "Feature": "MyFeature",
    "FeatureValue": "first-feature-supported",
    "Groups": ["first-feature-group"]
},
{
    "Feature": "MyFeature",
    "FeatureValue": "second-feature-supported",
    "Groups": ["second-feature-group"]
},
...
```

Wenn beide Testgruppen erfolgreich sind, wird der Feature-Wert auf `first-feature-supported, second-feature-supported` gesetzt.

`Groups`

Optional. Ein Array von Testgruppen IDs. Alle Tests innerhalb jeder angegebenen Testgruppe müssen bestanden werden, damit die Funktion unterstützt wird.

`OneOfGroups`

Optional. Ein Array von Testgruppen IDs. Alle Tests innerhalb mindestens einer der angegebenen Testgruppen müssen bestanden werden, damit die Funktion unterstützt wird.

`TestCases`

Optional. Eine Reihe von Testfällen IDs. Wenn Sie diesen Wert angeben, gilt Folgendes:

- Alle angegebenen Testfälle müssen bestanden werden, damit die Funktion unterstützt wird.
- `Groups`darf nur eine Testgruppen-ID enthalten.
- `OneOfGroups`darf nicht angegeben werden.

`IsRequired`

Optional. Stellen Sie auf ein`false`, um diese Funktion im Bericht als optionale Funktion zu kennzeichnen. Der Standardwert ist `true`.

`ExecutionMethods`

Optional. Eine Reihe von Ausführungsmethoden, die dem in der `device.json` Datei angegebenen `protocol` Wert entsprechen. Wenn dieser Wert angegeben ist, müssen Testläufer einen `protocol` Wert angeben, der einem der Werte in diesem Array entspricht, um das Feature in den Bericht aufzunehmen. Wenn dieser Wert nicht angegeben wird, wird das Feature immer in den Bericht aufgenommen.

Um den `AddProductFeatures` Status verwenden zu können, müssen Sie den Wert von `ResultVar` in the `RunTask` state auf einen der folgenden Werte festlegen:

- Wenn Sie einen einzelnen Testfall angegeben haben IDs, legen Sie `ResultVar` den Wert auf fest*`group-id_test-id`*`_passed`.
- Wenn Sie keinen individuellen Testfall angegeben haben IDs, legen Sie `ResultVar` den Wert auf fest*`group-id`*`_passed`.

Der `AddProductFeatures` Staat sucht auf folgende Weise nach Testergebnissen:

- Wenn Sie keinen Testfall angegeben haben IDs, wird das Ergebnis für jede Testgruppe anhand des Werts der *`group-id`*`_passed` Variablen im State-Machine-Kontext bestimmt.
- Wenn Sie einen Testfall angegeben haben IDs, wird das Ergebnis für jeden der Tests anhand des Werts der *`group-id_test-id`*`_passed` Variablen im Zustandsmaschinen-Kontext bestimmt.

Fehlerbehandlung

Wenn eine in diesem Status angegebene Gruppen-ID keine gültige Gruppen-ID ist, führt dieser Status zu einem `AddProductFeaturesError` Ausführungsfehler. Wenn im Status ein Ausführungsfehler auftritt, wird auch die `hasExecutionErrors` Variable im Zustandsmaschinenkontext auf gesetzt`true`.

Bericht

Der `Report` Status generiert die `awsiotdevicetester_report.xml` Dateien *`suite-name`*`_Report.xml` und. In diesem Status wird der Bericht auch an die Konsole gestreamt.

```
{
    "Type": "Report",
    "Next": "<state-name>"
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`Next`

Der Name des Status, zu dem nach der Ausführung der Aktionen im aktuellen Status übergegangen werden soll.

Sie sollten immer gegen Ende der Testausführung in den `Report` Status wechseln, damit Testläufer die Testergebnisse einsehen können. In der Regel ist der nächste Status nach diesem Status`Succeed`.

Fehlerbehandlung

Wenn in diesem Status Probleme beim Generieren der Berichte auftreten, wird der `ReportError` Ausführungsfehler ausgegeben.

LogMessage

Der `LogMessage` Status generiert die `test_manager.log` Datei und streamt die Protokollnachricht an die Konsole.

```
{
    "Type": "LogMessage",
    "Next": "<state-name>"
    "Level": "info | warn | error"
    "Message": "<message>"
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`Next`

Der Name des Status, zu dem nach der Ausführung der Aktionen im aktuellen Status übergegangen werden soll.

`Level`

Die Fehlerstufe, auf der die Protokollnachricht erstellt werden soll. Wenn Sie eine ungültige Stufe angeben, generiert dieser Status eine Fehlermeldung und verwirft sie.

`Message`

Die zu protokollierende Nachricht.

SelectGroup

Der `SelectGroup` Status aktualisiert den Kontext der Zustandsmaschine, um anzugeben, welche Gruppen ausgewählt wurden. Die in diesem Status festgelegten Werte werden von allen nachfolgenden `Choice` Staaten verwendet.

```
{
    "Type": "SelectGroup",
    "Next": "<state-name>"
    "TestGroups": [
        <group-id>"
    ]
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`Next`

Der Name des Status, zu dem nach der Ausführung der Aktionen im aktuellen Status übergegangen werden soll.

`TestGroups`

Eine Reihe von Testgruppen, die als ausgewählt markiert werden. Für jede Testgruppen-ID in diesem Array wird die *`group-id`*`_selected` Variable `true` im Kontext auf gesetzt. Stellen Sie sicher, dass Sie eine gültige Testgruppe angeben IDs , da IDT nicht überprüft, ob die angegebenen Gruppen existieren.

Fehler

Der `Fail` Status weist darauf hin, dass die Zustandsmaschine nicht korrekt ausgeführt wurde. Dies ist ein Endzustand für die Zustandsmaschine, und jede Zustandsmaschine muss diesen Zustand enthalten.

```
{
    "Type": "Fail"
}
```

Succeed

Der `Succeed` Status gibt an, dass die Zustandsmaschine korrekt ausgeführt wurde. Dies ist ein Endzustand für die Zustandsmaschine, und jede Zustandsmaschine muss diesen Zustand enthalten.

```
{
    "Type": "Succeed"
}
```

Kontext der Zustandsmaschine

Der Zustandsmaschinenkontext ist ein schreibgeschütztes JSON-Dokument, das Daten enthält, die der Zustandsmaschine während der Ausführung zur Verfügung stehen. Der Zustandsmaschinen-Kontext ist nur von der Zustandsmaschine aus zugänglich und enthält Informationen, die den Testablauf bestimmen. Sie können beispielsweise Informationen verwenden, die von Testläufern in der `userdata.json` Datei konfiguriert wurden, um festzustellen, ob ein bestimmter Test ausgeführt werden muss.

Der State-Machine-Kontext verwendet das folgende Format:

```
{
    "pool": {
        <device-json-pool-element>
    },
    "userData": {
        <userdata-json-content>
    },
    "config": {
        <config-json-content>
    },
```

```
    "suiteFailed": true | false,
    "specificTestGroups": [
        "<group-id>"
    ],
    "specificTestCases": [
        "<test-id>"
    ],
    "hasExecutionErrors": true
}
```

`pool`

Informationen über den Gerätepool, der für den Testlauf ausgewählt wurde. Für einen ausgewählten Gerätepool werden diese Informationen aus dem entsprechenden Gerätepool-Array-Element der obersten Ebene abgerufen, das in der `device.json` Datei definiert ist.

`userData`

Informationen in der `userdata.json` Datei.

`config`

Informationen an die `config.json` Datei anheften.

`suiteFailed`

Der Wert wird auf den `false` Zeitpunkt gesetzt, zu dem die Zustandsmaschine gestartet wird. Wenn eine Testgruppe in einem `RunTask` Status ausfällt, wird dieser Wert `true` für die verbleibende Dauer der State-Machine-Ausführung auf gesetzt.

`specificTestGroups`

Wenn der Testläufer anstelle der gesamten Testsuite bestimmte Testgruppen zur Ausführung auswählt, wird dieser Schlüssel erstellt und enthält die Liste der spezifischen Testgruppen IDs.

`specificTestCases`

Wenn der Testläufer anstelle der gesamten Testsuite bestimmte Testfälle zur Ausführung auswählt, wird dieser Schlüssel erstellt und enthält die Liste der spezifischen Testfälle IDs.

`hasExecutionErrors`

Wird nicht beendet, wenn die Zustandsmaschine gestartet wird. Wenn in einem Status ein Ausführungsfehler auftritt, wird diese Variable erstellt und `true` für die verbleibende Dauer der State-Machine-Ausführung auf „gesetzt“.

Sie können den Kontext mithilfe der JSONPath Notation abfragen. Die Syntax für JSONPath Abfragen in Statusdefinitionen lautet`{{$.query}}`. In einigen Bundesstaaten können Sie JSONPath Abfragen als Platzhalterzeichenfolgen verwenden. IDT ersetzt die Platzhalterzeichenfolgen durch den Wert der ausgewerteten JSONPath Abfrage aus dem Kontext. Sie können Platzhalter für die folgenden Werte verwenden:

- Der `TestCases` Wert in `RunTask` Bundesstaaten.
- Der `Expression` `Choice` Wertstatus.

Wenn Sie auf Daten aus dem State Machine-Kontext zugreifen, stellen Sie sicher, dass die folgenden Bedingungen erfüllt sind:

- Ihre JSON-Pfade müssen beginnen mit `$.`
- Jeder Wert muss eine Zeichenfolge, eine Zahl oder einen booleschen Wert ergeben.

Weitere Hinweise zur Verwendung der JSONPath Notation für den Zugriff auf Daten aus dem Kontext finden Sie unter. Verwenden Sie den IDT-Kontext

Ausführungsfehler

Ausführungsfehler sind Fehler in der Zustandsmaschinen-Definition, auf die der Zustandsmaschine bei der Ausführung eines Zustands stößt. IDT protokolliert Informationen zu jedem Fehler in der `test_manager.log` Datei und streamt die Protokollnachricht an die Konsole.

Sie können die folgenden Methoden verwenden, um Ausführungsfehler zu behandeln:

- Fügen Sie der Statusdefinition einen `CatchBlock` hinzu.
- Überprüfen Sie den `hasExecutionErrors`Wert des Werts im Kontext der Zustandsmaschine.

Fangen

Um es zu verwenden`Catch`, fügen Sie Ihrer Bundesstaatendefinition Folgendes hinzu:

```
"Catch": [
    {
        "ErrorEquals": [
            "<error-type>"
        ]
        "Next": "<state-name>"
```

```
    }
]
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`Catch.ErrorEquals`

Eine Reihe von Fehlertypen, die abgefangen werden sollen. Wenn ein Ausführungsfehler mit einem der angegebenen Werte übereinstimmt, wechselt die Zustandsmaschine in den unter angegebenen Status`Catch.Next`. In den einzelnen Statusdefinitionen finden Sie Informationen zur Art des Fehlers, den sie erzeugt.

`Catch.Next`

Der nächste Status, in den übergegangen werden soll, wenn im aktuellen Status ein Ausführungsfehler auftritt, der einem der in angegebenen Werte entspricht`Catch.ErrorEquals`.

Catch-Blöcke werden sequentiell behandelt, bis einer übereinstimmt. Wenn der Wert „Keine Fehler“ mit den in den Catch-Blöcken aufgelisteten Fehlern übereinstimmt, wird die Ausführung der Zustandsmaschinen fortgesetzt. Da Ausführungsfehler auf falsche Statusdefinitionen zurückzuführen sind, empfehlen wir, dass Sie in den Status Fail wechseln, wenn in einem Status ein Ausführungsfehler auftritt.

hasExecutionError

Wenn in einigen Staaten Ausführungsfehler auftreten, geben sie nicht nur den Fehler aus, sondern setzen den `hasExecutionError` Wert auch `true` im Kontext der Zustandsmaschine auf. Sie können diesen Wert verwenden, um zu erkennen, wann ein Fehler auftritt, und dann einen `Choice` Status verwenden, um die Zustandsmaschine in den `Fail` Status zu versetzen.

Diese Methode hat die folgenden Eigenschaften.

- Die Zustandsmaschine beginnt mit keinem Wert, der zugewiesen wurde`hasExecutionError`, und dieser Wert ist erst verfügbar, wenn ein bestimmter Status ihn festlegt. Das bedeutet, dass Sie den Status `false` für die `Choice` Staaten, die `Fallthrough0nError` auf diesen Wert zugreifen, explizit auf setzen müssen, um zu verhindern, dass der Zustandsmaschine angehalten wird, wenn keine Ausführungsfehler auftreten.
- Sobald der Wert auf gesetzt ist`true`, `hasExecutionError` wird er niemals auf False gesetzt oder aus dem Kontext entfernt. Das bedeutet, dass dieser Wert nur nützlich ist, wenn er zum ersten Mal auf gesetzt wird`true`, und für alle nachfolgenden Zustände bietet er keinen aussagekräftigen Wert.

- Der `hasExecutionError` Wert wird von allen Zweigstatusmaschinen im `Parallel` Bundesstaat gemeinsam genutzt, was je nach Reihenfolge, in der auf ihn zugegriffen wird, zu unerwarteten Ergebnissen führen kann.

Aufgrund dieser Eigenschaften empfehlen wir nicht, diese Methode zu verwenden, wenn Sie stattdessen einen Catch-Block verwenden können.

Beispiel für Zustandsmaschinen

Dieser Abschnitt enthält einige Beispielkonfigurationen von Zustandsmaschinen.

Beispiele

- Beispiel für eine Zustandsmaschine: Führen Sie eine einzelne Testgruppe aus
- Beispiel für eine Zustandsmaschine: Führen Sie vom Benutzer ausgewählte Testgruppen aus
- Beispiel für eine Zustandsmaschine: Führen Sie eine einzelne Testgruppe mit Produktfunktionen aus
- Beispiel State Machine: Zwei Testgruppen parallel ausführen

Beispiel für eine Zustandsmaschine: Führen Sie eine einzelne Testgruppe aus

Dieser Zustandsmaschine:

- Führt die Testgruppe mit der ID aus`GroupA`, die in der Suite in einer `group.json` Datei vorhanden sein muss.
- Prüft auf Ausführungsfehler und wechselt zu, `Fail` ob welche gefunden wurden.
- Generiert einen Bericht und wechselt zu, `Succeed` ob keine Fehler vorliegen, und `Fail` andernfalls.

```
{
    "Comment": "Runs a single group and then generates a report.",
    "StartAt": "RunGroupA",
    "States": {
        "RunGroupA": {
            "Type": "RunTask",
            "Next": "Report",
            "TestGroup": "GroupA",
            "Catch": [
```

```
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Report": {
            "Type": "Report",
            "Next": "Succeed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "ReportError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

Beispiel für eine Zustandsmaschine: Führen Sie vom Benutzer ausgewählte Testgruppen aus

Dieser Zustandsmaschine:

- Prüft, ob der Testläufer bestimmte Testgruppen ausgewählt hat. Die Zustandsmaschine sucht nicht nach bestimmten Testfällen, da Testläufer keine Testfälle auswählen können, ohne auch eine Testgruppe auszuwählen.
- Wenn Testgruppen ausgewählt sind:
  - Führt die Testfälle innerhalb der ausgewählten Testgruppen aus. Zu diesem Zweck spezifiziert die Zustandsmaschine nicht explizit Testgruppen oder Testfälle im `RunTask` Status.
  - Generiert einen Bericht, nachdem alle Tests ausgeführt und beendet wurden.
- Wenn keine Testgruppen ausgewählt sind:

- Führt Tests in einer Testgruppe aus`GroupA`.
- Generiert Berichte und beendet das Programm.

```
{
    "Comment": "Runs specific groups if the test runner chose to do that, otherwise
 runs GroupA.",
    "StartAt": "SpecificGroupsCheck",
    "States": {
        "SpecificGroupsCheck": {
            "Type": "Choice",
            "Default": "RunGroupA",
            "FallthroughOnError": true,
            "Choices": [
                {
                    "Expression": "{{$.specificTestGroups[0]}} != ''",
                    "Next": "RunSpecificGroups"
                }
            ]
        },
        "RunSpecificGroups": {
            "Type": "RunTask",
            "Next": "Report",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "RunGroupA": {
            "Type": "RunTask",
            "Next": "Report",
            "TestGroup": "GroupA",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
```

```
            ]
        },
        "Report": {
            "Type": "Report",
            "Next": "Succeed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "ReportError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

Beispiel für eine Zustandsmaschine: Führen Sie eine einzelne Testgruppe mit Produktfunktionen aus

Dieser Zustandsmaschine:

- Führt die Testgruppe aus`GroupA`.
- Prüft auf Ausführungsfehler und wechselt, `Fail` ob welche gefunden wurden.
- Fügt der `awsiotdevicetester_report.xml` Datei das `FeatureThatDependsOnGroupA` Feature hinzu:
  - Wenn der `GroupA` Wert erfolgreich ist, wird das Feature auf gesetzt`supported`.
  - Die Funktion ist im Bericht nicht als optional gekennzeichnet.
- Generiert einen Bericht und wechselt zu, `Succeed` wenn keine Fehler vorliegen, und `Fail` andernfalls

```
{
    "Comment": "Runs GroupA and adds product features based on GroupA",
    "StartAt": "RunGroupA",
    "States": {
```

```
        "RunGroupA": {
            "Type": "RunTask",
            "Next": "AddProductFeatures",
            "TestGroup": "GroupA",
            "ResultVar": "GroupA_passed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "AddProductFeatures": {
            "Type": "AddProductFeatures",
            "Next": "Report",
            "Features": [
                {
                    "Feature": "FeatureThatDependsOnGroupA",
                    "Groups": [
                        "GroupA"
                    ],
                    "IsRequired": true
                }
            ]
        },
        "Report": {
            "Type": "Report",
            "Next": "Succeed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "ReportError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
```

```
        }
    }
}
```

Beispiel State Machine: Zwei Testgruppen parallel ausführen

Diese Zustandsmaschine:

- Führt die Gruppen `GroupA` und die `GroupB` Testgruppen parallel aus. Die `ResultVar` Variablen, die im Kontext der `RunTask` Bundesstaaten gespeichert sind, stehen dem `AddProductFeatures` Staat zur Verfügung.
- Prüft auf Ausführungsfehler und wechselt, `Fail` falls welche gefunden wurden. Diese Zustandsmaschine verwendet keinen `Catch` Block, da diese Methode keine Ausführungsfehler in Zweigzustandsmaschinen erkennt.
- Fügt der `awsiotdevicetester_report.xml` Datei Funktionen hinzu, die auf den Gruppen basieren, die erfolgreich sind
  - Bei `GroupA` erfolgreicher Prüfung wird das Feature auf gesetzt`supported`.
  - Die Funktion ist im Bericht nicht als optional gekennzeichnet.
- Generiert einen Bericht und wechselt zu, `Succeed` wenn keine Fehler vorliegen, und `Fail` andernfalls

Wenn zwei Geräte im Gerätepool konfiguriert sind, `GroupB` können beide `GroupA` Geräte gleichzeitig ausgeführt werden. Wenn jedoch einer `GroupA` oder `GroupB` mehrere Tests enthalten sind, können beide Geräte diesen Tests zugewiesen werden. Wenn nur ein Gerät konfiguriert ist, werden die Testgruppen nacheinander ausgeführt.

```
{
    "Comment": "Runs GroupA and GroupB in parallel",
    "StartAt": "RunGroupAAndB",
    "States": {
        "RunGroupAAndB": {
            "Type": "Parallel",
            "Next": "CheckForErrors",
            "Branches": [
                {
                    "Comment": "Run GroupA state machine",
                    "StartAt": "RunGroupA",
                    "States": {
                        "RunGroupA": {
```

```
                    "Type": "RunTask",
                    "Next": "Succeed",
                    "TestGroup": "GroupA",
                    "ResultVar": "GroupA_passed",
                    "Catch": [
                        {
                            "ErrorEquals": [
                                "RunTaskError"
                            ],
                            "Next": "Fail"
                        }
                    ]
                },
                "Succeed": {
                    "Type": "Succeed"
                },
                "Fail": {
                    "Type": "Fail"
                }
            }
        },
        {
            "Comment": "Run GroupB state machine",
            "StartAt": "RunGroupB",
            "States": {
                "RunGroupA": {
                    "Type": "RunTask",
                    "Next": "Succeed",
                    "TestGroup": "GroupB",
                    "ResultVar": "GroupB_passed",
                    "Catch": [
                        {
                            "ErrorEquals": [
                                "RunTaskError"
                            ],
                            "Next": "Fail"
                        }
                    ]
                },
                "Succeed": {
                    "Type": "Succeed"
                },
                "Fail": {
                    "Type": "Fail"
```

```
                    }
                }
            }
        ]
    },
    "CheckForErrors": {
        "Type": "Choice",
        "Default": "AddProductFeatures",
        "FallthroughOnError": true,
        "Choices": [
            {
                "Expression": "{{$.hasExecutionErrors}} == true",
                "Next": "Fail"
            }
        ]
    },
    "AddProductFeatures": {
        "Type": "AddProductFeatures",
        "Next": "Report",
        "Features": [
            {
                "Feature": "FeatureThatDependsOnGroupA",
                "Groups": [
                    "GroupA"
                ],
                "IsRequired": true
            },
            {
                "Feature": "FeatureThatDependsOnGroupB",
                "Groups": [
                    "GroupB"
                ],
                "IsRequired": true
            }
        ]
    },
    "Report": {
        "Type": "Report",
        "Next": "Succeed",
        "Catch": [
            {
                "ErrorEquals": [
                    "ReportError"
                ],
```

```
                    "Next": "Fail"
                }
            ]
        },
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

Ausführbare IDT-Testfalldateien erstellen

Sie können ausführbare Testfalldateien auf folgende Weise erstellen und in einem Testsuite-Ordner ablegen:

- Für Testsuiten, die Argumente oder Umgebungsvariablen aus den `test.json` Dateien verwenden, um zu bestimmen, welche Tests ausgeführt werden sollen, können Sie einen einzelnen ausführbaren Testfall für die gesamte Testsuite oder eine ausführbare Testdatei für jede Testgruppe in der Testsuite erstellen.
- Für eine Testsuite, in der Sie bestimmte Tests auf der Grundlage bestimmter Befehle ausführen möchten, erstellen Sie für jeden Testfall in der Testsuite eine ausführbare Testfalldatei.

Als Testautor können Sie bestimmen, welcher Ansatz für Ihren Anwendungsfall geeignet ist, und die ausführbare Testfalldatei entsprechend strukturieren. Stellen Sie sicher, dass Sie in jeder `test.json` Datei den richtigen Pfad für die ausführbare Testfalldatei angeben und dass die angegebene ausführbare Datei korrekt ausgeführt wird.

Wenn alle Geräte für die Ausführung eines Testfalls bereit sind, liest IDT die folgenden Dateien:

- Der `test.json` für den ausgewählten Testfall festgelegte Prozess bestimmt, welche Prozesse gestartet und welche Umgebungsvariablen gesetzt werden sollen.
- Die `suite.json` für die Testsuite bestimmt die zu setzenden Umgebungsvariablen.

IDT startet den erforderlichen ausführbaren Testprozess auf der Grundlage der in der `test.json` Datei angegebenen Befehle und Argumente und übergibt die erforderlichen Umgebungsvariablen an den Prozess.

Verwenden Sie das IDT Client SDK

Mit dem IDT-Client SDKs können Sie das Schreiben von Testlogik in Ihre Testdatei mithilfe von API-Befehlen vereinfachen, die Sie verwenden können, um mit IDT und Ihren zu testenden Geräten zu interagieren. IDT bietet derzeit Folgendes: SDKs

- IDT-Client-SDK für Python
- IDT Client SDK for Go
- IDT Client SDK for Java

Diese SDKs befinden sich im *<device-tester-extract-location>*`/sdks` Ordner. Wenn Sie eine neue ausführbare Testfalldatei erstellen, müssen Sie das SDK, das Sie verwenden möchten, in den Ordner kopieren, der Ihre ausführbare Testfalldatei enthält, und in Ihrem Code auf das SDK verweisen. Dieser Abschnitt enthält eine kurze Beschreibung der verfügbaren API-Befehle, die Sie in Ihren ausführbaren Testfalldateien verwenden können.

In diesem Abschnitt

- Geräteinteraktion
- IDT-Interaktion
- Interaktion mit dem Host

Geräteinteraktion

Mit den folgenden Befehlen können Sie mit dem zu testenden Gerät kommunizieren, ohne zusätzliche Funktionen für die Geräteinteraktion und das Konnektivitätsmanagement implementieren zu müssen.

`ExecuteOnDevice`

Ermöglicht es Testsuiten, Shell-Befehle auf einem Gerät auszuführen, das SSH- oder Docker-Shell-Verbindungen unterstützt.

`CopyToDevice`

Ermöglicht Testsuiten, eine lokale Datei vom Host-Computer, auf dem IDT ausgeführt wird, an einen bestimmten Speicherort auf einem Gerät zu kopieren, das SSH- oder Docker-Shell-Verbindungen unterstützt.

`ReadFromDevice`

Ermöglicht es Testsuiten, von der seriellen Schnittstelle von Geräten zu lesen, die UART-Verbindungen unterstützen.

> **Note**
>
> Da IDT keine direkten Verbindungen zu Geräten verwaltet, die mithilfe von Gerätezugriffsinformationen aus dem Kontext hergestellt werden, empfehlen wir, diese API-Befehle für Geräteinteraktionen in Ihren ausführbaren Testfalldateien zu verwenden. Wenn diese Befehle jedoch nicht Ihren Testfallanforderungen entsprechen, können Sie Gerätezugriffsinformationen aus dem IDT-Kontext abrufen und sie verwenden, um über die Testsuite eine direkte Verbindung zum Gerät herzustellen.
> Um eine direkte Verbindung herzustellen, rufen Sie die Informationen in den `device.connectivity` `resource.devices.connectivity` Feldern für Ihr zu testendes Gerät bzw. für Ressourcengeräte ab. Weitere Informationen zur Verwendung des IDT-Kontextes finden Sie unterVerwenden Sie den IDT-Kontext.

IDT-Interaktion

Die folgenden Befehle ermöglichen es Ihren Testsuiten, mit IDT zu kommunizieren.

`PollForNotifications`

Ermöglicht Testsuiten, nach Benachrichtigungen von IDT zu suchen.

`GetContextValue` und `GetContextString`

Ermöglicht Testsuiten das Abrufen von Werten aus dem IDT-Kontext. Weitere Informationen finden Sie unter Verwenden Sie den IDT-Kontext.

`SendResult`

Ermöglicht es Testsuiten, Testfallergebnisse an IDT zu melden. Dieser Befehl muss am Ende jedes Testfalls in einer Testsuite aufgerufen werden.

Interaktion mit dem Host

Mit dem folgenden Befehl können Ihre Testsuiten mit dem Host-Computer kommunizieren.

`PollForNotifications`

Ermöglicht Testsuiten, nach Benachrichtigungen von IDT zu suchen.

`GetContextValue` und `GetContextString`

Ermöglicht Testsuiten das Abrufen von Werten aus dem IDT-Kontext. Weitere Informationen finden Sie unter Verwenden Sie den IDT-Kontext.

`ExecuteOnHost`

Ermöglicht Testsuiten die Ausführung von Befehlen auf dem lokalen Computer und ermöglicht IDT die Verwaltung des Lebenszyklus der ausführbaren Testfälle.

IDT-CLI-Befehle aktivieren

Der `run-suite` Befehl IDT CLI bietet mehrere Optionen, mit denen Test Runner die Testausführung anpassen kann. Um es Testläufern zu ermöglichen, diese Optionen zum Ausführen Ihrer benutzerdefinierten Testsuite zu verwenden, implementieren Sie Unterstützung für die IDT-CLI. Wenn Sie keine Unterstützung implementieren, können Testläufer weiterhin Tests ausführen, aber einige CLI-Optionen funktionieren nicht richtig. Um ein optimales Kundenerlebnis zu bieten, empfehlen wir, die Unterstützung für die folgenden Argumente für den `run-suite` Befehl in der IDT-CLI zu implementieren:

`timeout-multiplier`

Gibt einen Wert größer als 1,0 an, der auf alle Timeouts beim Ausführen von Tests angewendet wird.

Testläufer können dieses Argument verwenden, um das Timeout für die Testfälle zu erhöhen, die sie ausführen möchten. Wenn ein Testläufer dieses Argument in seinem `run-suite` Befehl angibt, verwendet IDT es, um den Wert der Umgebungsvariablen IDT_TEST_TIMEOUT zu berechnen, und legt das Feld im IDT-Kontext fest. `config.timeoutMultiplier` Um dieses Argument zu unterstützen, müssen Sie wie folgt vorgehen:

- Anstatt den Timeout-Wert direkt aus der `test.json` Datei zu verwenden, lesen Sie die Umgebungsvariable IDT_TEST_TIMEOUT, um den korrekt berechneten Timeout-Wert zu erhalten.
- Rufen Sie den `config.timeoutMultiplier` Wert aus dem IDT-Kontext ab und wenden Sie ihn auf Timeouts mit langer Laufzeit an.

Weitere Hinweise zum vorzeitigen Beenden aufgrund von Timeout-Ereignissen finden Sie unter. Geben Sie das Exit-Verhalten an

`stop-on-first-failure`

Gibt an, dass IDT die Ausführung aller Tests beenden soll, wenn ein Fehler auftritt.

Wenn ein Testläufer dieses Argument in seinem `run-suite` Befehl angibt, beendet IDT die Ausführung von Tests, sobald ein Fehler auftritt. Wenn Testfälle jedoch parallel ausgeführt werden, kann dies zu unerwarteten Ergebnissen führen. Um Support zu implementieren, stellen Sie sicher, dass Ihre Testlogik alle laufenden Testfälle anweist, anzuhalten, temporäre Ressourcen zu bereinigen und ein Testergebnis an IDT zu melden, wenn IDT auf dieses Ereignis trifft. Weitere Informationen zum vorzeitigen Beenden von Fehlern finden Sie unter. Geben Sie das Exit-Verhalten an

`group-id` und `test-id`

Gibt an, dass IDT nur die ausgewählten Testgruppen oder Testfälle ausführen soll.

Testläufer können diese Argumente mit ihrem `run-suite` Befehl verwenden, um das folgende Verhalten bei der Testausführung anzugeben:

- Führt alle Tests innerhalb der angegebenen Testgruppen aus.
- Führt eine Auswahl von Tests innerhalb einer angegebenen Testgruppe aus.

Um diese Argumente zu unterstützen, muss der Test-Orchestrator für Ihre Testsuite einen bestimmten Satz von `RunTask Choice` Endstatus in Ihrem Test-Orchestrator enthalten. Wenn Sie keine benutzerdefinierte Zustandsmaschine verwenden, enthält der standardmäßige IDT-Testorchestrator die erforderlichen Status für Sie, und Sie müssen keine zusätzlichen Maßnahmen ergreifen. Wenn Sie jedoch einen benutzerdefinierten Test-Orchestrator verwenden, verwenden Sie ihn Beispiel für eine Zustandsmaschine: Führen Sie vom Benutzer ausgewählte Testgruppen aus als Beispiel, um die erforderlichen Status in Ihrem Test-Orchestrator hinzuzufügen.

Weitere Informationen zu IDT-CLI-Befehlen finden Sie unterDebuggen und Ausführen benutzerdefinierter Testsuiten.

Schreiben Sie Ereignisprotokolle

Während der Test läuft, senden Sie Daten an `stdout` `stderr` die Konsole und schreiben dort Ereignisprotokolle und Fehlermeldungen. Hinweise zum Format von Konsolenmeldungen finden Sie unterNachrichtenformat der Konsole.

Wenn das IDT die Ausführung der Testsuite beendet hat, sind diese Informationen auch in der `test_manager.log` Datei im *<devicetester-extract-location>*`/results/`*<execution-id>*`/logs` Ordner verfügbar.

Sie können jeden Testfall so konfigurieren, dass die Protokolle des Testlaufs, einschließlich der Protokolle des zu testenden Geräts, in die *<group-id>*`_`*<test-id>* Datei im *<device-tester-extract-location>*`/results/`*execution-id*`/logs` Ordner geschrieben werden. Rufen Sie dazu den Pfad zur Protokolldatei aus dem IDT-Kontext mit der `testData.logFilePath` Abfrage ab, erstellen Sie eine Datei unter diesem Pfad und schreiben Sie den gewünschten Inhalt hinein. IDT aktualisiert den Pfad automatisch auf der Grundlage des laufenden Testfalls. Wenn Sie sich dafür entscheiden, die Protokolldatei für einen Testfall nicht zu erstellen, wird keine Datei für diesen Testfall generiert.

Sie können Ihre ausführbare Textdatei auch so einrichten, dass sie bei Bedarf zusätzliche Protokolldateien im *<device-tester-extract-location>*`/logs` Ordner erstellt. Wir empfehlen Ihnen, eindeutige Präfixe für Protokolldateinamen anzugeben, damit Ihre Dateien nicht überschrieben werden.

Ergebnisse an IDT melden

IDT schreibt Testergebnisse in die `awsiotdevicetester_report.xml` und die *suite-name*`_report.xml` Dateien. Diese Berichtsdateien befinden sich in*<device-tester-extract-location>*`/results/`*<execution-id>*`/`. Beide Berichte erfassen die Ergebnisse der Ausführung der Testsuite. Weitere Informationen zu den Schemas, die IDT für diese Berichte verwendet, finden Sie unter Überprüfen Sie die IDT-Testergebnisse und -protokolle

Um den Inhalt der *suite-name*`_report.xml` Datei aufzufüllen, müssen Sie den `SendResult` Befehl verwenden, um die Testergebnisse an IDT zu melden, bevor die Testausführung abgeschlossen ist. Wenn IDT die Ergebnisse eines Tests nicht finden kann, gibt es einen Fehler für den Testfall aus. Der folgende Python-Auszug zeigt die Befehle zum Senden eines Testergebnisses an IDT:

```
request-variable = SendResultRequest(TestResult(result))
```

```
client.send_result(request-variable)
```

Wenn Sie Ergebnisse nicht über die API melden, sucht IDT im Ordner mit den Testartefakten nach Testergebnissen. Der Pfad zu diesem Ordner wird in der Datei im `testData.testArtifactsPath` IDT-Kontext gespeichert. In diesem Ordner verwendet IDT die erste alphabetisch sortierte XML-Datei, die es findet, als Testergebnis.

Wenn Ihre Testlogik JUnit XML-Ergebnisse liefert, können Sie die Testergebnisse in eine XML-Datei im Ordner artefacts schreiben, um die Ergebnisse direkt an IDT weiterzugeben, anstatt die Ergebnisse zu analysieren und sie dann über die API an IDT zu senden.

Wenn Sie diese Methode verwenden, stellen Sie sicher, dass Ihre Testlogik die Testergebnisse korrekt zusammenfasst, und formatieren Sie Ihre Ergebnisdatei im gleichen Format wie die Datei. *`suite-name`*`_report.xml` IDT führt keine Überprüfung der von Ihnen bereitgestellten Daten durch, mit den folgenden Ausnahmen:

- IDT ignoriert alle Eigenschaften des Tags. `testsuites` Stattdessen werden die Tag-Eigenschaften anhand anderer gemeldeter Testgruppenergebnisse berechnet.
- Darin `testsuites` muss mindestens ein `testsuite` Tag vorhanden sein.

Da IDT für alle Testfälle denselben Ordner mit Artefakten verwendet und Ergebnisdateien nicht zwischen Testläufen löscht, kann diese Methode auch zu fehlerhaften Berichten führen, wenn IDT die falsche Datei liest. Es wird empfohlen, für alle Testfälle denselben Namen für die generierte XML-Ergebnisdatei zu verwenden, um die Ergebnisse für jeden Testfall zu überschreiben und sicherzustellen, dass IDT die richtigen Ergebnisse zur Verfügung hat. Sie können in Ihrer Testsuite zwar einen gemischten Ansatz für die Berichterstattung verwenden, d. h. für einige Testfälle eine XML-Ergebnisdatei verwenden und für andere die Ergebnisse über die API einreichen, wir empfehlen diesen Ansatz jedoch nicht.

Geben Sie das Exit-Verhalten an

Konfigurieren Sie Ihre ausführbare Textdatei so, dass sie immer mit dem Exit-Code 0 beendet wird, auch wenn ein Testfall einen Fehler oder ein Fehlerergebnis meldet. Verwenden Sie Exit-Codes ungleich Null nur, um anzuzeigen, dass ein Testfall nicht ausgeführt wurde oder dass die ausführbare Testfalldatei keine Ergebnisse an IDT übermitteln konnte. Wenn IDT einen Exit-Code ungleich Null empfängt, bedeutet dies, dass der Testfall auf einen Fehler gestoßen ist, der seine Ausführung verhindert hat.

IDT kann in den folgenden Fällen anfordern oder erwarten, dass die Ausführung eines Testfalls beendet wird, bevor er abgeschlossen ist. Verwenden Sie diese Informationen, um Ihre ausführbare Testfalldatei so zu konfigurieren, dass jedes dieser Ereignisse anhand des Testfalls erkannt wird:

Timeout (Zeitüberschreitung)

Tritt auf, wenn ein Testfall länger als der in der `test.json` Datei angegebene Timeout-Wert ausgeführt wird. Wenn der Testläufer das `timeout-multiplier` Argument verwendet hat, um einen Timeout-Multiplikator anzugeben, berechnet IDT den Timeout-Wert mit dem Multiplikator.

Verwenden Sie die Umgebungsvariable IDT_TEST_TIMEOUT, um dieses Ereignis zu erkennen. Wenn ein Testläufer einen Test startet, setzt IDT den Wert der Umgebungsvariablen IDT_TEST_TIMEOUT auf den berechneten Timeout-Wert (in Sekunden) und übergibt die Variable an die ausführbare Testfalldatei. Sie können den Variablenwert lesen, um einen geeigneten Timer festzulegen.

Unterbrechen

Tritt auf, wenn der Test-Runner IDT unterbricht. Zum Beispiel durch Drücken von. Ctrl+C

Da Terminals Signale an alle untergeordneten Prozesse weiterleiten, können Sie in Ihren Testfällen einfach einen Signal-Handler konfigurieren, um Interrupt-Signale zu erkennen.

Alternativ können Sie die API regelmäßig abfragen, um den Wert des `CancellationRequested` booleschen Werts in der API-Antwort zu überprüfen. `PollForNotifications`
Wenn IDT ein Interruptsignal empfängt, setzt es den Wert des booleschen Werts auf. `CancellationRequested` `true`

Stoppt beim ersten Fehler

Tritt auf, wenn ein Testfall, der parallel zum aktuellen Testfall ausgeführt wird, fehlschlägt und der Testläufer das `stop-on-first-failure` Argument verwendet hat, um anzugeben, dass IDT beendet werden soll, wenn ein Fehler auftritt.

Um dieses Ereignis zu erkennen, können Sie die API regelmäßig abfragen, um den Wert des `CancellationRequested` booleschen Werts in der `PollForNotifications` API-Antwort zu überprüfen. Wenn IDT auf einen Fehler stößt und so konfiguriert ist, dass es beim ersten Fehler stoppt, wird der Wert des booleschen Werts `CancellationRequested` auf gesetzt. `true`

Wenn eines dieser Ereignisse eintritt, wartet IDT 5 Minuten, bis alle derzeit laufenden Testfälle abgeschlossen sind. Wenn nicht alle laufenden Testfälle innerhalb von 5 Minuten beendet werden,

erzwingt IDT, jeden ihrer Prozesse zu beenden. Wenn IDT vor dem Ende der Prozesse keine Testergebnisse erhalten hat, werden die Testfälle als Timeout markiert. Als bewährte Methode sollten Sie sicherstellen, dass Ihre Testfälle die folgenden Aktionen ausführen, wenn sie auf eines der Ereignisse stoßen:

1. Beenden Sie die Ausführung der normalen Testlogik.
2. Bereinigen Sie alle temporären Ressourcen, z. B. Testartefakte, auf dem zu testenden Gerät.
3. Melden Sie IDT ein Testergebnis, z. B. einen Testfehler oder einen Fehler.
4. Beenden.

Verwenden Sie den IDT-Kontext

Wenn IDT eine Testsuite ausführt, kann die Testsuite auf einen Datensatz zugreifen, anhand dessen bestimmt werden kann, wie die einzelnen Tests ausgeführt werden. Diese Daten werden als IDT-Kontext bezeichnet. Beispielsweise wird die Benutzerdatenkonfiguration, die von Testläufern in einer `userdata.json` Datei bereitgestellt wird, Testsuiten im IDT-Kontext zur Verfügung gestellt.

Der IDT-Kontext kann als schreibgeschütztes JSON-Dokument betrachtet werden. Testsuiten können mithilfe von Standard-JSON-Datentypen wie Objekten, Arrays, Zahlen usw. Daten aus dem Kontext abrufen und Daten in den Kontext schreiben.

Kontextschema

Der IDT-Kontext verwendet das folgende Format:

```
{
    "config": {
        <config-json-content>
        "timeoutMultiplier": timeout-multiplier
    },
    "device": {
        <device-json-device-element>
    },
    "devicePool": {
        <device-json-pool-element>
    },
    "resource": {
        "devices": [
            {
                <resource-json-device-element>
```

```
                "name": "<resource-name>"
            }
        ]
    },
    "testData": {
        "awsCredentials": {
            "awsAccessKeyId": "<access-key-id>",
            "awsSecretAccessKey": "<secret-access-key>",
            "awsSessionToken": "<session-token>"
        },
        "logFilePath": "/path/to/log/file"
    },
    "userData": {
        <userdata-json-content>
    }
}
```

`config`

Informationen aus der `config.json`Datei. Das `config` Feld enthält außerdem das folgende zusätzliche Feld:

`config.timeoutMultiplier`

Der Multiplikator für jeden Timeout-Wert, der von der Testsuite verwendet wird. Dieser Wert wird vom Test-Runner über die IDT-CLI angegeben. Der Standardwert ist 1.

`device`

Informationen über das Gerät, das für den Testlauf ausgewählt wurde. Diese Information entspricht dem `devices` Array-Element in der `device.json`Datei für das ausgewählte Gerät.

`devicePool`

Informationen über den Gerätepool, der für den Testlauf ausgewählt wurde. Diese Informationen entsprechen dem Element des Gerätepool-Arrays der obersten Ebene, das in der `device.json` Datei für den ausgewählten Gerätepool definiert ist.

`resource`

Informationen zu Ressourcengeräten aus der `resource.json` Datei.

`resource.devices`

Diese Informationen entsprechen dem in der `resource.json` Datei definierten `devices` Array. Jedes `devices` Element enthält das folgende zusätzliche Feld:

`resource.device.name`

Der Name des Ressourcengeräts. Dieser Wert ist auf den `requiredResource.name` Wert in der `test.json` Datei festgelegt.

`testData.awsCredentials`

Die AWS Anmeldeinformationen, die vom Test für die Verbindung mit der AWS Cloud verwendet wurden. Diese Informationen werden aus der `config.json` Datei abgerufen.

`testData.logFilePath`

Der Pfad zur Protokolldatei, in die der Testfall Protokollnachrichten schreibt. Die Testsuite erstellt diese Datei, falls sie nicht existiert.

`userData`

Informationen, die vom Testrunner in der `userdata.json`Datei bereitgestellt wurden.

Greifen Sie auf Daten im Kontext zu

Sie können den Kontext mithilfe der JSONPath Notation aus Ihren JSON-Dateien und aus Ihrer ausführbaren Textdatei mit dem `GetContextValue` und abfragen `GetContextString` APIs. Die Syntax für JSONPath Zeichenketten für den Zugriff auf den IDT-Kontext variiert wie folgt:

- In `suite.json` und `test.json` verwenden `{{`*`query`*`}}` Sie. Das heißt, verwenden Sie nicht das Stammelement`$.`, um Ihren Ausdruck zu starten.
- In`test_orchestrator.yaml`, du benutzt`{{`*`query`*`}}`.

  Wenn Sie die veraltete Zustandsmaschine verwenden, dann verwenden Sie. `state_machine.json` `{{$.`*`query`*`}}`
- In API-Befehlen verwenden Sie je nach Befehl *`query`* oder`{{$.`*`query`*`}}`. Weitere Informationen finden Sie in der Inline-Dokumentation im SDKs.

In der folgenden Tabelle werden die Operatoren in einem typischen JSONPath Ausdruck beschrieben:

Operator	Beschreibung
$	Das Stammelement. Da es sich bei dem Kontextwert der obersten Ebene für IDT um ein

Operator	Beschreibung
	Objekt handelt, verwenden Sie ihn in der Regel, um Ihre `$.` Abfragen zu starten.
`.childName`	Greift auf das untergeordnete Element mit dem Namen eines Objekts `childName` zu. Wenn es auf ein Array angewendet wird, ergibt dies ein neues Array, bei dem dieser Operator auf jedes Element angewendet wird. Beim Elementnamen wird zwischen Groß- und Kleinschreibung unterschieden. Die Abfrage für den Zugriff auf den `awsRegion` Wert im `config` Objekt lautet beispielsweise `$.config.awsRegion` .
`[start:end]`	Filtert Elemente aus einem Array und ruft Elemente ab, die mit dem `start` Index beginnen und bis zum `end` Index aufsteigen, jeweils inklusive.
`[index1, index2, ... , indexN]`	Filtert Elemente aus einem Array und ruft nur Elemente aus den angegebenen Indizes ab.
`[?(expr)]`	Filtert Elemente aus einem Array mithilfe des `expr` Ausdrucks. Dieser Ausdruck muss einen booleschen Wert ergeben.

Verwenden Sie die folgende Syntax, um Filterausdrücke zu erstellen:

```
<jsonpath> | <value> operator <jsonpath> | <value>
```

In dieser Syntax gilt:

- `jsonpath`ist ein JSONPath , das die Standard-JSON-Syntax verwendet.
- `value`ist ein beliebiger benutzerdefinierter Wert, der die Standard-JSON-Syntax verwendet.
- `operator`ist einer der folgenden Operatoren:

- <(Weniger als)
- <=(Weniger als oder gleich)
- ==(Gleich)

  Wenn der Wert JSONPath oder in Ihrem Ausdruck ein Array, ein boolescher Wert oder ein Objektwert ist, dann ist dies der einzige unterstützte binäre Operator, den Sie verwenden können.
- >=(Größer als oder gleich)
- >(Größer als)
- =~(Übereinstimmung mit regulären Ausdrücken). Um diesen Operator in einem Filterausdruck zu verwenden, muss der Wert JSONPath oder auf der linken Seite des Ausdrucks eine Zeichenfolge ergeben, und auf der rechten Seite muss es sich um einen Musterwert handeln, der der RE2Syntax folgt.

Sie können JSONPath Abfragen in der Form {{*query*}} als Platzhalterzeichenfolgen innerhalb der `environmentVariables` Felder `args` und in `test.json` Dateien und innerhalb der `environmentVariables` Felder in `suite.json` Dateien verwenden. IDT führt eine Kontextsuche durch und füllt die Felder mit dem ausgewerteten Wert der Abfrage auf. In der `suite.json` Datei können Sie beispielsweise Platzhalterzeichenfolgen verwenden, um Umgebungsvariablenwerte anzugeben, die sich mit jedem Testfall ändern, und IDT füllt die Umgebungsvariablen mit dem richtigen Wert für jeden Testfall. Wenn Sie jedoch Platzhalterzeichenfolgen in `test.json` und `suite.json` -Dateien verwenden, gelten für Ihre Abfragen die folgenden Überlegungen:

- Sie müssen jedes Vorkommen des `devicePool` Schlüssels in Ihrer Abfrage in Kleinbuchstaben angeben. Das heißt, verwenden Sie `devicepool` stattdessen.
- Für Arrays können Sie nur Zeichenketten-Arrays verwenden. Darüber hinaus verwenden Arrays ein nicht standardmäßiges Format. `item1, item2,...,itemN` Wenn das Array nur ein Element enthält, wird es als serialisiert`item`, sodass es nicht von einem Zeichenkettenfeld zu unterscheiden ist.
- Sie können keine Platzhalter verwenden, um Objekte aus dem Kontext abzurufen.

Aus diesen Gründen empfehlen wir, wann immer möglich, die API für den Zugriff auf den Kontext in Ihrer Testlogik anstelle von Platzhalterzeichenfolgen in `test.json` und `suite.json` -Dateien zu verwenden. In einigen Fällen kann es jedoch praktischer sein, JSONPath Platzhalter zu verwenden, um einzelne Zeichenketten abzurufen, die als Umgebungsvariablen festgelegt werden sollen.

Einstellungen für Testläufer konfigurieren

Um benutzerdefinierte Testsuiten auszuführen, müssen Testläufer ihre Einstellungen auf der Grundlage der Testsuite konfigurieren, die sie ausführen möchten. Die Einstellungen werden auf der Grundlage von Vorlagen für Konfigurationsdateien angegeben, die sich im *<device-tester-extract-location>*`/configs/` Ordner befinden. Falls erforderlich, müssen Testläufer auch AWS Anmeldeinformationen einrichten, die IDT für die Verbindung mit der AWS Cloud verwendet.

Als Testautor müssen Sie diese Dateien konfigurieren, um Ihre Testsuite zu debuggen. Sie müssen den Testläufern Anweisungen geben, damit sie die folgenden Einstellungen nach Bedarf für die Ausführung Ihrer Testsuiten konfigurieren können.

Konfigurieren von device.json

Die `device.json` Datei enthält Informationen über die Geräte, auf denen die Tests ausgeführt werden (z. B. IP-Adresse, Anmeldeinformationen, Betriebssystem und CPU-Architektur).

Testläufer können diese Informationen mithilfe der folgenden `device.json` Vorlagendatei bereitstellen, die sich im *<device-tester-extract-location>*`/configs/` Ordner befindet.

```
[
    {
        "id": "<pool-id>",
        "sku": "<pool-sku>",
        "features": [
            {
                "name": "<feature-name>",
                "value": "<feature-value>",
                "configs": [
                    {
                        "name": "<config-name>",
                        "value": "<config-value>"
                    }
                ],
            }
        ],
        "devices": [
            {
                "id": "<device-id>",
                "connectivity": {
                    "protocol": "ssh | uart | docker",
```

```
                    // ssh
                    "ip": "<ip-address>",
                    "port": <port-number>,
                    "auth": {
                        "method": "pki | password",
                        "credentials": {
                            "user": "<user-name>",
                            // pki
                            "privKeyPath": "/path/to/private/key",

                            // password
                            "password": "<password>",
                        }
                    },

                    // uart
                    "serialPort": "<serial-port>",

                    // docker
                    "containerId": "<container-id>",
                    "containerUser": "<container-user-name>",
                }
            }
        ]
    }
]
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`id`

Eine benutzerdefinierte alphanumerische ID, die eine Sammlung von Geräten, den sogenannten Gerätepool, eindeutig identifiziert. Geräte, die zu einem Pool gehören, müssen über identische Hardware verfügen. Bei der Ausführung einer Reihe von Tests werden Geräte im Pool verwendet, um die Workload zu parallelisieren. Mehrere Geräte werden verwendet, um verschiedene Tests auszuführen.

`sku`

Ein alphanumerischer Wert, durch den das zu testende Gerät eindeutig identifiziert wird. Die SKU wird verwendet, um qualifizierte Geräte nachzuverfolgen.

> **Note**
>
> Wenn du dein Motherboard im AWS Partner Gerätekatalog auflisten möchtest, muss die hier angegebene SKU mit der SKU übereinstimmen, die du bei der Angebotserstellung verwendest.

`features`

Optional. Ein Array, das die Funktionen enthält, die das Gerät unterstützt. Gerätefunktionen sind benutzerdefinierte Werte, die Sie in Ihrer Testsuite konfigurieren. Sie müssen Ihren Testläufern Informationen über die Namen und Werte der Funktionen zur Verfügung stellen, die in die `device.json` Datei aufgenommen werden sollen. Wenn Sie beispielsweise ein Gerät testen möchten, das als MQTT-Server für andere Geräte fungiert, können Sie Ihre Testlogik so konfigurieren, dass bestimmte unterstützte Stufen für ein Feature mit dem Namen `MQTT_QOS` validiert werden. Testläufer geben diesen Feature-Namen an und setzen den Feature-Wert auf die QOS-Stufen, die von ihrem Gerät unterstützt werden. Sie können die bereitgestellten Informationen aus dem IDT-Kontext mit der `devicePool.features` Abfrage oder aus dem Test-Orchestrator-Kontext mit der Abfrage abrufen. `pool.features`

`features.name`

Der Name der Funktion.

`features.value`

Die unterstützten Feature-Werte.

`features.configs`

Konfigurationseinstellungen für die Funktion, falls erforderlich.

`features.config.name`

Der Name der Konfigurationseinstellung.

`features.config.value`

Die unterstützten Einstellungswerte.

`devices`

Eine Reihe von Geräten im Pool, die getestet werden sollen. Es ist mindestens ein Gerät erforderlich.

`devices.id`

Eine benutzerdefinierte eindeutige Kennung für das zu testende Gerät.

`connectivity.protocol`

Das Kommunikationsprotokoll, das für die Kommunikation mit diesem Gerät verwendet wird. Jedes Gerät in einem Pool muss dasselbe Protokoll verwenden.

Derzeit werden nur Werte `uart` für physische Geräte `ssh` und `docker` für Docker-Container unterstützt.

`connectivity.ip`

Die IP-Adresse des zu testenden Geräts.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.port`

Optional. Die Portnummer, die für SSH-Verbindungen verwendet werden soll.

Der Standardwert ist 22.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.auth`

Authentifizierungsinformationen für die Verbindung.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.auth.method`

Die Authentifizierungsmethode, die für den Zugriff über ein bestimmtes Verbindungsprotokoll auf ein Gerät verwendet wird.

Unterstützte Werte sind:

- `pki`
- `password`

`connectivity.auth.credentials`

Die für die Authentifizierung verwendeten Anmeldeinformationen.

`connectivity.auth.credentials.password`

Das Passwort für die Anmeldung am Gerät wird überprüft.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `password` festgelegt ist.

`connectivity.auth.credentials.privKeyPath`

Der vollständige Pfad zum privaten Schlüssel, der für die Anmeldung bei dem zu testenden Gerät verwendet wird.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `pki` festgelegt ist.

`connectivity.auth.credentials.user`

Der Benutzername für die Anmeldung bei dem zu testenden Gerät.

`connectivity.serialPort`

Optional. Die serielle Schnittstelle, an die das Gerät angeschlossen ist.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `uart` festgelegt ist.

`connectivity.containerId`

Die Container-ID oder der Name des getesteten Docker-Containers.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.containerUser`

Optional. Der Name des Benutzers gegenüber dem Benutzer innerhalb des Containers. Der Standardwert ist der im Dockerfile angegebene Benutzer.

Der Standardwert ist 22.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

> **Note**
>
> Um zu überprüfen, ob Testläufer die falsche Geräteverbindung für einen Test konfigurieren, können Sie den Test `pool.Devices[0].Connectivity.Protocol`

aus dem Orchestrator-Kontext abrufen und ihn mit dem erwarteten Wert in einem `Choice` Status vergleichen. Wenn ein falsches Protokoll verwendet wird, drucken Sie eine Nachricht mit dem `LogMessage` Status und wechseln Sie zum `Fail` Status. Alternativ können Sie den Fehlerbehandlungscode verwenden, um einen Testfehler für falsche Gerätetypen zu melden.

(Optional) Konfigurieren Sie userdata.json

Die `userdata.json` Datei enthält alle zusätzlichen Informationen, die für eine Testsuite erforderlich sind, aber nicht in der Datei angegeben sind. `device.json` Das Format dieser Datei hängt von der `userdata_scheme.json`Datei ab, die in der Testsuite definiert ist. Wenn Sie ein Testautor sind, stellen Sie sicher, dass Sie diese Informationen Benutzern zur Verfügung stellen, die die von Ihnen geschriebenen Testsuiten ausführen.

(Optional) Konfigurieren Sie resource.json

Die `resource.json` Datei enthält Informationen zu allen Geräten, die als Ressourcengeräte verwendet werden. Ressourcengeräte sind Geräte, die zum Testen bestimmter Funktionen eines zu testenden Geräts erforderlich sind. Um beispielsweise die Bluetooth-Fähigkeit eines Geräts zu testen, können Sie ein Ressourcengerät verwenden, um zu testen, ob Ihr Gerät erfolgreich eine Verbindung zu dem Gerät herstellen kann. Ressourcengeräte sind optional, und Sie können so viele Ressourcengeräte benötigen, wie Sie benötigen. Als Testautor verwenden Sie die Datei test.json, um die Funktionen der Ressourcengeräte zu definieren, die für einen Test erforderlich sind. Testläufer verwenden dann die `resource.json` Datei, um einen Pool von Ressourcengeräten bereitzustellen, die über die erforderlichen Funktionen verfügen. Stellen Sie sicher, dass Sie diese Informationen Benutzern zur Verfügung stellen, die die von Ihnen geschriebenen Testsuiten ausführen werden.

Testläufer können diese Informationen mithilfe der folgenden `resource.json` Vorlagendatei bereitstellen, die sich im *`<device-tester-extract-location>`*`/configs/` Ordner befindet.

```
[
    {
        "id": "<pool-id>",
        "features": [
            {
                "name": "<feature-name>",
                "version": "<feature-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 und wie diese Tests ausgeführt werden. Wenn die Testsuite keine Funktionen benötigt, ist dieses Feld nicht erforderlich.

`features.name`

Der Name der Funktion.

`features.version`

Die Feature-Version.

`features.jobSlots`

Einstellung, die angibt, wie viele Tests das Gerät gleichzeitig verwenden können. Der Standardwert ist 1.

`devices`

Eine Reihe von Geräten im Pool, die getestet werden sollen. Es ist mindestens ein Gerät erforderlich.

`devices.id`

Eine benutzerdefinierte eindeutige Kennung für das zu testende Gerät.

`connectivity.protocol`

Das Kommunikationsprotokoll, das für die Kommunikation mit diesem Gerät verwendet wird. Jedes Gerät in einem Pool muss dasselbe Protokoll verwenden.

Derzeit werden nur Werte `uart` für physische Geräte `ssh` und `docker` für Docker-Container unterstützt.

`connectivity.ip`

Die IP-Adresse des zu testenden Geräts.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.port`

Optional. Die Portnummer, die für SSH-Verbindungen verwendet werden soll.

Der Standardwert ist 22.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.auth`

Authentifizierungsinformationen für die Verbindung.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.auth.method`

Die Authentifizierungsmethode, die für den Zugriff über ein bestimmtes Verbindungsprotokoll auf ein Gerät verwendet wird.

Unterstützte Werte sind:

- `pki`
- `password`

`connectivity.auth.credentials`

Die für die Authentifizierung verwendeten Anmeldeinformationen.

`connectivity.auth.credentials.password`

Das Passwort für die Anmeldung am Gerät wird überprüft.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `password` festgelegt ist.

`connectivity.auth.credentials.privKeyPath`

Der vollständige Pfad zum privaten Schlüssel, der für die Anmeldung bei dem zu testenden Gerät verwendet wird.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `pki` festgelegt ist.

`connectivity.auth.credentials.user`

Der Benutzername für die Anmeldung bei dem zu testenden Gerät.

`connectivity.serialPort`

Optional. Die serielle Schnittstelle, an die das Gerät angeschlossen ist.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `uart` festgelegt ist.

`connectivity.containerId`

Die Container-ID oder der Name des getesteten Docker-Containers.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.containerUser`

Optional. Der Name des Benutzers gegenüber dem Benutzer innerhalb des Containers. Der Standardwert ist der im Dockerfile angegebene Benutzer.

Der Standardwert ist 22.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

(Optional) Konfigurieren Sie config.json

Die `config.json` Datei enthält Konfigurationsinformationen für IDT. In der Regel müssen Testläufer diese Datei nicht ändern, es sei denn, sie müssen ihre AWS Benutzeranmeldeinformationen für IDT und optional eine AWS Region angeben. Wenn AWS Anmeldeinformationen mit den erforderlichen Berechtigungen bereitgestellt werden, werden Nutzungsmetriken AWS IoT Device Tester erfasst und an diese gesendet. AWS Dies ist eine Opt-in-Funktion, die zur Verbesserung der IDT-Funktionalität verwendet wird. Weitere Informationen finden Sie unter IDT-Nutzungsmetriken.

Testläufer können ihre AWS Anmeldeinformationen auf eine der folgenden Arten konfigurieren:

- Anmeldeinformationsdatei

  IDT verwendet die gleiche Anmeldeinformationsdatei wie das AWS CLI. Weitere Informationen finden Sie unter Konfigurations- und Anmeldeinformationsdateien.

  Der Speicherort der Datei mit den Anmeldeinformationen variiert je nach verwendetem Betriebssystem:

  - macOS Linux: `~/.aws/credentials`
  - Windows: `C:\Users\UserName\.aws\credentials`
- Umgebungsvariablen

  Umgebungsvariablen sind Variablen, die vom Betriebssystem gepflegt und von Systembefehlen verwendet werden. Variablen, die während einer SSH-Sitzung definiert wurden, sind nach dem Schließen dieser Sitzung nicht verfügbar. IDT kann die `AWS_SECRET_ACCESS_KEY` Umgebungsvariablen `AWS_ACCESS_KEY_ID` und zum Speichern von Anmeldeinformationen verwenden AWS

  Um diese Variablen auf Linux, macOS oder Unix festzulegen, verwenden Sie export:

```
export AWS_ACCESS_KEY_ID=<your_access_key_id>
export AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

In Windows können Sie die Variablen mit set festlegen:

```
set AWS_ACCESS_KEY_ID=<your_access_key_id>
set AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

Um AWS Anmeldeinformationen für IDT zu konfigurieren, bearbeiten Testläufer den `auth` Abschnitt in der `config.json` Datei, die sich im *<device-tester-extract-location>*`/configs/` Ordner befindet.

```
{
    "log": {
        "location": "logs"
    },
    "configFiles": {
        "root": "configs",
        "device": "configs/device.json"
    },
    "testPath": "tests",
    "reportPath": "results",
    "awsRegion": "<region>",
    "auth": {
        "method": "file | environment",
        "credentials": {
            "profile": "<profile-name>"
        }
    }
}
]
```

Nachfolgend sind alle Pflichtfelder beschrieben:

> **Note**
>
> Alle Pfade in dieser Datei sind relativ zu definiert. *<device-tester-extract-location>*

`log.location`

Der Pfad zum Protokollordner in der*<device-tester-extract-location>*.

`configFiles.root`

Der Pfad zu dem Ordner, der die Konfigurationsdateien enthält.

`configFiles.device`

Der Pfad zur `device.json` Datei.

`testPath`

Der Pfad zu dem Ordner, der Testsuiten enthält.

`reportPath`

Der Pfad zu dem Ordner, der Testergebnisse enthält, nachdem IDT eine Testsuite ausgeführt hat.

`awsRegion`

Optional. Die AWS Region, die die Testsuiten verwenden werden. Wenn nicht festgelegt, verwenden Testsuiten die in jeder Testsuite angegebene Standardregion.

`auth.method`

Die Methode, die IDT zum Abrufen von AWS Anmeldeinformationen verwendet. Unterstützte Werte sind `file` das Abrufen von Anmeldeinformationen aus einer Anmeldeinformationsdatei und `environment` das Abrufen von Anmeldeinformationen mithilfe von Umgebungsvariablen.

`auth.credentials.profile`

Das zu verwendende Anmeldeinformationsprofil aus der Anmeldeinformationsdatei. Diese Eigenschaft gilt nur, wenn `auth.method` auf `file` festgelegt ist.

Debuggen und Ausführen benutzerdefinierter Testsuiten

Nachdem die erforderliche Konfiguration festgelegt wurde, kann IDT Ihre Testsuite ausführen. Die Laufzeit der vollständigen Testsuite hängt von der Hardware und der Zusammensetzung der Testsuite ab. Als Referenz: Es dauert ungefähr 30 Minuten, bis die vollständige AWS IoT Greengrass Qualifizierungstestsuite auf einem Raspberry Pi 3B abgeschlossen ist.

Während Sie Ihre Testsuite schreiben, können Sie IDT verwenden, um die Testsuite im Debug-Modus auszuführen, um Ihren Code vor der Ausführung zu überprüfen oder ihn Testläufern zur Verfügung zu stellen.

Führen Sie IDT im Debug-Modus aus

Da Testsuiten auf IDT angewiesen sind, um mit Geräten zu interagieren, den Kontext bereitzustellen und Ergebnisse zu erhalten, können Sie Ihre Testsuiten nicht einfach ohne IDT-Interaktion in einer IDE debuggen. Zu diesem Zweck stellt die IDT-CLI den `debug-test-suite` Befehl bereit, mit dem Sie IDT im Debug-Modus ausführen können. Führen Sie den folgenden Befehl aus, um die verfügbaren Optionen für anzuzeigen: `debug-test-suite`

```
devicetester_[linux | mac | win_x86-64] debug-test-suite -h
```

Wenn Sie IDT im Debug-Modus ausführen, startet IDT weder die Testsuite noch den Test-Orchestrator. Stattdessen interagiert IDT mit Ihrer IDE, um auf Anfragen von der in der IDE ausgeführten Testsuite zu antworten und die Protokolle auf der Konsole auszudrucken. IDT hat kein Timeout und wartet mit dem Beenden, bis es manuell unterbrochen wird. Im Debug-Modus führt IDT auch den Test-Orchestrator nicht aus und generiert keine Berichtsdateien. Um Ihre Testsuite zu debuggen, müssen Sie Ihre IDE verwenden, um einige Informationen bereitzustellen, die IDT normalerweise aus den JSON-Konfigurationsdateien bezieht. Stellen Sie sicher, dass Sie die folgenden Informationen angeben:

- Umgebungsvariablen und Argumente für jeden Test. IDT liest diese Informationen nicht von `test.json` oder`suite.json`.
- Argumente zur Auswahl von Ressourcengeräten. IDT liest diese Informationen nicht aus`test.json`.

Gehen Sie wie folgt vor, um Ihre Testsuiten zu debuggen:

1. Erstellen Sie die Einstellungskonfigurationsdateien, die für die Ausführung der Testsuite erforderlich sind. Wenn Ihre Testsuite beispielsweise die`device.json`, und erfordert`resource.json`, stellen Sie sicher`user data.json`, dass Sie sie alle nach Bedarf konfigurieren.
2. Führen Sie den folgenden Befehl aus, um IDT in den Debug-Modus zu versetzen und alle Geräte auszuwählen, die für die Ausführung des Tests erforderlich sind.

   ```
   devicetester_[linux | mac | win_x86-64] debug-test-suite [options]
   ```

Nachdem Sie diesen Befehl ausgeführt haben, wartet IDT auf Anfragen von der Testsuite und beantwortet sie dann. IDT generiert auch die Umgebungsvariablen, die für den Fallprozess für das IDT Client SDK erforderlich sind.

3. Verwenden Sie in Ihrer IDE die `debug` Konfiguration `run` oder, um Folgendes zu tun:

   a. Legen Sie die Werte der von IDT generierten Umgebungsvariablen fest.

   b. Legen Sie den Wert aller Umgebungsvariablen oder Argumente fest, die Sie in Ihrer `test.json` AND-Datei angegeben haben. `suite.json`

   c. Legen Sie nach Bedarf Haltepunkte fest.

4. Führen Sie die Testsuite in Ihrer IDE aus.

   Sie können die Testsuite so oft wie nötig debuggen und erneut ausführen. IDT tritt im Debug-Modus nicht auf.

5. Nachdem Sie das Debuggen abgeschlossen haben, unterbrechen Sie IDT, um den Debug-Modus zu verlassen.

IDT-CLI-Befehle zum Ausführen von Tests

Im folgenden Abschnitt werden die IDT-CLI-Befehle beschrieben:

IDT v4.0.0

`help`

Listet Informationen über den angegebenen Befehl auf.

`list-groups`

Listet die Gruppen in der jeweiligen Testsuite auf.

`list-suites`

Listet die verfügbaren Testsuites auf.

`list-supported-products`

Listet die unterstützten Produkte für Ihre Version von IDT auf, in diesem Fall AWS IoT Greengrass Versionen, und die Versionen der AWS IoT Greengrass Qualifizierungstestsuite, die für die aktuelle IDT-Version verfügbar sind.

`list-test-cases`

Listet die Testfälle in einer bestimmten Testgruppe auf. Die folgende Option wird unterstützt:

- `group-id`. Die Testgruppe, nach der gesucht werden soll. Diese Option ist erforderlich und muss eine einzelne Gruppe angeben.

`run-suite`

Führt eine Reihe von Tests in einem Pool von Geräten aus. Im Folgenden sind einige häufig verwendete Optionen aufgeführt:

- `suite-id`. Die auszuführende Version der Testsuite. Wenn nicht angegeben, verwendet IDT die neueste Version im `tests`-Ordner.
- `group-id`. Die auszuführenden Testgruppen als 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. Testläufer müssen einen Pool angeben, wenn in Ihrer `device.json` Datei mehrere Gerätepools definiert sind.
- `timeout-multiplier`. Konfiguriert IDT so, dass das in der `test.json` Datei angegebene Timeout für die Testausführung für einen Test mit einem benutzerdefinierten Multiplikator geändert wird.
- `stop-on-first-failure`. Konfiguriert IDT so, dass die Ausführung beim ersten Fehler gestoppt wird. Diese Option sollte mit `group-id` verwendet werden, um die angegebenen Testgruppen zu debuggen.
- `userdata`. Legt die Datei fest, die Benutzerdateninformationen enthält, die zum Ausführen der Testsuite erforderlich sind. Dies ist nur erforderlich, wenn `userdataRequired` es in der `suite.json` Datei für die Testsuite auf true gesetzt ist.

Weitere Informationen zu `run-suite`-Optionen erhalten Sie mit der `help`-Option:

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

`debug-test-suite`

Führen Sie die Testsuite im Debug-Modus aus. Weitere Informationen finden Sie unter Führen Sie IDT im Debug-Modus aus.

Überprüfen Sie die IDT-Testergebnisse und -protokolle

In diesem Abschnitt wird das Format beschrieben, in dem IDT Konsolenprotokolle und Testberichte generiert.

Nachrichtenformat der Konsole

AWS IoT Device Tester verwendet ein Standardformat für das Drucken von Nachrichten auf der Konsole, wenn eine Testsuite gestartet wird. Der folgende Auszug zeigt ein Beispiel für eine von IDT generierte Konsolennachricht.

```
time="2000-01-02T03:04:05-07:00" level=info msg=Using suite: MyTestSuite_1.0.0
executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

Die meisten Konsolennachrichten bestehen aus den folgenden Feldern:

`time`

Ein vollständiger ISO 8601-Zeitstempel für das protokollierte Ereignis.

`level`

Die Nachrichtenebene für das protokollierte Ereignis. In der Regel ist die Ebene der protokollierten Nachricht eine von `info`, `warn`, oder`error`. IDT gibt eine `fatal` `panic` OR-Nachricht aus, wenn es auf ein erwartetes Ereignis trifft, das dazu führt, dass es vorzeitig beendet wird.

`msg`

Die protokollierte Nachricht.

`executionId`

Eine eindeutige ID-Zeichenfolge für den aktuellen IDT-Prozess. Diese ID wird verwendet, um zwischen einzelnen IDT-Läufen zu unterscheiden.

Von einer Testsuite generierte Konsolennachrichten enthalten zusätzliche Informationen über das zu testende Gerät und die Testsuite, Testgruppe und Testfälle, die IDT ausführt. Der folgende Auszug zeigt ein Beispiel für eine Konsolennachricht, die aus einer Testsuite generiert wurde.

```
time="2000-01-02T03:04:05-07:00" level=info msg=Hello world! suiteId=MyTestSuite
groupId=myTestGroup testCaseId=myTestCase deviceId=my-device
executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

Der für die Testsuite spezifische Teil der Konsolennachricht enthält die folgenden Felder:

`suiteId`

Der Name der Testsuite, die gerade läuft.

`groupId`

Die ID der Testgruppe, die gerade läuft.

`testCaseId`

Die ID des aktuell laufenden Testfalls.

`deviceId`

Eine ID des zu testenden Geräts, das der aktuelle Testfall verwendet.

Um eine Testzusammenfassung auf der Konsole zu drucken, wenn ein IDT die Ausführung eines Tests abgeschlossen hat, müssen Sie in Ihrem Test-Orchestrator einen `ReportStatus` angeben. Die Testzusammenfassung enthält Informationen über die Testsuite, die Testergebnisse für jede Gruppe, die ausgeführt wurde, und die Speicherorte der generierten Protokolle und Berichtsdateien. Das folgende Beispiel zeigt eine Meldung mit einer Testzusammenfassung.

```
========== Test Summary ==========
Execution Time:     5m00s
Tests Completed:    4
Tests Passed:       3
Tests Failed:       1
Tests Skipped:      0
----------------------------------
Test Groups:
    GroupA:         PASSED
    GroupB:         FAILED
----------------------------------
Failed Tests:
    Group Name: GroupB
        Test Name: TestB1
            Reason: Something bad happened
----------------------------------
Path to IoT Device Tester Report: /path/to/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/logs
Path to Aggregated JUnit Report: /path/to/MyTestSuite_Report.xml
```

AWS IoT Device Tester Berichtsschema

`awsiotdevicetester_report.xml`ist ein signierter Bericht, der die folgenden Informationen enthält:

- Die IDT-Version.
- Die Version der Testsuite.
- Die Berichtssignatur und der Schlüssel, die zum Signieren des Berichts verwendet wurden.
- Die Geräte-SKU und der Name des Gerätepools, die in der `device.json` Datei angegeben sind.
- Die Produktversion und die getesteten Gerätefunktionen.
- Die aggregierte Zusammenfassung der Testergebnisse. Diese Informationen entsprechen denen, die in der *`suite-name`*`_report.xml` Datei enthalten sind.

```
<apnreport>
    <awsiotdevicetesterversion>idt-version</awsiotdevicetesterversion>
    <testsuiteversion>test-suite-version</testsuiteversion>
    <signature>signature</signature>
    <keyname>keyname</keyname>
    <session>
        <testsession>execution-id</testsession>
        <starttime>start-time</starttime>
        <endtime>end-time</endtime>
    </session>
    <awsproduct>
        <name>product-name</name>
        <version>product-version</version>
        <features>
            <feature name="<feature-name>" value="supported | not-supported | <feature-
value>" type="optional | required"/>
        </features>
    </awsproduct>
    <device>
        <sku>device-sku</sku>
        <name>device-name</name>
        <features>
            <feature name="<feature-name>" value="<feature-value>"/>
        </features>
        <executionMethod>ssh | uart | docker</executionMethod>
    </device>
    <devenvironment>
```

```
        <os name="<os-name>"/>
    </devenvironment>
    <report>
        <suite-name-report-contents>
    </report>
</apnreport>
```

Die Datei `awsiotdevicetester_report.xml` enthält ein `<awsproduct>`-Tag mit Informationen zum getesteten Produkt und den Produktfunktionen, die nach einer Reihe von Tests validiert wurden.

Im **`<awsproduct>`**-Tag verwendete Attribute

`name`

Der Name des getesteten Produkts.

`version`

Die Version des getesteten Produkts.

`features`

Die validierten Funktionen Als markierte Funktionen `required` sind erforderlich, damit die Testsuite das Gerät validieren kann. Der folgende Ausschnitt zeigt, wie diese Informationen in der Datei `awsiotdevicetester_report.xml` angezeigt werden.

```
<feature name="ssh" value="supported" type="required"></feature>
```

Als markierte Funktionen `optional` sind für die Validierung nicht erforderlich. Die folgenden Codeausschnitte zeigen optionale Funktionen.

```
<feature name="hsi" value="supported" type="optional"></feature>

<feature name="mqtt" value="not-supported" type="optional"></feature>
```

Berichtsschema der Testsuite

Der *`suite-name`*`_Result.xml` Bericht ist im JUnit XML-Format. Sie können ihn in Continuous Integration and Deployment-Plattformen wie Jenkins, Bamboo usw. integrieren. Der Bericht enthält eine Zusammenfassung der Testergebnisse.

```
<testsuites name="<suite-name> results" time="<run-duration>" tests="<number-of-test>"
 failures="<number-of-tests>" skipped="<number-of-tests>" errors="<number-of-tests>"
 disabled="0">
    <testsuite name="<test-group-id>" package="" tests="<number-of-tests>"
 failures="<number-of-tests>" skipped="<number-of-tests>" errors="<number-of-tests>"
 disabled="0">
        <!--success-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>"/>
        <!--failure-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>">
            <failure type="<failure-type>">
                reason
            </failure>
        </testcase>
        <!--skipped-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>">
            <skipped>
                reason
            </skipped>
        </testcase>
        <!--error-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>">
            <error>
                reason
            </error>
        </testcase>
    </testsuite>
</testsuites>
```

Im Berichtsabschnitt sowohl im `awsiotdevicetester_report.xml` als auch im Abschnitt *suite-name*`_report.xml` werden die durchgeführten Tests und die Ergebnisse aufgeführt.

Im ersten XML-Tag `<testsuites>` ist die Zusammenfassung der Testausführung enthalten. Zum Beispiel:

```
<testsuites name="MyTestSuite results" time="2299" tests="28" failures="0" errors="0"
 disabled="0">
```

Im **<testsuites>**-Tag verwendete Attribute

`name`

Name der Testsuite

`time`

Die Zeit in Sekunden, die zum Ausführen der Testsuite benötigt wurde.

`tests`

Die Anzahl der ausgeführten Tests.

`failures`

Die Anzahl der ausgeführten Tests, die den Test nicht bestanden haben

`errors`

Die Anzahl der Tests, die IDT nicht ausführen konnte.

`disabled`

Dieses Attribut wird nicht verwendet und kann ignoriert werden.

Falls bei Tests Fehler auftreten, können Sie den fehlgeschlagenen Test identifizieren, indem Sie die XML-Tags von `<testsuites>` überprüfen. Die XML-Tags von `<testsuite>` im `<testsuites>`-Tag zeigen die Ergebniszusammenfassung eines Tests für eine Testgruppe. Zum Beispiel:

```
<testsuite name="combination" package="" tests="1" failures="0" time="161" disabled="0"
 errors="0" skipped="0">
```

Das Format ähnelt dem `<testsuites>`-Tag, weist aber das Attribut `skipped` auf, das nicht verwendet wird und ignoriert werden kann. Innerhalb der einzelnen `<testsuite>`-XML-Tags befinden sich `<testcase>`-Tags für alle ausgeführten Tests einer Testgruppe. Zum Beispiel:

```
<testcase classname="Security Test" name="IP Change Tests" attempts="1"></testcase>>
```

Im **<testcase>**-Tag verwendete Attribute

`name`

Der Name des Tests

`attempts`

Gibt an, wie oft IDT den Testfall ausgeführt hat.

Wenn ein Testfall fehlschlägt oder ein Fehler auftritt, werden `<failure>`- oder `<error>`-Tags hinzugefügt, um das `<testcase>`-Tag mit Informationen für die Fehlerbehebung zu versehen. Zum Beispiel:

```
<testcase classname="mcu.Full_MQTT" name="MQTT_TestCase" attempts="1">
 <failure type="Failure">Reason for the test failure</failure>
 <error>Reason for the test execution error</error>
</testcase>
```

IDT-Nutzungsmetriken

Wenn Sie AWS Anmeldeinformationen mit den erforderlichen Berechtigungen angeben, 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 *<device-tester-extract-location>* Ordner
- Die Anzahl der im Gerätepool konfigurierten Geräte
- Testfallnamen und Laufzeiten
- Informationen zu den Testergebnissen, z. B. ob Tests bestanden oder fehlgeschlagen sind, ob Fehler aufgetreten sind oder ob sie übersprungen wurden
- Getestete Produktmerkmale
- Verhalten beim Beenden von IDT, z. B. unerwartetes oder vorzeitiges Beenden

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 sich für die Erfassung von Nutzungsmetriken entscheiden.

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 Anmeldedaten

Wenn Sie noch keine haben AWS-Konto, müssen Sie eine erstellen. Wenn Sie bereits über eine verfügen AWS-Konto, müssen Sie lediglich die erforderlichen Berechtigungen für Ihr Konto konfigurieren, damit IDT in Ihrem Namen Nutzungsdaten AWS an diese senden kann.

Schritt 1: Erstellen 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/die Anmeldung.
2. Folgen Sie den Online-Anweisungen.

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

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

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 AWS CLI die Konfiguration für die Verwendung AWS IAM Identity Center im AWS Command Line Interface Benutzerhandbuch vornehmen.
In IAM (Nicht empfohlen)	Verwendung von langfristigen Anmeldeinformationen für den Zugriff auf AWS.	Folgen Sie den Anleitungen unter IAM-Benutzer für den Notfallzugriff erstellen im IAM-Benutzerhandbuch.	Sie konfigurieren den programmgesteuerten Zugriff unter Verwendung der Informationen unter Verwalten der Zugriffsschlüssel für IAM-Benutzer im IAM-Benutzerhandbuch.

Schritt 2: Konfigurieren von Berechtigungen für IDT

In diesem Schritt konfigurieren Sie die Berechtigungen, die IDT zum Ausführen von Tests und zum Sammeln von IDT-Nutzungsdaten verwendet. Sie können das AWS-Managementkonsole oder AWS Command Line Interface (AWS CLI) verwenden, um eine IAM-Richtlinie und einen Benutzer für IDT zu erstellen und dann Richtlinien an den Benutzer anzuhängen.

- So konfigurieren Sie Berechtigungen für IDT (Konsole):
- So konfigurieren Sie Berechtigungen für IDT (AWS CLI):

So konfigurieren Sie Berechtigungen für IDT (Konsole)

Gehen Sie folgendermaßen vor, um mithilfe der Konsole Berechtigungen für IDT für AWS IoT Greengrass zu konfigurieren.

1. Melden Sie sich bei der IAM-Konsole an.
2. Erstellen Sie eine vom Kunden verwaltete Richtlinie, die Berechtigungen zum Erstellen von Rollen mit bestimmten Berechtigungen erteilt.

   a. Wählen Sie im Navigationsbereich Richtlinien und dann Richtlinie erstellen.

   b. Ersetzen Sie auf der Registerkarte JSON den Platzhalterinhalt durch die folgende Richtlinie.

      JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot-device-tester:SendMetrics"
            ],
            "Resource": "*"
        }
    ]
}
```

   c. Wählen Sie 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 IDTUsageIAMPermissionsMetrics-Richtlinie, die Sie im vorherigen Schritt erstellt haben. Markieren Sie das Kontrollkästchen.

   c. Wählen Sie Weiter: Tags aus.

   d. Wählen Sie Next: Review (Weiter: Überprüfen), um eine Zusammenfassung Ihrer Auswahlmöglichkeiten anzuzeigen.

   e. Wählen Sie Create user (Benutzer erstellen) aus.

   f. Um die Zugriffsschlüssel des Benutzers (Zugriffsschlüssel IDs und geheime Zugriffsschlüssel) anzuzeigen, wählen Sie neben dem Passwort und dem Zugriffsschlüssel die Option Anzeigen aus. Zum Speichern der Zugriffsschlüssel wählen Sie Download .csv (CSV-Datei herunterladen) aus und speichern die Datei an einem sicheren Speicherort. Sie verwenden diese Informationen später, um Ihre AWS Anmeldeinformationsdatei zu konfigurieren.

So konfigurieren Sie Berechtigungen für IDT (AWS CLI)

Gehen Sie wie folgt vor, AWS CLI um Berechtigungen für IDT für AWS IoT Greengrass zu konfigurieren.

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. *`user-name`*Ersetzen Sie es durch Ihren IAM-Benutzernamen und *`<account-id>`* im Befehl durch die ID Ihres. AWS-Konto

   ```
   aws iam attach-user-policy --user-name user-name --policy-arn
    arn:aws:iam::<account-id>:policy/IDTGreengrassIAMPermissions
   ```

4. Erstellen Sie einen geheimen Zugriffsschlüssel für den Benutzer.

   ```
   aws iam create-access-key --user-name user-name
   ```

   Speichern Sie die Ausgabe an einem sicheren Ort. Sie verwenden diese Informationen später, um Ihre AWS Anmeldeinformationsdatei zu konfigurieren.

Geben Sie die AWS Anmeldeinformationen für IDT ein

Gehen Sie wie folgt vor, damit IDT auf Ihre AWS Anmeldeinformationen zugreifen und Messwerte an AWS senden kann:

1. Speichern Sie die AWS Anmeldeinformationen für Ihren IAM-Benutzer als Umgebungsvariablen oder in einer Anmeldeinformationsdatei:

   a. Führen Sie 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 Sie config.json.

Problembehandlung IDT für V2 AWS IoT Greengrass

IDT für AWS IoT Greengrass V2 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 Verzeichnis der IDT-Testprotokolle 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 Ausführung des Tests auf die Konsole geschrieben wurden. Die Zusammenfassung der Ergebnisse 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. Für Tests, die Greengrass-Komponenten bereitstellen, wird die Testfall-Protokolldatei aufgerufen`greengrass-test-run.log` .
test-group-id `/`*test-case-id* `/greengrass.log`	Detaillierte Protokolle für die AWS IoT Greengrass Core-Software. IDT kopiert diese Datei von dem zu testenden Gerät, wenn es Tests ausführt, bei denen AWS IoT Greengrass Core-Software auf dem Gerät installiert wird. Weitere Informationen zu den Meldungen in

Datei	Beschreibung
	dieser Protokolldatei finden Sie 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 für AWS IoT Greengrass V2-Fehler beheben

Bevor Sie IDT for ausführen AWS IoT Greengrass, sollten Sie die richtigen Konfigurationsdateien einrichten. 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 ist vorhanden, Fehler
- Die Anmeldeinformationen konnten nicht gelesen werden
- Guide-Fehler mit Greengrass PreInstalled
- Ungültige Signaturausnahme
- Qualifizierungsfehler beim maschinellen Lernen
- Fehlgeschlagene Bereitstellungen von 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 in der angezeigt`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 Ihr `device.json` und die richtigen Informationen `userdata.json` enthalten, die für Ihre Testsuite erforderlich sind. Informationen zur Konfiguration, die für die AWS IoT Greengrass Qualifizierung erforderlich ist, finden Sie unterKonfigurieren Sie die IDT-Einstellungen, um die AWS IoT Greengrass Qualification Suite auszuführen.

Konfliktfehler

Möglicherweise wird der folgende Fehler angezeigt, wenn Sie die AWS IoT Greengrass Qualification Suite gleichzeitig auf mehr als einem Gerät ausführen.

```
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 für die AWS IoT Greengrass Qualification Suite noch nicht unterstützt. 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 verweist direkt aus Ihrer `device.json` Datei auf den Poolnamen.
- Stellen Sie sicher, dass die Geräte in Ihrem Pool über die richtigen Konfigurationsparameter verfügen.

Das Docker-Qualifikationsbild ist vorhanden, Fehler

Die Qualifizierungstests für Docker Application Manager verwenden das `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 bereits heruntergeladen und den `amazon/amazon-ec2-metadata-mock` Container auf Ihrem Gerät ausgeführt haben, stellen Sie sicher, dass Sie dieses Image von dem zu testenden Gerät entfernen, bevor Sie die Qualifizierungstests ausführen.

Die Anmeldeinformationen konnten nicht gelesen werden

Beim Testen von Windows-Geräten kann der `Failed to read credential` Fehler in der `greengrass.log` Datei auftreten, wenn der Benutzer, den Sie für die Verbindung mit dem zu testenden Gerät verwenden, 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 Greengrass PreInstalled

Wenn Sie beim Ausführen von IDT mit PreInstalled Greengrass auf den Fehler `Guice` oder stoßen`ErrorInCustomProvider`, überprüfen Sie, ob die Datei den

`InstalledDirRootOnDevice` Greengrass-Installationsordner `userdata.json` enthält. IDT sucht nach der Datei unter. `effectiveConfig.yaml` `<InstallationDirRootOnDevice>/config/effectiveConfig.yaml`

Weitere Informationen finden Sie unter Konfigurieren Sie Benutzeranmeldeinformationen für Windows-Geräte.

Ungültige Signaturausnahme

Wenn Sie Lambda-Qualifizierungstests ausführen, kann der `invalidsignatureexception` Fehler auftreten, wenn auf Ihrem IDT-Hostcomputer Netzwerkzugriffsprobleme auftreten. Setzen Sie Ihren Router zurück und führen Sie die Tests erneut aus.

Qualifizierungsfehler beim maschinellen Lernen

Wenn Sie Qualifizierungstests für maschinelles Lernen (ML) durchführen, kann es zu Qualifizierungsfehlern kommen, wenn Ihr Gerät die Anforderungen für die Bereitstellung der AWS bereitgestellten ML-Komponenten nicht erfüllt. Gehen Sie wie folgt vor, um ML-Qualifizierungsfehler 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.
- Fügen Sie der `test.json` Datei das `-Dgg.persist=installed.software` Argument für den fehlgeschlagenen Testfall hinzu. Die `test.json` Datei befindet sich im *<device-tester-extract-location>*`/tests/GGV2Q_`*version* `directory.`

Fehlgeschlagene Bereitstellungen von Open Test Framework (OTF)

Wenn die Bereitstellung durch OTF-Tests nicht abgeschlossen werden kann, kann dies wahrscheinlich an den für den übergeordneten Ordner von `TempResourcesDirOnDevice` und `InstallationDirRootOnDevice` festgelegten Berechtigungen liegen. Führen Sie den folgenden Befehl aus, um die Berechtigungen für diesen Ordner korrekt festzulegen. *folder-name*Ersetzen Sie ihn 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 durchführen. JSONLint

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 der AWS IoT Greengrass V2 Qualification Suite (GGV2Q) zur Erstellung von Qualifizierungsberichten, die Sie einreichen können, AWS Partner Network um Ihre Geräte in den AWS Partner Gerätekatalog aufzunehmen. Frühere Versionen der Qualification Suite generieren keine Qualifikationsberichte.

Wenn Sie Fragen zu den Support-Richtlinien haben, wenden Sie sich an AWS Support.

Fehler aufgrund fehlender erforderlicher Parameter

Wenn IDT neue Funktionen hinzufügt, kann es zu Änderungen an den Konfigurationsdateien kommen. Bei Verwendung einer alten Konfigurationsdatei kann Ihre Konfiguration beschädigt werden. In diesem Fall listet die Datei *<test_case_id>*`.log` unter `/results/`*<execution-id>*`/logs` ausdrücklich alle fehlenden Parameter auf. IDT 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-Menü die Systemeinstellungen.
2. Wählen Sie Sicherheit und Datenschutz und dann auf der Registerkarte Allgemein das Schlosssymbol, um Änderungen an den Sicherheitseinstellungen vorzunehmen.
3. Suchen Sie im Falle einer Blockierung `devicetester_mac_x86-64` nach der Nachricht `"devicetester_mac_x86-64" was blocked from use because it is not from an identified developer.` und wählen Sie „Trotzdem zulassen“.
4. Setzen Sie den IDT-Test 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-Meldungen werden oben in dieser Protokolldatei angezeigt, da die Verbindung zu einem getesteten Gerät eine der ersten Operationen ist, die IDT ausführt.

Die meisten Windows-Konfigurationen verwenden die TTy Pu-Terminal-Anwendung, 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 Ihrer `device.json` Datei konfigurieren, verwenden Sie PEM-Dateien. Wenn Sie eine PPK-Datei verwenden, kann IDT keine SSH-Verbindung mit dem AWS IoT Greengrass Gerät herstellen und keine Tests ausführen.

Ab IDT v4.4.0 wird in der Protokolldatei möglicherweise der folgende Fehler angezeigt, wenn Sie SFTP auf Ihrem zu testenden Gerät nicht aktiviert haben.

```
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 angezeigt.

```
Failed to upload data to S3
```

Dieser Fehler kann auftreten, wenn Stream Manager die AWS Anmeldeinformationen in der `~/root/.aws/credentials` Datei auf Ihrem Gerät verwendet, anstatt die Umgebungsinformationen 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`. Zum Beispiel:

```
./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 `Access denied` Fehler enthalten, verfügt Ihr Greengrass-Installationsordner möglicherweise nicht über die richtigen Dateiberechtigungen. Führen Sie den folgenden Befehl für jeden Ordner im Installationspfad aus, den Sie in Ihrer Datei definiert haben. `userdata.json`

  ```
  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` es auf dem zu testenden Gerät installiert ist.
  - Wenn Ihre `userdata.json` Datei den `GreengrassCliVersion` Konfigurationsparameter enthält, entfernen Sie 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 (`idt-gg2-lambda-function-idt-<resource-id>.log`) erhalten, die besagt`Error: Could not find or load main class com.amazonaws.greengrass.runtime.LambdaRuntime.`, gehen Sie wie folgt vor:
  - Überprüfen Sie, für welchen Ordner `InstallationDirRootOnDevice` in der Datei verwendet wurde. `userdata.json`
  - Stellen Sie sicher, dass die richtigen Benutzerberechtigungen auf Ihrem Gerät eingerichtet sind. Weitere Informationen finden Sie unter Benutzerberechtigungen auf Ihrem Gerät konfigurieren.

Fehler bei der Versionsprüfung

IDT gibt den folgenden Fehler aus, wenn die AWS Benutzeranmeldeinformationen für den IDT-Benutzer nicht über die erforderlichen IAM-Berechtigungen verfügen.

```
Failed to check version compatibility
```

Der AWS Benutzer, der nicht über die erforderlichen IAM-Berechtigungen verfügt.

Unterstützungsrichtlinie AWS IoT Device Tester für AWS IoT Greengrass

AWS IoT Device Tester for AWS IoT Greengrass ist ein Tool zur Testautomatisierung, mit dem Sie Ihre AWS IoT Greengrass Geräte für die Aufnahme in den AWS Partner Gerätekatalog validieren und qualifizieren können. Wir empfehlen Ihnen, die neueste Version von AWS IoT Greengrass und AWS IoT Device Tester zum Testen oder Qualifizieren Ihrer Geräte zu verwenden.

Für jede unterstützte Version von AWS IoT Device Tester ist mindestens eine Version von verfügbar AWS IoT Greengrass. Informationen zu unterstützten Versionen von AWS IoT Greengrass finden Sie unter Greengrass Nucleus-Versionen. Informationen zu unterstützten Versionen von finden Sie AWS IoT Device Tester unterUnterstützte Versionen von AWS IoT Device Tester für AWS IoT Greengrass V2.

Sie können auch jede der unterstützten Versionen von AWS IoT Greengrass und verwenden, um Ihre Geräte AWS IoT Device Tester zu testen oder zu qualifizieren. Sie können zwar weiterhin nicht unterstützte Versionen von verwenden AWS IoT Device Tester, für diese Versionen gibt es jedoch keine Bugfixes oder Updates. Wenn Sie Fragen zu den Support-Richtlinien haben, wenden Sie sich an AWS Support.

IoT-Lösungen auf Basis von Greengrass

Everyware von Eurotech GreenEdge befindet sich in einer Vorschauversion für AWS IoT Greengrass und kann sich ändern. Diese Lösung wird nicht unterstützt von. AWS Bei Problemen mit diesem Gerät müssen Sie sich an Eurotech wenden.

AWS IoT Greengrass bietet Lösungen von Partnern, um Ihre Erfahrung mit der Installation von Greengrass zu optimieren. Im Folgenden finden Sie eine Lösung, die AWS in Zusammenarbeit mit Eurotech angeboten wurde. Bei dieser Lösung sind AWS IoT Greengrass Core Edge Runtime und zusätzliche Funktionen vorinstalliert.

Eurotech

AWS hat sich mit Eurotech zusammengetan, um eine IoT-Lösung für Kunden anzubieten, die nach einem Gerät suchen, auf dem die AWS IoT Greengrass Core-Software vorinstalliert ist. Everyware von Eurotech GreenEdge ist eine IoT-Edge-Software, die von vorkonfiguriert und vorqualifiziert ist. AWS Diese Lösung vereint die Funktionen von Greengrass und dem Eurotech Everyware Software Framework (ESF), um Kunden umfassende Southbound-Konnektivität über Protokolladapter wie Modbus, OPC-UA und mehr zu bieten. Client/Server, S7, TwinCAT, J1939, DNP3 Master/ Outstation Mit dieser Lösung können Sie auch Daten an alle AWS Northbound-Dienste (wie AWS IoT Core,, AWS IoT SiteWise AWS IoT Analytics, Amazon S3 AWS Cloud und Amazon Kinesis Video Streams) senden und eine Verbindung zu diesen herstellen. In Kombination mit Everyware Cloud, der Geräteverwaltungslösung von Eurotech, führt diese Lösung einen neuartigen Zero-Touch-Provisioning-Service 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
- AWS IoT Greengrass Kernprobleme mit der Software
- AWS IoT Greengrass Cloud-Probleme
- Hauptprobleme bei der Gerätebereitstellung
- Probleme mit den wichtigsten Gerätekomponenten
- Probleme mit den Lambda-Funktionskomponenten des Kerngeräts
- Die Version der Komponente wurde eingestellt
- Probleme mit der Greengrass-Befehlszeilenschnittstelle
- AWS Command Line Interface Probleme
- Detaillierte Fehlercodes für die Bereitstellung
- Detaillierte Statuscodes für Komponenten

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 CloudWatch Protokolle in Logs schreiben, sodass Sie Probleme bei Kerngeräten remote beheben können. Mithilfe dieser Protokolle können Sie Probleme mit Komponenten, Bereitstellungen und Kerngeräten identifizieren. Weitere Informationen finden Sie unter AWS IoT Greengrass Protokolle überwachen.

AWS IoT Greengrass Kernprobleme mit der Software

Beheben Sie Probleme mit der AWS IoT Greengrass Kernsoftware.

Themen

- ThrottlingException von der ListDeployments API
- Das Kerngerät konnte nicht eingerichtet werden
- Die AWS IoT Greengrass Core-Software konnte nicht als Systemdienst gestartet werden
- Nucleus kann nicht als Systemdienst eingerichtet werden
- Es konnte keine Verbindung hergestellt werden AWS IoT Core
- Fehler: Nicht genügend Arbeitsspeicher
- Greengrass CLI kann nicht installiert werden
- User root is not allowed to execute
- com.aws.greengrass.lifecyclemanager.GenericExternalService: Could not determine user/group to run with
- Failed to map segment from shared object: operation not permitted
- Der Windows-Dienst konnte nicht eingerichtet werden
- com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager
- com.aws.greengrass.deployment.IotJobsHelper: No connection available during subscribing to Iot Jobs descriptions topic. Will retry in sometime
- software.amazon.awssdk.services.iam.model.IamException: The security token included in the request is invalid
- software.amazon.awssdk.services.iot.model.IotException: User: <user> is not authorized to perform: iot:GetPolicy
- Error: com.aws.greengrass.shadowmanager.sync.model.FullShadowSyncRequest: Could not execute cloud shadow get request
- Operation aws.greengrass#<operation> is not supported by Greengrass
- java.io.FileNotFoundException: <stream-manager-store-root-dir>/stream_manager_metadata_store (Permission denied)
- com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: Private key or certificate with label <label> does not exist
- software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: User: <user> is not authorized to perform: secretsmanager:GetSecretValue on resource: <arn>
- software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: Access to KMS is not allowed
- java.lang.NoClassDefFoundError: com/aws/greengrass/security/CryptoKeySpi

- com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: CKR_OPERATION_NOT_INITIALIZED
- Greengrass core device stuck on nucleus v2.12.3
- Greengrass nucleus v2.14.0 systemd template issue

ThrottlingException von der ListDeployments API

`ThrottlingException`von der `ListDeployments` API: Dies wird möglicherweise angezeigt, wenn Sie eine große Anzahl von Bereitstellungen im Konto haben.

Gehen Sie wie folgt vor, um dieses Problem zu lösen:

- Wenn Sie SDK verwenden, geben Sie bitte den MaxResult Parameter an. Zum Beispiel für JavaSDK mit einem kleinen Wert (z. B. 5).
- Sie können AWS Service Quotas verwenden, um eine Limiterhöhung des Ratenlimits der `DescribeJob` API zu beantragen. Sie können zur Service-Kontingentkonsole gehen, Kontingente von auswählen AWS IoT und das Limit heißt DescribeJob Throttle Limit. Sie können es von 10 auf 50 erhöhen.

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 Sie die 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 einen Systemdienst aus. Sie können die Software mit einem Loader-Skript ausführen, das der Installer im Greengrass-Stammordner einrichtet. Weitere Informationen finden Sie unter Führen Sie die AWS IoT Greengrass Core-Software ohne Systemdienst aus.

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 unterKonfiguration der AWS IoT Greengrass Core-Software.
- Vergewissern Sie sich, dass AWS IoT das Gerät 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 erhalten 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. */path/to/use*Ersetzen Sie ihn durch den Pfad zu einem Ordner, den die AWS CRT-Bibliothek verwenden kann.

  ```
  {
    "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 unterGeräteverkehr über einen Proxy oder eine Firewall zulassen.
- Suchen Sie in den Greengrass-Protokollen nach anderen Fehlern, die andere Ursachen 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
+FvwqnKwRc0IfrRh3c/LTo6UDdyJwOOvEVPvLXCrrrUtdnniCEXAMPLE/
IvU1dYUg2RVAJBanLiHb4IgRmpRV3zrkuWJOgQs8IZZaIv2BXIa2R4Olgk
```

Sie können die AWS IoT Greengrass Core-Software auch ohne Angabe von AWS Anmeldeinformationen installieren. Für weitere Informationen siehe 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 Sie die AWS IoT Richtlinie eines Kerngeräts

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.
- Stellen Sie sicher, dass der private Schlüssel und das Zertifikat dieselbe Objektbezeichnung im HSM verwenden.
- Wenn Ihr HSM Objekte unterstützt IDs, überprüfen Sie, ob der private Schlüssel und das Zertifikat dieselbe Objekt-ID im HSM verwenden.

In der Dokumentation für Ihr 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 Bereitstellungen überarbeiten.
   - Starten Sie die AWS IoT Greengrass Core-Software neu, um die Bereitstellung erneut zu versuchen. Weitere Informationen finden Sie unter Führen Sie die AWS IoT Greengrass Core-Software aus.

   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-Dienstdatei 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"
```

Greengrass nucleus v2.14.0 systemd template issue

Dieses Problem kann auftreten, wenn Sie Greengrass Nucleus v2.14.0 auf einem Linux-Gerät mit der standardmäßigen Systemd-Dienstvorlage installiert haben. Gehen Sie wie folgt vor:

1. Führen Sie auf Ihrem Greengrass-Core-Gerät den folgenden Befehl aus, um die Systemd-Dienstdatei auf den Stand von Nucleus v2.13.0 und früheren Versionen zurückzusetzen.

   Linux or Unix

   ```
   sudo sed -i 's|ExecStart=/bin/sh -c "\(.*\) >> .*/logs/loader.log 2>&1"|
   ExecStart=/bin/sh \1|' /etc/systemd/system/greengrass.service
   ```

2. Wenden Sie die Änderungen an.

   Linux or Unix

   ```
   sudo systemctl daemon-reload
   sudo systemctl restart 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, möglicherweise wird jedoch ein falscher Bereitstellungsstatus von `INACTIVE` angezeigt.

Hauptprobleme bei der Gerätebereitstellung

Beheben Sie Bereitstellungsprobleme auf Greengrass-Core-Geräten. 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 keine `s3:GetBucketLocation` Anrufberechtigung hat. 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 Core-Geräts. 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 V2 Kerngeräte.

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. Bereitstellungen überarbeiten Weitere Informationen zum Überarbeiten 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 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 von Greengrass Nucleus.

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 AWS-mitgelieferte 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.

Im Folgenden finden Sie 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 erhält, 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 Sie die AWS IoT Richtlinie eines Kerngeräts.

Warn: com.aws.greengrass.deployment.DeploymentService: Failed to get thing group hierarchy

Diese Warnung wird möglicherweise angezeigt, wenn das Kerngerät eine Bereitstellung erhält und die AWS IoT Richtlinie des Kerngeräts 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. AWS IoT Greengrass Komponenten auf Geräten bereitstellen

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 Sie die AWS IoT Richtlinie eines Kerngeräts.

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 Sie eine 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
- Python-Skript protokolliert keine Nachrichten
- Die Komponentenkonfiguration wird nicht aktualisiert, wenn die Standardkonfiguration geändert wird
- awsiot.greengrasscoreipc.model.UnauthorizedError
- com.aws.greengrass.authorization.exceptions.AuthorizationException: Duplicate policy ID "<id>" for principal "<componentList>"
- com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 400)
- com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 403)
- com.aws.greengrass.tes.CredentialsProviderError: Could not load credentials from any providers
- Received error when attempting to retrieve ECS metadata: Could not connect to the endpoint URL: "<tokenExchangeServiceEndpoint>"
- copyFrom: <configurationPath> is already a container, not a leaf
- com.aws.greengrass.componentmanager.plugins.docker.exceptions.DockerLoginException: Error logging into the registry using credentials - 'The stub received bad data.'
- java.io.IOException: Cannot run program "cmd" ...: [LogonUser] The password for this account has expired.
- aws.greengrass.StreamManager: Instant exceeds minimum or maximum instant

Warn: '<command>' is not recognized as an internal or external command

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 Führen Sie die AWS IoT Greengrass Core-Software aus.
- 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 unterAWS IoT Greengrass Protokolle überwachen.

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

Mit dem folgenden Befehl wird eine Bereitstellung auf einem Core-Gerät erstellt.

```
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 die Rezeptur `DefaultConfiguration` 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 eine eindeutige Richtlinie zu erstellen IDs, 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 *MyGreengrassV2TokenExchangeRole* ersetzen Sie ihn 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.

JSON

```
{
  "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` anzunehmen, 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 V2 Kerngeräte und Aktualisieren Sie die AWS IoT Richtlinie eines Kerngeräts.

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 erhalten 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` Aus diesem Grund 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 habe`docker-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. *`ggc_user`*Ersetzen Sie es durch den Systembenutzer und *`password`* ersetzen Sie es 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 LocalSystem Konto zu speichern. *`password`*Ersetzen Sie es durch das von Ihnen festgelegte Benutzerkennwort.

   ```
   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 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"}
```

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
- ipc_client.py:64,HTTP Error 400:Bad Request, b'No subscription exists for the source <label-or-lambda-arn> and subject <label-or-lambda-arn>

The following cgroup subsystems are not mounted: devices, memory

In den folgenden Fällen wird dieser Fehler möglicherweise angezeigt, wenn Sie eine containerisierte Lambda-Funktion ausführen:

- Auf dem Core-Gerät ist cgroup v1 für die Speicher- oder Geräte-Cgroups nicht aktiviert.
- Auf dem Core-Gerät 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 V1-Lambda-Funktionen importieren.

Die Version der Komponente 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 zur Überarbeitung 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'

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
- Minimale AWS CLI V2-Version: v2.1.11

> **Tip**
>
> Sie können den folgenden Befehl ausführen, um die Version von zu überprüfen AWS CLI , die 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 Fehlercodes für die Bereitstellung

Verwenden Sie die Fehlercodes und Lösungen in diesen Abschnitten, um Probleme mit der Komponentenbereitstellung zu lösen, wenn Sie Greengrass Nucleus Version 2.8.0 oder höher verwenden.

Der Greengrass-Kern meldet Bereitstellungsfehler in einer Hierarchie vom unspezifischsten bis zum spezifischsten verfügbaren Code. Sie können diese Hierarchie verwenden, um den Grund für einen Bereitstellungsfehler zu ermitteln. Im Folgenden ist beispielsweise eine mögliche Fehlerhierarchie aufgeführt:

- DEPLOYMENT_FAILURE
  - ARTIFACT_DOWNLOAD_ERROR
    - IO_ERROR
      - DISK_SPACE_CRITICAL

Die Fehlercodes sind in Typen unterteilt. Jeder Typ steht für eine Klasse von Fehlern, die auftreten können. AWS IoT Greengrass meldet diese Fehlertypen in der Konsole, der API und AWS 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`.

Die Typen sind:

- PERMISSION_ERROR — Der Zugriff auf einen Vorgang, für den eine Genehmigung erforderlich ist, wurde verweigert.
- REQUEST_ERROR — Aufgrund eines Problems im Bereitstellungsdokument ist ein Fehler aufgetreten.
- COMPONENT_RECIPE_ERROR — Aufgrund eines Fehlers in einem Komponentenrezept ist ein Fehler aufgetreten.
- AWS_COMPONENT_ERROR — Beim Starten oder Entfernen einer bereitgestellten Komponente ist ein Fehler aufgetreten. AWS
- USER_COMPONENT_ERROR — Beim Starten oder Entfernen einer Benutzerkomponente ist ein Fehler aufgetreten.
- COMPONENT_ERROR — Beim Starten oder Entfernen einer Komponente ist ein Fehler aufgetreten, aber der Greengrass-Kern konnte nicht feststellen, ob es sich bei der Komponente um eine AWS bereitgestellte Komponente oder eine Benutzerkomponente handelt.
- DEVICE_ERROR — Bei der lokalen I/O ist ein Fehler aufgetreten, oder es ist ein anderer Gerätefehler aufgetreten.
- DEPENDENCY_ERROR — Bei einer Bereitstellung konnte kein Artefakt von Amazon S3 heruntergeladen oder ein Bild aus einer ECR-Registrierung abgerufen werden.
- HTTP_ERROR — Bei einer HTTP-Anfrage ist ein Fehler aufgetreten.
- NETWORK_ERROR — Im Gerätenetzwerk ist ein Fehler aufgetreten.
- NUCLEUS_ERROR — Der Greengrass-Kern konnte eine Komponente nicht lokalisieren oder die aktive Nukleus-Version nicht finden.
- SERVER_ERROR — Ein Server hat als Antwort auf eine Anfrage einen Fehler vom Typ 500 zurückgegeben.
- CLOUD_SERVICE_ERROR — Beim Cloud-Dienst ist ein Fehler aufgetreten. AWS IoT Greengrass
- UNKNOWN_ERROR — Eine ungeprüfte Ausnahme wurde von der Komponente ausgelöst.

Viele der Fehler in diesem Abschnitt führen zu zusätzlichen Informationen in den Core-Protokollen. AWS IoT Greengrass Diese Protokolle werden im lokalen Dateisystem des Kerngeräts gespeichert. Es gibt Protokolle für die AWS IoT Greengrass Core-Kernsoftware und für jede einzelne Komponente. Informationen zum Zugriff auf die Protokolle finden Sie unterGreifen Sie auf Dateisystemprotokolle zu.

Berechtigungsfehler

ACCESS_DENIED

Dieser Fehler tritt möglicherweise auf, wenn ein AWS Dienstvorgang einen 403-Fehler zurückgibt, weil die Berechtigungen nicht korrekt eingerichtet sind. Einzelheiten finden Sie im spezifischeren Fehlercode.

GET_DEPLOYMENT_CONFIGURATION_ACCESS_DENIED

Dieser Fehler wird möglicherweise angezeigt, wenn die Richtlinie die Erlaubnis zum Aufrufen des AWS IoT Vorgangs nicht zulässt. `GetDeploymentConfiguration` Fügen Sie die `greengrass::GetDeploymentConfiguration` Berechtigung zur Richtlinie des Kerngeräts hinzu.

GET_COMPONENT_VERSION_ARTIFACT_ACCESS_DENIED

Möglicherweise wird dieser Fehler angezeigt, wenn die Kerngeräterichtlinie die Genehmigung nicht zulässt. AWS IoT `greengrass:GetComponentVersionArtifact` Fügen Sie die Berechtigung zur Richtlinie des Kerngeräts hinzu.

RESOLVE_COMPONENT_CANDIDATES_ACCESS_DENIED

Möglicherweise wird dieser Fehler angezeigt, wenn die Kerngeräterichtlinie die Genehmigung nicht zulässt. AWS IoT `greengrass:ResolveComponentCandidates` Fügen Sie die Berechtigung zur Richtlinie des Kerngeräts hinzu.

GET_ECR_CREDENTIALS AL_ERROR

Dieser Fehler tritt möglicherweise auf, wenn die Bereitstellung 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.

USER_NOT_AUTHORIZED_FOR_DOCKER

Dieser Fehler kann auftreten, wenn der Greengrass-Benutzer nicht berechtigt ist, Docker zu verwenden. Stellen Sie sicher, dass Sie Greengrass als Root ausführen oder dass der

Benutzer der `docker` Gruppe hinzugefügt wurde. Versuchen Sie dann erneut, das Deployment durchzuführen.

S3_ACCESS_DENIED

Dieser Fehler kann auftreten, wenn ein Amazon S3 S3-Vorgang einen 403-Fehler zurückgibt. Weitere Informationen finden Sie in allen zusätzlichen Fehlercodes oder Protokollen.

S3_HEAD_OBJECT_ACCESS_DENIED

Dieser Fehler kann entweder auftreten, wenn die Token-Austauschrolle des Geräts es der AWS IoT Greengrass Core-Software nicht erlaubt, das Komponentenartefakt von der S3-Objekt-URL herunterzuladen, die Sie im Rezept der Komponente angeben, oder wenn das Komponentenartefakt nicht verfügbar ist. Vergewissern Sie sich, dass die Token-Austauschrolle `s3:GetObject` die URL des S3-Objekts berücksichtigt, unter der das Artefakt verfügbar ist, und ob das Artefakt vorhanden ist.

S3_GET_BUCKET_LOCATION_ACCESS_DENIED

Möglicherweise wird dieser Fehler angezeigt, wenn die Token-Austauschrolle des Geräts die `s3:GetBucketLocation` Genehmigung für den Amazon S3 S3-Bucket, in dem das Artefakt verfügbar ist, nicht zulässt. Vergewissern Sie sich, dass das Gerät die Erlaubnis erteilt, und versuchen Sie dann erneut, die Bereitstellung durchzuführen.

S3_GET_OBJECT_ACCESS_DENIED

Dieser Fehler kann entweder auftreten, wenn die Token-Austauschrolle des Geräts es der AWS IoT Greengrass Core-Software nicht erlaubt, das Komponentenartefakt von der S3-Objekt-URL herunterzuladen, die Sie im Rezept der Komponente angeben, oder wenn das Komponentenartefakt nicht verfügbar ist. Vergewissern Sie sich, dass die Token-Austauschrolle `s3:GetObject` die URL des S3-Objekts berücksichtigt, unter der das Artefakt verfügbar ist, und ob das Artefakt vorhanden ist.

Fehler bei der Anfrage

NUCLEUS_MISSING_REQUIRED_CAPABILITIES

Dieser Fehler tritt möglicherweise auf, wenn die Nucleus-Version in der Bereitstellung einen angeforderten Vorgang nicht ausführen kann, z. B. das Herunterladen einer großen Konfiguration oder das Festlegen von Linux-Ressourcenlimits. Versuchen Sie erneut, die Bereitstellung mit einer Nucleus-Version durchzuführen, die den Vorgang unterstützt.

MULTIPLE_NUCLEUS_RESOLVED_ERROR

Dieser Fehler wird möglicherweise angezeigt, wenn bei einer Bereitstellung versucht wird, mehrere Nucleus-Komponenten bereitzustellen. Sehen Sie im Protokoll nach, was den Fehler verursacht hat, und schauen Sie dann auf der Nucleus-Softwareupdateseite nach, ob das Problem in einer neueren Version von Nucleus behoben wurde, oder kontaktieren Sie uns Support.

COMPONENT_CIRCULAR_DEPENDENCY_ERROR

Dieser Fehler kann auftreten, wenn zwei Komponenten in Ihrer Bereitstellung voneinander abhängen. Überarbeiten Sie das Komponenten-Setup, sodass die Komponenten in Ihrer Bereitstellung nicht aufeinander angewiesen sind.

UNAUTHORIZED_NUCLEUS_MINOR_VERSION_UPDATE

Dieser Fehler tritt möglicherweise auf, 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 Updates für Nebenversionen von Komponenten, die von einer anderen Version abhängen, zu reduzieren. Nehmen Sie die neue Nucleus-Niederversion in die Bereitstellung auf.

MISSING_DOCKER_APPLICATION_MANAGER

Dieser Fehler wird möglicherweise angezeigt, wenn Sie eine Docker-Komponente bereitstellen, ohne den Docker-Anwendungsmanager bereitzustellen. Stellen Sie sicher, dass Ihre Bereitstellung den Docker-Anwendungsmanager enthält.

MISSING_TOKEN_EXCHANGE_SERVICE

Dieser Fehler wird möglicherweise angezeigt, wenn die Bereitstellung ein Docker-Image-Artefakt aus einer privaten ECR-Registrierung herunterladen möchte, ohne den Token-Exchange-Dienst bereitzustellen. Stellen Sie sicher, dass Ihre Bereitstellung den Token-Austauschdienst umfasst.

COMPONENT_VERSION_REQUIREMENTS_NOT_MET

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>.

THROTTLING_ERROR

Dieser Fehler tritt möglicherweise auf, wenn ein AWS Servicevorgang ein Ratenkontingent überschritten hat. Wiederholen Sie die Bereitstellung.

CONFLICTED_REQUEST

Dieser Fehler tritt möglicherweise auf, wenn ein AWS Dienstvorgang einen 409-Fehler zurückgibt, weil Ihre Bereitstellung versucht, mehr als einen Vorgang gleichzeitig auszuführen. Wiederholen Sie die Bereitstellung.

RESOURCE_NOT_FOUND

Dieser Fehler kann auftreten, wenn ein AWS Servicevorgang einen 404-Fehler zurückgibt, weil eine Ressource nicht gefunden werden konnte. Suchen Sie im Protokoll nach der fehlenden Ressource.

RUN_WITH_CONFIG_NOT_VALID

Dieser Fehler wird möglicherweise angezeigt, wenn die für die Ausführung der Komponente angegebenen `windowsUser` Informationen `posixUserposixGroup`, oder nicht gültig sind. Überprüfen Sie, ob der Benutzer gültig ist, und wiederholen Sie dann die Bereitstellung.

UNSUPPORTED_REGION

Dieser Fehler wird möglicherweise angezeigt, wenn die für die Bereitstellung angegebene Region von nicht unterstützt wird. AWS IoT GreengrassÜberprüfen Sie die Region und versuchen Sie es erneut mit der Bereitstellung.

IOT_CRED_ENDPOINT_NOT_VALID

Dieser Fehler wird möglicherweise angezeigt, wenn der in der Konfiguration angegebene Endpunkt für Anmeldeinformationen nicht gültig ist AWS IoT . Überprüfen Sie den Endpunkt und versuchen Sie es erneut mit Ihrer Anfrage.

IOT_DATA_ENDPOINT_NOT_VALID

Möglicherweise wird dieser Fehler angezeigt, wenn der in der Konfiguration angegebene Datenendpunkt nicht gültig ist AWS IoT . Überprüfen Sie den Endpunkt und versuchen Sie es erneut mit Ihrer Anfrage.

S3_HEAD_OBJECT_RESOURCE_NOT_FOUND

Dieser Fehler wird möglicherweise angezeigt, wenn das Komponentenartefakt nicht unter der S3-Objekt-URL verfügbar ist, 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.

S3_GET_BUCKET_LOCATION_RESOURCE_NOT_FOUND

Möglicherweise wird dieser Fehler angezeigt, wenn der Amazon S3 S3-Bucket nicht gefunden wird. Überprüfen Sie, ob der Bucket vorhanden ist, und versuchen Sie es erneut mit der Bereitstellung.

S3_GET_OBJECT_RESOURCE_NOT_FOUND

Dieser Fehler wird möglicherweise angezeigt, wenn das Komponentenartefakt nicht unter der S3-Objekt-URL verfügbar ist, 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.

IO_MAPPING_ERROR

Dieser Fehler tritt möglicherweise auf, wenn beim Analysieren des Bereitstellungsdokuments oder -rezepts ein I/O-Fehler auftritt. Weitere Informationen finden Sie in allen zusätzlichen Fehlercodes oder Protokollen.

Fehler bei der Rezeptur der Komponente

RECIPE_PARSE_ERROR

Dieser Fehler tritt möglicherweise auf, wenn das Bereitstellungsrezept nicht analysiert werden konnte, da ein Fehler in der Struktur des Rezepts vorliegt. Vergewissern Sie sich, dass das Rezept korrekt formatiert ist, und versuchen Sie es erneut mit der Bereitstellung.

RECIPE_METADATA_PARSE_ERROR

Dieser Fehler tritt möglicherweise auf, wenn die aus der Cloud heruntergeladenen Metadaten des Bereitstellungsrezepts nicht analysiert werden konnten. Kontakt Support.

ARTIFACT_URI_NOT_VALID

Dieser Fehler kann auftreten, wenn eine Artefakt-URI in einem Rezept nicht richtig formatiert ist. Suchen Sie im Protokoll nach dem ungültigen URI, aktualisieren Sie den URI im Rezept und versuchen Sie dann erneut, die Bereitstellung durchzuführen.

S3_ARTIFACT_URI_NOT_VALID

Dieser Fehler kann auftreten, wenn die Amazon S3 S3-URI eines Artefakts in einem Rezept nicht gültig ist. Suchen Sie im Protokoll nach dem ungültigen URI, aktualisieren Sie den URI im Rezept und versuchen Sie dann erneut, die Bereitstellung durchzuführen.

DOCKER_ARTIFACT_URI_NOT_VALID

Möglicherweise wird dieser Fehler angezeigt, wenn die Docker-URI eines Artefakts in einem Rezept nicht gültig ist. Suchen Sie im Protokoll nach dem ungültigen URI, aktualisieren Sie den URI im Rezept und versuchen Sie dann erneut, die Bereitstellung durchzuführen.

EMPTY_ARTIFACT_URI

Dieser Fehler kann auftreten, wenn die URI eines Artefakts nicht in einem Rezept angegeben ist. Suchen Sie im Protokoll nach dem Artefakt, dem ein URI fehlt, aktualisieren Sie den URI im Rezept und versuchen Sie dann erneut, die Bereitstellung durchzuführen.

EMPTY_ARTIFACT_SCHEME

Dieser Fehler kann auftreten, wenn für ein Artefakt kein URI-Schema definiert ist. Suchen Sie im Protokoll nach dem URI, der nicht gültig ist, aktualisieren Sie den URI im Rezept und versuchen Sie dann erneut, die Bereitstellung durchzuführen.

UNSUPPORTED_ARTIFACT_SCHEME

Dieser Fehler wird möglicherweise angezeigt, wenn ein URI-Schema von der laufenden Nucleus-Version nicht unterstützt wird. Entweder ist ein URI nicht gültig oder Sie müssen die Nucleus-Version aktualisieren. Wenn der URI nicht gültig ist, suchen Sie im Protokoll nach dem ungültigen URI, aktualisieren Sie den URI im Rezept und versuchen Sie dann erneut, die Bereitstellung durchzuführen.

RECIPE_MISSING_MANIFEST

Möglicherweise wird dieser Fehler angezeigt, wenn der Manifestabschnitt nicht im Rezept enthalten ist. Fügen Sie das Manifest zum Rezept hinzu und versuchen Sie es erneut mit der Bereitstellung.

RECIPE_MISSING_ARTIFACT_HASH_ALGORITHM

Dieser Fehler kann auftreten, wenn ein Artefakt, das nicht lokal ist, in einem Rezept ohne Hash-Algorithmus angegeben wird. Fügen Sie den Algorithmus dem Artefakt hinzu und versuchen Sie es dann 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, die Bereitstellung durchzuführen. Weitere Informationen finden Sie unter Error: com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchException: Integrity check for downloaded artifact failed. Probably due to file corruption..

COMPONENT_DEPENDENCY_NOT_VALID

Dieser Fehler wird möglicherweise angezeigt, wenn der in einem Bereitstellungsrezept angegebene Abhängigkeitstyp nicht gültig ist. Überprüfen Sie das Rezept und versuchen Sie es dann erneut mit Ihrer Anfrage.

CONFIG_INTERPOLATE_ERROR

Dieser Fehler kann auftreten, wenn Sie eine Rezeptvariable interpolieren. Einzelheiten finden Sie im Protokoll.

IO_MAPPING_ERROR

Dieser Fehler tritt möglicherweise auf, wenn beim Analysieren des Bereitstellungsdokuments oder -rezepts ein I/O-Fehler auftritt. Weitere Informationen finden Sie in allen zusätzlichen Fehlercodes oder Protokollen.

AWS Komponentenfehler, 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 von bereitgestellt wird AWS IoT Greengrass, kehrt sie zurück`AWS_COMPONENT_ERROR`. Wenn die Komponente als Benutzerkomponente identifiziert wird, kehrt der Greengrass-Kern zurück`USER_COMPONENT_ERROR`. Wenn der Greengrass-Kern das nicht weiß, kehrt er zurück`COMPONENT_ERROR`.

COMPONENT_UPDATE_ERROR

Dieser Fehler kann auftreten, wenn eine Komponente während einer Bereitstellung nicht aktualisiert wird. Überprüfen Sie alle weiteren Fehlercodes oder schauen Sie im Protokoll nach, was den Fehler verursacht hat.

COMPONENT_BROKEN

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.

REMOVE_COMPONENT_ERROR

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 tritt möglicherweise auf, wenn die Bootstrap-Aufgabe einer Komponente länger als das konfigurierte Timeout gedauert hat. 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 tritt möglicherweise auf, wenn bei der Bootstrap-Aufgabe einer Komponente ein Fehler auftritt. Überprüfen Sie das Protokoll auf Fehlerdetails und versuchen Sie dann erneut, die Bereitstellung durchzuführen.

COMPONENT_CONFIGURATION_NOT_VALID

Dieser Fehler kann auftreten, wenn der Nucleus die bereitgestellte Konfiguration für die Komponente nicht validieren 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 wird möglicherweise angezeigt, wenn nicht genügend Speicherplatz zur Verfügung steht, um eine Bereitstellungsanfrage abzuschließen. Sie müssen über mindestens 20 MB verfügbarer Speicherplatz verfügen oder über 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 ein 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.

LOCAL_RECIPE_NOT_FOUND

Dieser Fehler kann auftreten, wenn die lokale Kopie einer Rezeptdatei nicht gefunden werden konnte. Versuchen Sie die Bereitstellung erneut.

LOCAL_RECIPE_CORRUPTED

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 es erneut mit der Bereitstellung.

LOCAL_RECIPE_METADATA_NOT_FOUND

Dieser Fehler kann auftreten, wenn die lokale Kopie der Rezept-Metadatendatei nicht gefunden werden konnte. Versuchen Sie die Bereitstellung erneut.

LAUNCH_DIRECTORY_CORRUPTED

Dieser Fehler kann auftreten, wenn das Verzeichnis, das zum Starten von Greengrass Nucleus (`/greengrass/v2/alts/current`) verwendet wurde, seit dem letzten Start des Nucleus geändert wurde. Starten Sie den Nucleus neu und versuchen Sie dann erneut, das Deployment durchzuführen.

HASHING_ALGORITHM_UNAVAILABLE

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.

DEVICE_CONFIG_NOT_VALID_FOR_ARTIFACT_DOWNLOAD

Dieser Fehler wird möglicherweise angezeigt, wenn ein Fehler in der Gerätekonfiguration vorliegt, der das Herunterladen des Artefakts durch die Bereitstellung von Amazon S3 oder der Greengrass-Cloud verhindert hat. Überprüfen Sie das Protokoll auf einen bestimmten Konfigurationsfehler und versuchen Sie dann erneut, die Bereitstellung durchzuführen.

Abhängigkeitsfehler

DOCKER_ERROR

Möglicherweise wird dieser Fehler beim Abrufen eines Docker-Images angezeigt. Weitere Informationen finden Sie in allen zusätzlichen Fehlercodes oder Protokollen.

DOCKER_SERVICE_UNAVAILABLE

Möglicherweise wird dieser Fehler angezeigt, wenn Greengrass sich nicht bei der Docker-Registrierung anmelden konnte. Überprüfen Sie das Protokoll auf einen bestimmten Fehler und versuchen Sie dann erneut, die Bereitstellung durchzuführen.

DOCKER_LOGIN_ERROR

Dieser Fehler wird möglicherweise angezeigt, 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. Suchen Sie im Protokoll nach einem bestimmten Fehler und versuchen Sie die Bereitstellung erneut.

DOCKER_IMAGE_QUERY_ERROR

Dieser Fehler wird möglicherweise angezeigt, wenn bei der Abfrage von Docker nach verfügbaren Images ein unerwarteter Fehler auftritt. Suchen Sie im Protokoll nach dem spezifischen Fehler und versuchen Sie die Bereitstellung erneut.

S3_ERROR

Dieser Fehler kann auftreten, wenn Sie ein Amazon S3 S3-Artefakt herunterladen. Weitere Informationen finden Sie in allen zusätzlichen Fehlercodes oder Protokollen.

S3_RESOURCE_NOT_FOUND

Dieser Fehler kann auftreten, wenn ein Amazon S3 S3-Vorgang einen 404-Fehler zurückgibt. Weitere Informationen finden Sie in allen zusätzlichen Fehlercodes oder Protokollen.

S3_BAD_REQUEST

Dieser Fehler kann auftreten, wenn ein Amazon S3 S3-Vorgang einen 400-Fehler zurückgibt. Überprüfen Sie das Protokoll auf einen bestimmten Fehler und versuchen Sie es erneut mit der Anfrage.

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.

DOWNLOAD_DEPLOYMENT_DOCUMENT_ERROR

Dieser Fehler wird möglicherweise angezeigt, wenn beim Herunterladen des Bereitstellungsdokuments ein HTTP-Fehler aufgetreten ist. Suchen Sie im Protokoll nach dem 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. Suchen Sie im Protokoll nach dem spezifischen HTTP-Fehler.

DOWNLOAD_GREENGRASS_ARTIFACT_ERROR

Dieser Fehler kann auftreten, wenn beim Herunterladen eines Artefakts einer öffentlichen Komponente ein HTTP-Fehler aufgetreten ist. Suchen Sie im Protokoll nach dem spezifischen HTTP-Fehler.

Netzwerkfehler

NETWORK_ERROR

Dieser Fehler wird möglicherweise angezeigt, wenn während einer Bereitstellung ein Verbindungsproblem auftritt. Überprüfen Sie die Verbindung des Geräts mit dem Internet und versuchen Sie die Bereitstellung erneut.

Nucleus-Fehler

BAD_REQUEST

Dieser Fehler wird möglicherweise angezeigt, wenn ein AWS Cloud-Vorgang einen 400-Fehler zurückgibt. Sehen Sie im Protokoll nach, welche API den Fehler verursacht hat, und schauen Sie dann auf der Nucleus-Softwareupdateseite nach, ob das Problem in einer späteren Version von Nucleus behoben wurde, oder kontaktieren Sie uns Support.

NUCLEUS_VERSION_NOT_FOUND

Dieser Fehler kann auftreten, wenn ein Core-Gerät die Version des aktiven Nucleus nicht finden kann. Sehen Sie im Protokoll nach, was den Fehler verursacht hat, und schauen Sie dann auf der Nucleus-Softwareupdateseite nach, ob das Problem in einer neueren Version von Nucleus behoben wurde, oder kontaktieren Sie uns Support.

NUCLEUS_RESTART_FAILURE

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 kontaktieren Sie uns Support.

INSTALLED_COMPONENT_NOT_FOUND

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-Softwareupdateseite nach, ob das Problem in einer neueren Version von Nucleus behoben wurde, oder kontaktieren Sie uns Support.

DEPLOYMENT_DOCUMENT_NOT_VALID

Möglicherweise wird dieser Fehler angezeigt, wenn das Gerät ein Bereitstellungsdokument empfängt, das nicht gültig ist. Überprüfen Sie alle weiteren Fehlercodes oder schauen Sie im Protokoll nach, was den Fehler verursacht hat.

EMPTY_DEPLOYMENT_REQUEST

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-Softwareupdateseite nach, ob das Problem in einer neueren Version von Nucleus behoben wurde, oder kontaktieren Sie uns Support.

DEPLOYMENT_DOCUMENT_PARSE_ERROR

Dieser Fehler wird möglicherweise angezeigt, 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 kontaktieren Sie uns Support.

COMPONENT_METADATA_NOT_VALID_IN_DEPLOYMENT

Dieser Fehler wird möglicherweise angezeigt, wenn die Bereitstellungsanfrage Komponentenmetadaten enthält, die nicht gültig sind. Sehen Sie im Protokoll nach, was den Fehler verursacht hat, und schauen Sie dann auf der Nucleus-Softwareupdateseite nach, ob das Problem in einer neueren Version von Nucleus behoben wurde, oder kontaktieren Sie uns Support.

LAUNCH_DIRECTORY_CORRUPTED

Dieser Fehler kann auftreten, 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 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

SERVER_ERROR

Dieser Fehler tritt möglicherweise auf, wenn ein AWS Dienstvorgang einen Fehler vom Typ 500 zurückgibt, weil der Dienst die Anfrage momentan nicht verarbeiten kann. Versuchen Sie die Bereitstellung später erneut.

S3_SERVER_ERROR

Dieser Fehler kann auftreten, wenn ein Amazon S3 S3-Vorgang einen Fehler vom Typ 500 zurückgibt. Weitere Informationen finden Sie in allen zusätzlichen Fehlercodes oder Protokollen.

Fehler beim Cloud-Dienst

RESOLVE_COMPONENT_CANDIDATES_BAD_RESPONSE

Dieser Fehler kann auftreten, wenn der Greengrass-Cloud-Dienst eine inkompatible Antwort auf den `ResolveComponentCandidates` Vorgang sendet. Sehen Sie im Protokoll nach, was den Fehler verursacht hat, und schauen Sie dann auf der Nucleus-Softwareupdateseite nach, ob das Problem in einer neueren Version von Nucleus behoben wurde, oder kontaktieren Sie uns Support.

DEPLOYMENT_DOCUMENT_SIZE_EXCEEDED

Dieser Fehler wird möglicherweise angezeigt, wenn das angeforderte Bereitstellungsdokument das maximale Größenkontingent überschritten hat. Reduzieren Sie die Größe des Bereitstellungsdokuments und versuchen Sie es erneut.

GREENGRASS_ARTIFACT_SIZE_NOT_FOUND

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-Softwareupdateseite nach, ob das Problem in einer neueren Version von Nucleus behoben wurde, oder kontaktieren Sie uns. Support

DEPLOYMENT_DOCUMENT_NOT_VALID

Möglicherweise wird dieser Fehler angezeigt, wenn das Gerät ein Bereitstellungsdokument empfängt, das nicht gültig ist. Überprüfen Sie alle weiteren Fehlercodes oder schauen Sie im Protokoll nach, was den Fehler verursacht hat.

EMPTY_DEPLOYMENT_REQUEST

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-Softwareupdateseite nach, ob das Problem in einer neueren Version von Nucleus behoben wurde, oder kontaktieren Sie uns Support.

DEPLOYMENT_DOCUMENT_PARSE_ERROR

Dieser Fehler wird möglicherweise angezeigt, 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 kontaktieren Sie uns Support.

COMPONENT_METADATA_NOT_VALID_IN_DEPLOYMENT

Dieser Fehler wird möglicherweise angezeigt, wenn die Bereitstellungsanfrage Komponentenmetadaten enthält, die nicht gültig sind. Sehen Sie im Protokoll nach, was den Fehler verursacht hat, und schauen Sie dann auf der Nucleus-Softwareupdateseite nach, ob das Problem in einer neueren Version von Nucleus behoben wurde, oder kontaktieren Sie uns Support.

Generische Fehler

Diesen generischen Fehlern ist kein Fehlertyp zugeordnet.

DEPLOYMENT_INTERRUPTED

Dieser Fehler tritt möglicherweise auf, wenn eine Bereitstellung aufgrund eines Herunterfahrens von Nucleus oder eines anderen externen Ereignisses nicht abgeschlossen werden kann. Weitere Informationen finden Sie in allen weiteren Fehlercodes oder Protokollen.

ARTIFACT_DOWNLOAD_ERROR

Dieser Fehler kann auftreten, wenn beim Herunterladen eines Artefakts ein Problem auftritt. Weitere Informationen finden Sie in allen zusätzlichen Fehlercodes oder Protokollen.

NO_AVAILABLE_COMPONENT_VERSION

Dieser Fehler kann auftreten, wenn eine Komponentenversion nicht in der Cloud oder lokal existiert oder wenn ein Konflikt bei der Abhängigkeitsauflösung vorliegt. Weitere Informationen finden Sie in allen zusätzlichen Fehlercodes oder Protokollen.

COMPONENT_PACKAGE_LOADING_ERROR

Dieser Fehler kann auftreten, wenn bei der Verarbeitung der heruntergeladenen Artefakte ein Fehler auftritt. Weitere Informationen finden Sie in allen zusätzlichen Fehlercodes oder Protokollen.

WOLKE_ API_ERROR

Möglicherweise wird dieser Fehler angezeigt, wenn beim Aufrufen einer AWS Service-API ein Fehler aufgetreten ist. Weitere Informationen finden Sie in allen zusätzlichen Fehlercodes oder Protokollen.

IO_ERROR

Dieser Fehler wird möglicherweise angezeigt, wenn während einer Bereitstellung ein I/O-Fehler auftritt. Weitere Informationen finden Sie in allen zusätzlichen Fehlercodes oder Protokollen.

COMPONENT_UPDATE_ERROR

Dieser Fehler kann auftreten, wenn eine Komponente während einer Bereitstellung nicht aktualisiert wird. Überprüfen Sie alle weiteren Fehlercodes oder schauen Sie im Protokoll nach, was den Fehler verursacht hat.

Unbekannter Fehler

DEPLOYMENT_FAILURE

Dieser Fehler tritt möglicherweise auf, 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-Softwareupdateseite nach, ob das Problem in einer neueren Version von Nucleus behoben wurde, oder kontaktieren Sie uns Support.

DEPLOYMENT_TYPE_NOT_VALID

Dieser Fehler wird möglicherweise angezeigt, 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 kontaktieren Sie uns Support.

Detaillierte Statuscodes für Komponenten

Verwenden Sie die Statuscodes und Lösungen in diesen Abschnitten, um Probleme mit Komponenten zu lösen, wenn Sie Greengrass Nucleus Version 2.8.0 oder höher verwenden.

Viele der Status in diesem Thema enthalten zusätzliche Informationen in den AWS IoT Greengrass Core-Protokollen. Diese Protokolle werden im lokalen Dateisystem des Kerngeräts gespeichert. Es gibt Protokolle für jede einzelne Komponente. Informationen zum Zugriff auf die Protokolle finden Sie unterGreifen Sie auf Dateisystemprotokolle zu.

INSTALL_ERROR

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 es erneut mit der Bereitstellung.

INSTALL_IO_ERROR

Dies kann auftreten, wenn bei der Installation einer Komponente ein I/O-Fehler aufgetreten ist. Einzelheiten zu dem Fehler finden Sie im Fehlerprotokoll der Komponente.

INSTALL_MISSING_DEFAULT_RUNWITH

Dieser Fehler tritt möglicherweise auf, wenn Sie nicht ermitteln AWS IoT Greengrass können, welcher Benutzer oder welche Gruppe bei der Installation einer Komponente verwendet werden soll. Vergewissern Sie sich, dass der `runWith` Abschnitt Ihrer Installationsanleitung einen gültigen Benutzer oder eine gültige Gruppe enthält.

INSTALL_TIMEOUT

Möglicherweise wird dieser Fehler angezeigt, wenn das Installationsskript nicht innerhalb des konfigurierten Timeouts abgeschlossen wurde. Verlängern 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.

STARTUP_ERROR

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_NOT_VALID

Dieser Fehler kann auftreten, wenn die Installation einer Komponente nicht abgeschlossen werden konnte, weil der `startup` Abschnitt des Rezepts nicht gültig ist. Suchen Sie im Startbereich Ihres Rezepts nach Fehlern und versuchen Sie es erneut mit der Bereitstellung.

STARTUP_IO_ERROR

Dies kann auftreten, wenn beim Start einer Komponente ein I/O-Fehler aufgetreten ist. Einzelheiten zu dem Fehler finden Sie im Fehlerprotokoll der Komponente.

STARTUP_MISSING_DEFAULT_RUNWITH

Dieser Fehler tritt möglicherweise auf, wenn Sie nicht ermitteln AWS IoT Greengrass können, welcher Benutzer oder welche Gruppe beim Ausführen einer Komponente verwendet werden soll. Vergewissern Sie sich, dass der `runWith` Abschnitt Ihres Startrezepts einen gültigen Benutzer oder eine gültige Gruppe enthält.

STARTUP_TIMEOUT

Möglicherweise wird dieser Fehler angezeigt, wenn das Startskript nicht innerhalb des konfigurierten Timeout-Zeitraums abgeschlossen wurde. Verlängern 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 tritt möglicherweise auf, wenn Sie nicht ermitteln AWS IoT Greengrass können, welcher Benutzer oder welche Gruppe beim Ausführen einer Komponente verwendet werden soll.

Vergewissern Sie sich, dass der `runWith` Abschnitt Ihres Ausführungsrezepts 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. Suchen Sie im Abschnitt „Ausführen“ Ihres Rezepts nach Fehlern und versuchen Sie es erneut mit der Bereitstellung.

RUN_IO_ERROR

Dies kann auftreten, wenn während der Ausführung der Komponente ein I/O-Fehler aufgetreten ist. Einzelheiten zu dem Fehler finden Sie im Fehlerprotokoll der Komponente.

RUN_TIMEOUT

Möglicherweise wird dieser Fehler angezeigt, wenn das Ausführungsskript nicht innerhalb des konfigurierten Timeout-Zeitraums abgeschlossen wurde. Verlängern 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.

SHUTDOWN_ERROR

Dies kann auftreten, wenn beim Herunterfahren 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

Möglicherweise wird dieser Fehler angezeigt, wenn das Shutdown-Skript nicht innerhalb des konfigurierten Timeout-Zeitraums abgeschlossen wurde. Verlängern 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.

Kennzeichnen Sie Ihre AWS IoT Greengrass Version 2 Ressourcen

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 für AWS IoT Abrechnungsgruppen oder Kostenverteilungsberichte nicht unterstützt.

Verwenden von Tags in AWS IoT Greengrass V2

Sie können Tags verwenden, um Ihre AWS IoT Greengrass Ressourcen nach Zweck, Eigentümer, Umgebung oder einer anderen Klassifizierung für Ihren Anwendungsfall zu kategorisieren. Wenn Sie über viele Ressourcen desselben Typs verfügen, können Sie anhand von Tags eine bestimmte Ressource leichter identifizieren.

Jeder Tag (Markierung) besteht aus einem Schlüssel und einem optionalen Wert, beides können Sie bestimmen. Sie könnten beispielsweise eine Reihe von Tags für Ihre Kerngeräte definieren, anhand derer Sie diese anhand der Kunden, denen die Geräte gehören, nachverfolgen 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.

Markieren Sie mit dem AWS-Managementkonsole

Der Tag-Editor im AWS-Managementkonsole bietet Ihnen eine zentrale, einheitliche Möglichkeit, Ihre Tags für Ressourcen aus allen AWS Diensten zu erstellen und zu verwalten. Weitere Informationen finden Sie unter Tag-Editor im AWS -Ressourcengruppen -Benutzerhandbuch.

Taggen Sie mit der AWS IoT Greengrass V2 API

Sie können die AWS IoT Greengrass V2 API auch 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 Sie den TagResourceVorgang, um einer vorhandenen Ressource Tags hinzuzufügen oder Tag-Werte zu aktualisieren.
- Verwenden Sie den UntagResourceVorgang, um Tags aus einer Ressource zu entfernen.
- Um die mit einer Ressource verknüpften Tags abzurufen, verwenden Sie die ListTagsForResourceOperation oder beschreiben Sie die Ressource und überprüfen Sie ihre `tags` Eigenschaften.

In der folgenden Tabelle sind Ressourcen aufgeführt, die Sie mithilfe der AWS IoT Greengrass V2 API taggen können, sowie die entsprechenden `Create` `Describe` `Get` Und-Oder-Operationen.

Ressourcen, die markiert AWS IoT Greengrass V2 werden können

Ressource	Operation erstellen	Beschreiben Sie oder lassen Sie sich operieren
Kerngerät	Keine. Führen Sie die AWS IoT Greengrass Core-Software auf einem Gerät aus, um ein Kerngerät 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.
- UntagResource— Entfernt Tags aus einer Ressource.

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 Sie Ressourcen-Tags verwenden, um den Benutzerzugriff und die Benutzerberechtigungen zu steuern. Beispielsweise können Richtlinien Benutzern erlauben, nur die Ressourcen zu erstellen, die einen spezifisches Tag aufweisen. Richtlinien können auch verhindern, dass Benutzer Ressourcen mit bestimmten Tags erstellen oder ändern.

> **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 Bedingungskontextschlüssel und -werte 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`*

Erfordern, dass bei der Erstellung oder Änderung einer markierbaren Ressource ein bestimmtes Tag verwendet oder nicht verwendet wird.

`aws:TagKeys: [tag-key, ...]`

Erfordern, dass bei der Erstellung oder Änderung einer markierbaren Ressource ein bestimmter Satz von Tag-Schlüsseln verwendet oder nicht verwendet wird.

> **Note**
>
> Die Bedingungskontextschlüssel und -werte in einer IAM-Richtlinie gelten nur für Aktionen, für die eine markierbare Ressource als erforderlichen Parameter erforderlich ist. Sie können beispielsweise den tagbasierten bedingten Zugriff für einrichten. GetComponent

Weitere Informationen finden Sie unter Steuern des Zugriffs auf AWS Ressourcen mithilfe von Ressourcen-Tags und IAM-JSON-Richtlinienreferenz im IAM-Benutzerhandbuch.

AWS IoT Greengrass Ressourcen erstellen mit AWS CloudFormation

AWS IoT Greengrass ist in einen Service integriert AWS CloudFormation, der Ihnen hilft, Ihre AWS Ressourcen zu modellieren und einzurichten, sodass Sie weniger Zeit mit der Erstellung und Verwaltung Ihrer Ressourcen und Infrastruktur verbringen müssen. Sie erstellen eine Vorlage, die alle gewünschten AWS Ressourcen beschreibt (z. B. Komponentenversionen und Bereitstellungen) und CloudFormation diese Ressourcen für Sie bereitstellt und konfiguriert.

Wenn Sie sie verwenden CloudFormation, können Sie Ihre Vorlage wiederverwenden, um Ihre AWS IoT Greengrass Ressourcen konsistent und wiederholt einzurichten. Beschreiben Sie Ihre Ressourcen einmal und stellen Sie dann dieselben Ressourcen immer wieder in mehreren AWS-Konten Regionen bereit.

AWS IoT Greengrass und CloudFormation Vorlagen

Um Ressourcen für und zugehörige Dienste bereitzustellen AWS IoT Greengrass und zu konfigurieren, müssen Sie CloudFormation Vorlagen verstehen. Vorlagen sind formatierte Textdateien in JSON oder YAML. Diese Vorlagen beschreiben die Ressourcen, die Sie in Ihren CloudFormation Stacks bereitstellen möchten. Wenn Sie mit JSON oder YAML nicht vertraut sind, können Sie CloudFormation Designer verwenden, um Ihnen die ersten Schritte mit Vorlagen zu erleichtern. CloudFormation Weitere Informationen finden Sie unter Was ist CloudFormation - Designer? im AWS CloudFormation -Benutzerhandbuch.

AWS IoT Greengrass unterstützt das Erstellen von Komponentenversionen und Bereitstellungen in. CloudFormationWeitere Informationen, einschließlich Beispielen für JSON- und YAML-Vorlagen für Komponentenversionen und Bereitstellungen, finden Sie in der Referenz zum AWS IoT Greengrass Ressourcentyp im AWS CloudFormation Benutzerhandbuch.

ComponentVersion Beispiel für eine Vorlage

Das Folgende ist die YAML-Vorlage für eine Version einer einfachen Komponente. Das JSON-Rezept enthält zur besseren Lesbarkeit Zeilenumbrüche.

```
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 für eine 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
```

Erfahren Sie mehr über CloudFormation

Weitere Informationen CloudFormation dazu finden Sie in den folgenden Ressourcen:

- AWS CloudFormation
- AWS CloudFormation Benutzerhandbuch
- CloudFormation API Reference
- AWS CloudFormation -Benutzerhandbuch für die Befehlszeilenschnittstelle

AWS IoT Greengrass Open-Source-Kernsoftware

Die AWS IoT Greengrass Version 2 Edge-Runtime (Nucleus) und andere Komponenten der AWS IoT Greengrass Core-Software sind Open Source. Das bedeutet, dass Sie den Code überprüfen können, um Fehler bei 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 Greengrass Core-Software finden Sie bei der Organisation aws-greengrass unter. GitHub Ihre Verwendung von Open-Source-Software unterliegt der Open-Source-Lizenz im entsprechenden Repository. GitHub

Ihre Nutzung der AWS IoT Greengrass Core-Software und der 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
Greengrass CLI v2.16.1 veröffentlicht	Die Greengrass CLI-Kompo nente v2.16.1 ist verfügbar.	23. Dezember 2025
AWS IoT Greengrass Softwareupdate Core v2.16.1	Diese Version stellt Version 2.16.1 der Greengrass Nucleus-Komponente und mit Updates AWS bereitgestellte Komponenten bereit.	23. Dezember 2025
System Log Forwarder v2.1.0 veröffentlicht	System Log Forwarder v2.1.0 ist verfügbar.	6. November 2025
Shadow Manager v2.3.12 veröffentlicht	Shadow Manager v2.3.12 ist verfügbar.	6. November 2025
Log Manager v2.3.11 veröffent licht	Die Log Manager-Komponente v2.3.11 ist verfügbar.	6. November 2025
Greengrass Nucleus Lite-Komponente v2.3.0 veröffent licht	Version 2.3.0 der Greengras s Nucleus Lite-Komponente ist verfügbar. Diese Version bietet Unterstützung für TPM 2.0, eine breitere Palette von Betriebssystemen und IPC. RestartComponent Diese Version enthält auch	6. November 2025

	mehrere andere Bugfixes und Verbesserungen.	
Greengrass CLI v2.16.0 veröffentlicht	Die Greengrass CLI-Komponente v2.16.0 ist verfügbar.	6. November 2025
AWS IoT Greengrass Softwareupdate Core v2.16.0	Diese Version stellt Version 2.16.0 der Greengrass Nucleus-Komponente und mit Updates AWS bereitgestellte Komponenten bereit.	6. November 2025
AWS IoT Greengrass Softwareupdate Core v2.15.1	Diese Version stellt Version 2.15.1 der Greengrass Nucleus-Komponente und mit Updates AWS bereitgestellte Komponenten bereit.	6. November 2025
System Log Forwarder v2.0.1 veröffentlicht	Der System Log Forwarder v2.0.1 ist verfügbar. Diese Version aktualisiert das Komponentenrezept, um aarch64-Systeme (arm64) ordnungsgemäß zu unterstützen.	2. Oktober 2025
Greengrass Nucleus Lite-Komponente v2.2.2 veröffentlicht	Version 2.2.2 der Greengrass Nucleus Lite-Komponente ist verfügbar. Diese Version enthält mehrere Bugfixes und Verbesserungen.	18. August 2025
System Log Forwarder v2.0.0 wurde veröffentlicht	Die Systemprotokollweiterleitungskomponente v2.0.0 ist verfügbar. Erste Version veröffentlicht.	25 Juli 2025

Greengrass Nucleus Lite-Komponente v2.2.1 veröffentlicht	Version 2.2.1 der Greengrass Nucleus Lite-Komponente ist verfügbar. Diese Version behebt ein Problem, bei dem der Nucleus keine TES-Anmeldeinformationen abrufen kann.	25 Juli 2025
Greengrass Nucleus Lite-Komponente v2.2.0 veröffentlicht	Version 2.2.0 der Greengrass Nucleus Lite-Komponente ist verfügbar. Diese Version bietet Unterstützung für Container-Image-Artefakte. URIs	3. Juli 2025
AWS IoT Greengrass Softwareupdate Core v2.15.0	Diese Version stellt Version 2.15.0 der Greengrass Nucleus-Komponente und mit Updates AWS bereitgestellte Komponenten bereit.	3. Juli 2025
Secure Tunneling v1.1.3 veröffentlicht	Secure Tunneling v1.1.3 ist verfügbar. Diese Version aktualisiert den zugrunde liegenden AWS IoT Geräteclient, der von der Komponente aufgerufen wird, von Version 1.10.0 auf Version 1.10.1. Diese Version behebt auch die Kompatibilitätsprobleme der GNU C Library (Glibc) in der Secure Tunneling-Komponente Version 1.1.2.	16. Mai 2025

Greengrass Nucleus Lite-Komponente v2.1.0 veröffentlicht	Version 2.1.0 der Greengrass Nucleus Lite-Komponente ist verfügbar. Diese Version fügt HTTP-Proxyunterstützung hinzu. Diese Version enthält auch mehrere andere Bugfixes und Verbesserungen.	5. Mai 2025
Secret Manager v2.2.5 veröffentlicht	Secret Manager v2.2.5 ist verfügbar. Diese Version behebt ein Problem, bei dem ein Secret nicht aus dem abgerufen wird, AWS Cloud wenn es nicht im lokalen Cache vorhanden ist.	16. April 2025
AWS IoT Greengrass Core v2.14.3-Softwareupdate	Diese Version stellt Version 2.14.3 der Greengrass Nucleus-Komponente und mit Updates AWS bereitgestellte Komponenten bereit.	11. April 2025
Secret Manager v2.2.4 veröffentlicht	Secret Manager v2.2.4 ist verfügbar. Diese Version reduziert die Häufigkeit von Schreibvorgängen in den lokalen Geheimspeicher. Secret Manager schreibt jetzt nur noch in den lokalen Speicher, wenn Geheimnisse aktualisiert werden.	8. April 2025
Greengrass CLI v2.14.3 veröffentlicht	Die Greengrass CLI-Komponente v2.14.3 ist verfügbar.	8. April 2025

Secure Tunneling v1.1.2 veröffentlicht	Secure Tunneling v1.1.2 ist verfügbar. Diese Version aktualisiert den zugrunde liegenden AWS IoT Geräteclient, der von der Komponente aufgerufe n wird, von Version 1.9.0 auf Version 1.10.0. Secure Tunneling v1.1.2 behebt auch das Problem der Nutzlastü bertragung, das Benutzer daran hindert, große Dateien von Greengrass V2-Kernge räten über den sicheren Tunnel an das Quellgerät weiterzuleiten.	27. März 2025
Greengrass CLI v2.14.2 veröffentlicht	Die Greengrass CLI-Kompo nente v2.14.2 ist verfügbar.	27. März 2025
AWS IoT Greengrass Softwareupdate Core v2.14.2	Diese Version stellt Version 2.14.2 der Greengrass Nucleus-Komponente und mit Updates AWS bereitgestellte Komponenten bereit.	27. März 2025
Secret Manager v2.2.3 veröffentlicht	Secret Manager v2.2.3 ist verfügbar. Diese Version behebt ein Problem, bei dem Secret Manager die lokal gespeicherten Geheimnisse löscht, wenn das Kerngerät offline ist und der Sicherhei tsdienst des Geräts (z. B. ein HSM) nicht verfügbar ist.	18. März 2025

Stream Manager v2.2.1 veröffentlicht	Stream Manager v2.2.1 ist jetzt verfügbar. Diese Version behebt ein Problem, bei dem der Stream Manager keine Nachrichten an Kinesis-Z iele exportiert. Diese Version verbessert auch die Leistung von Stream Manager-Exporten zu Kinesis-Zielen.	12. März 2025
Die Authentifizierungs komponente für Client-Geräte, Version 2.5.3 wurde veröffent licht	Die Authentifizierungs komponente für Client-Ge räte v2.5.3 ist verfügbar. Diese Version behebt ein Problem, bei dem Client-Ge räte aufgrund veralteter Client-Zertifikate keine Verbindun g zum Kerngerät herstellen können.	12. März 2025
AWS IoT Das Fleet Provision ing Plugin v1.2.2 wurde veröffentlicht	AWS IoT Das Fleet Provision ing Plugin v1.2.2 ist verfügbar . Diese Version bietet Unterstützung für benutzerd efinierte Pfade für Zertifikate (`certificatePath`) und private Schlüssel (`privateKe yPath`) für Kerngeräte.	12. März 2025
Secure Tunneling v1.1.1 veröffentlicht	Secure Tunneling v1.1.1 ist verfügbar. Diese Version fügt eine Konfiguration zur Unterstützung von Greengrass Nucleus Lite hinzu.	7. Februar 2025
Greengrass CLI v2.14.1 veröffentlicht	Die Greengrass CLI-Kompo nente v2.14.1 ist verfügbar.	7. Februar 2025

AWS IoT Greengrass Softwareupdate Core v2.14.1	Diese Version stellt Version 2.14.1 der Greengrass Nucleus-Komponente und mit Updates AWS bereitgestellte Komponenten bereit.	7. Februar 2025
Greengrass Nucleus Lite-Komponente v2.0.2 veröffentlicht	Version 2.0.2 der Greengrass Nucleus Lite-Komponente ist verfügbar. Diese Version enthält allgemeine Bugfixes und Verbesserungen.	6. Februar 2025
Greengrass Nucleus Lite-Komponente v2.0.1 veröffentlicht	Version 2.0.1 der Greengrass Nucleus Lite-Komponente ist verfügbar. Diese Version enthält allgemeine Bugfixes und Verbesserungen.	28. Januar 2025
Greengrass CLI v2.14.0 veröffentlicht	Die Greengrass CLI-Komponente v2.14.0 ist verfügbar.	24. Dezember 2024
Stream Manager v2.2.0 veröffentlicht	Stream Manager v2.2.0 ist jetzt verfügbar.	16. Dezember 2024
Shadow Manager v2.3.10 veröffentlicht	Shadow Manager v2.3.10 ist verfügbar.	16. Dezember 2024
Secure Tunneling v1.1.0 veröffentlicht	Secure Tunneling v1.1.0 ist verfügbar. Diese Version fügt Rezeptunterstützung für Greengrass Nucleus Lite hinzu.	16. Dezember 2024
Neue Nukleus-Telemetrie-Emitter-Komponente	Version 1.0.10 der Nucleus-Telemetrie-Emitter-Komponente ist verfügbar.	16. Dezember 2024

Modbus-RTU-Protokolladapter v2.1.10 veröffentlicht	Die Modbus-RTU-Protokolladapter-Komponente v2.1.10 ist verfügbar.	16. Dezember 2024
Log Manager v2.3.9 veröffentlicht	Die Log Manager-Komponente v2.3.9 ist verfügbar.	16. Dezember 2024
Lokale Debug-Konsole v2.4.4 veröffentlicht	Die lokale Debug-Konsolenkomponente v2.4.4 ist verfügbar. Diese Version enthält allgemeine Bugfixes und Verbesserungen.	16. Dezember 2024
Lambda Manager v2.3.5 veröffentlicht	Die Lambda-Manager-Komponente v2.3.5 ist verfügbar.	16. Dezember 2024
IP-Detektor v2.2.1 veröffentlicht	Die IP-Detektor-Komponente v2.2.1 ist verfügbar.	16. Dezember 2024
Greengrass Nucleus Lite-Komponente v2.0.0 veröffentlicht	Version 2.0.0 der Greengrass Nucleus Lite-Komponente ist verfügbar. Dies ist die erste Version.	16. Dezember 2024
Disk Spooler v1.0.5 veröffentlicht	Die Disk Spooler-Komponente v1.0.5 ist verfügbar.	16. Dezember 2024
Die Komponente zur Authentifizierung von Client-Geräten v2.5.2 wurde veröffentlicht	Die Authentifizierungskomponente für Client-Geräte v2.5.2 ist verfügbar.	16. Dezember 2024

AWS IoT Greengrass Softwareupdate Core v2.14.0	Diese Version enthält Version 2.14.0 der Greengrass Nucleus-Komponente und neue AWS IoT Greengrass Nucleus Lite-Updates. AWS IoT Greengrass Nucleus Lite ist eine neue Runtime, die für AWS IoT Greengrass Version 2 verfügbar ist. Sie bietet eine Alternative mit reduziert em Speicherbedarf. Dies ist eine gute Option für Geräte mit begrenzten Ressourcen. Es implementiert einen Teil der Nucleus-Funktionalität mit verbesserter Funktions kompatibilität, die für future Versionen geplant ist. Der Quellcode ist ab sofort auf Github verfügbar. Mit der Nucleus Lite Runtime können Sie: • Stellen Sie Komponenten auf Greengrass-Kernger äten bereit. Verwenden Sie dasselbe Rezeptformat, obwohl einige erweiterte Funktionen möglicherweise noch nicht verfügbar sind. • Anwendungen, die als Greengrass-Kompone nten bereitgestellt werden, können das Gerät verwenden SDKs , um auf den unterstützten Greengras	16. Dezember 2024

s-IPC zuzugreifen APIs, z. B.: AWS IoT Core MQTT-Zugriff, lokaler Pub/Sub- und Greengrass-Konfigurationszugriff. Eine Liste der unterstützten IPCs finden Sie in der Kompatibilitätstabelle. APIs

- Einige AWS verwaltete Komponenten wurden für die Unterstützung von Nucleus Lite aktualisiert. Eine Liste vorhandener kompatibler Komponenten finden Sie unter den AWS bereitgestellten Komponenten.

Neue Funktionen:

- Benötigt weniger Arbeitsspeicher und Festplattenspeicher (weniger als 5 MB RAM und weniger als 5 MB Speicher).
- Die Komponenten sind in den Service Manager des Hostsystems integriert (systemd für derzeit unterstützte Linux-Plattformen).

Dinge, auf die Sie achten sollten:

- AWS IoT Greengrass Bei Nucleus Lite-Rezepten wird Groß- und Kleinschreibung beachtet. Stellen Sie sicher, dass das richtige (Schlüssel-) Gehäuse wie in der https://docs.aws.amazon.com/greengrass/v2/developerguide/component-recipe-reference.html Rezeptreferenz verwendet wird.
- Die Nucleus Lite-Runtime unterstützt Dinggruppen-Bereitstellungen und unterstützt noch nicht den Bereitstellungszieltyp (einzelnes) Core-Gerät. Verwenden Sie für die Bereitstellung auf einem einzelnen Greengrass-Gerät eine Dinggruppe, in der nur dieses eine Gerät enthalten ist.
- Die Nucleus Lite-Runtime verwendet begrenzte Speicherressourcen. Funktionen, die je nach Nutzung in der klassischen Runtime skaliert werden, können aufgrund der Überschreitung der auf Lite verfügbaren Ressourcen fehlschlagen. Dies beinhaltet eine aktuelle Beschränkung auf maximal 50 MQTT-Abonnements gleichzeitig

und Höchstgrenzen für die Größe und Bereitstellung von Rezeptdateien. Einige dieser Grenzwerte können zur Kompilierzeit konfiguriert werden, wenn Sie die Lite-Runtime selbst kompilieren.

- Die Nucleus Lite-Runtime ist nicht im Lieferumfang von Java enthalten. Um Komponenten verwenden zu können, die Java erfordern, muss Java auf dem System bereits installiert sein, oder es kann eine Komponent e zur Installation von Java verwendet werden.
- Wir empfehlen, die Nucleus Lite-Runtime aus dem Quellcode zu kompilieren und Ihren eigenen Build zu verwenden, der auf Ihr System zugeschnitten ist. Für Yocto-Systeme ist eine Ebene verfügbar, um die Nucleus Lite-Runtime in Ihr System-Image zu integrier en.
- Derzeit geht Nucleus Lite von einem Linux-Sys tem aus, das Systemd verwendet, oder von einem Container-Image, das Systemd verwendet.
- Sie können Docker-Co ntainer zwar mit Rezeptskr

ipten verwalten, von Greengrass verwaltete Container-Artefakte sind jedoch noch nicht verfügbar.

- Die Nucleus Lite-Runtime bietet noch keine Unterstützung für Schlüssel, die in einem PKCS11 Modul gespeichert sind. Wenn Ihr Anwendungsfall Schlüssel erfordert, die auf einem sicheren Element gespeichert sind, kann die klassische Runtime diesen Anwendungsfall derzeit unterstützen. Stellen Sie sicher, dass die Produktionsgeräte die vollständige Festplattenverschlüsselung verwenden, um zu verhindern, dass Ihre Geräteanmeldeinformationen durchsickern.

Neben der Einführung von Nucleus Lite veröffentlichen wir auch Nucleus v2.14.0. Dieses Update bringt erhebliche Verbesserungen für den bestehenden Greengrass-Kern.

Die wichtigsten Funktionen und Verbesserungen:

- Die neue Unterstützung von Dual-Stack-Endpunk

	ten ermöglicht die IPv6 Netzwerkkommunikation. • Verbesserte Widerstandsfähigkeit gegen Nucleus-Neustartfehler und Verzeichnisbeschädigung. • Speicherlecks beim Schließen von PubSub IPC-Abonnements wurden behoben.	
Stream Manager v2.1.13 veröffentlicht	Stream Manager v2.1.13 ist jetzt verfügbar. Diese Version bietet Unterstützung für den FIPS-Endpunkt für AWS IoT SiteWise	26. August 2024
Shadow Manager v2.3.9 veröffentlicht	Shadow Manager v2.3.9 ist verfügbar.	26. August 2024
Modbus-RTU-Protokolladapter v2.1.9 veröffentlicht	Die Modbus-RTU-Protokolladapter-Komponente v2.1.9 ist verfügbar.	26. August 2024
Log Manager v2.3.8 veröffentlicht	Die Log Manager-Komponente v2.3.8 ist verfügbar.	26. August 2024
Lokale Debug-Konsole v2.4.3 veröffentlicht	Die lokale Debug-Konsolenkomponente v2.4.3 ist verfügbar. Diese Version enthält allgemeine Bugfixes und Verbesserungen.	26. August 2024
Disk Spooler v1.0.4 veröffentlicht	Die Disk Spooler-Komponente v1.0.4 ist verfügbar.	26. August 2024

AWS IoT Greengrass Softwareupdate Core v2.13.0	Diese Version stellt Version 2.13.0 der Greengrass Nucleus-Komponente bereit.	26. August 2024
Neue Nukleus-Telemetrie-Emitter-Komponente	Version 1.0.9 der Nucleus-Telemetrie-Emitter-Komponente ist verfügbar.	23. August 2024
Lambda Manager v2.3.4 veröffentlicht	Die Lambda-Manager-Komponente v2.3.4 ist verfügbar.	23. August 2024
Greengrass CLI v2.13.0 veröffentlicht	Die Greengrass CLI-Komponente v2.13.0 ist verfügbar.	23. August 2024
Die Authentifizierungskomponente für Client-Geräte v2.5.1 wurde veröffentlicht	Die Authentifizierungskomponente für Client-Geräte v2.5.0 ist verfügbar. Diese Version bietet Unterstützung für FIPS-Endpunkte.	23. August 2024
Validierung des Rezepts	Es wurde eine Funktion zur Rezeptvalidierung hinzugefügt, die ein Komponentenrezept bei der Erstellung einer Komponentenversion validiert.	15. August 2024
IP-Detektor v2.2.0 veröffentlicht	Die IP-Detektor-Komponente v2.2.0 ist verfügbar. Diese Version bietet Unterstützung für. IPv6 Sie können es jetzt IPv6 für lokales Messaging verwenden.	29. Juli 2024

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 GGV2 Q 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 Bugfixes und Verbesserungen.	29. April 2024
Greengrass CLI v2.12.5 veröffentlicht	Die Greengrass CLI-Komponente v2.12.5 ist verfügbar.	25. April 2024
Authentifizierungskomponente für Client-Geräte v2.5.0 veröffentlicht	Die Authentifizierungskomponente für Client-Geräte v2.5.0 ist verfügbar. Diese Version bietet Unterstützung für Richtlinienvariablen für Dingnamen. Diese Version ermöglicht auch Richtlinienressourcen mit Platzhaltern.	25. April 2024

AWS IoT Greengrass Core v2.12.5 Softwareupdate	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 Q v2.5.3 veröffentlicht GGV2	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-Kompo nente v2.12.3 ist verfügbar.	25. März 2024
AWS IoT Device Tester v4.9.2 mit Q v2.5.2 veröffentlicht GGV2	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
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-Eigenschaft en, 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 Bugfixes 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
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

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
Authentifizierungskomponente für Client-Geräte v2.4.5 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 Greengras s Development Kit CLI ist verfügbar. Diese Version enthält Fehlerkorrekturen und Verbesserungen.	6. Dezember 2023
Validierung des Rezepts	Es wurde eine Funktion zur Rezeptvalidierung hinzugefü gt, die ein Komponentenrezept bei der Erstellung einer Komponentenversion validiert.	16. November 2023
Vom Publisher unterstützte Komponenten	AWS IoT Greengrass bietet jetzt von Publisher unterstüt zte Komponenten an. Diese Komponenten werden von Drittanbietern entwickelt, angeboten und gewartet.	16. November 2023
Greengrass Testing Framework v1.2.0 veröffent licht	Greengrass Testing Framework v1.2.0 ist verfügbar.	15. November 2023

Greengrass-Entwicklungskit CLI v1.6.0	Version 1.6.0 des Greengras s 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 Komponent enrezepten zu einem früheren Zeitpunkt des Komponent enerstellungsprozesses zu identifizieren. Diese Version fügt der Vorlage außerdem eine Suite für Sicherhei tstests hinzu, die mit dem `test-e2e init` Befehl abgerufen werden kann. Diese Konfidenz-Testsuit e umfasst acht generisch e Tests, die verwendet und erweitert werden können, um den grundlegenden Anforderu ngen 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 unterstüt zt 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 Greengras s-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-Kompo nente 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-Proxykonfigu ration 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
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
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 Softwareupdate Core 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 AI Edge Manager wurde eingestellt	Die Amazon SageMaker AI 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 Bugfixes.	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
Authentifizierungskomponente für Client-Geräte v2.3.2 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	Aktualisierung des Leitfadens zur Ausrichtung an bewährten IAM-Methoden. Weitere Informationen finden Sie unter Bewährte Sicherheitsmethoden in IAM.	3. Februar 2023

AWS IoT Greengrass Softwareupdate Core 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 Q v2.5.0 veröffentlicht GGV2	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 Greengras s 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 Schatteng rösse 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 Bereitste llung RESET auf den Konfigura tionsknoten ausgeführt wird.	8 Dezember 2022
Log Manager v2.3.0 veröffent licht	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 Q v2.4.1 veröffentlicht GGV2	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 CloudFormation	CloudFormation unterstützt jetzt AWS IoT Greengrass Bereitstellungen als Ressource.	6. Oktober 2022

SageMaker AI Edge Manager v1.3.0 veröffentlicht	Die Amazon SageMaker AI Edge Manager-Komponente v1.3.0 ist verfügbar. Diese Version bietet Unterstützung für diese Komponente zum Festlegen der Festplattengröße für den TensorRT-Modellcache und verbessert die Parallelität von Vorhersagen, um Gerätebeschleuniger-Engines wie GPUs besser zu nutzen.	01. September 2022
Verwenden Sie den IPC-Client (Interprocess Communication) V2	Es wurden Informationen über den IPC-Client V2 hinzugefügt, wodurch die Menge an Code reduziert wird, die Sie für die Verwendung von IPC-Operationen 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 Q v2.4.0 veröffentlicht GGV2	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 AI Edge Manager v1.2.0 veröffentlicht	Die Amazon SageMaker AI Edge Manager-Komponente v1.2.0 ist verfügbar. Diese Version bietet Unterstützung für diese Komponente zum automatischen Abrufen von SageMaker KI-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 publish/subscribe Themen, 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 Bugfixes 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 GGV2 Q v2.3.1 aktualisiert und IDT v4.5.1 mit Q v2.3.0 zu den unterstützten Versionen hinzugefügt GGV2	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 AWS IoT Greengrass V2 Qualification Suite (Q) v2.3.0 als unterstützte Version. GGV2 IDT 4.5.1 mit AWS IoT Greengrass 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-Protokolladapter-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 GGV2 Q 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 Q v2.3.0 veröffentlicht GGV2	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 Q v2.2.1 veröffentlicht GGV2	Version 4.4.1 von IDT für V2 ist verfügbar. AWS IoT Greengrass Diese Version enthält 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 GGV2 Q v2.2.0 veröffentlicht	Version 4.4.1 von IDT für V2 ist verfügbar. AWS IoT Greengrass Diese Version enthält 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 GGV2 Q 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 AI Edge Manager v1.1.0 veröffentlicht	Die Amazon SageMaker AI 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 vermeiden.	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 über die Endpunkte 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 Q v2.0.1 veröffentlicht GGV2	Version 4.2.0 von IDT für AWS IoT Greengrass V2 wurde mit der AWS IoT Greengrass V2 Qualification Suite (Q) v2.0.1 aktualisiert. GGV2 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 Q v2.0.0 veröffentlicht GGV2	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 für 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 AI Edge Manager-Komponente v1.0.2 wurde veröffentlicht	Die Amazon SageMaker AI Edge Manager-Komponente v1.0.2 ist verfügbar. Diese Version 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.	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 Q v1.1.1 veröffentlicht GGV2	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 Softwareupdate Core v2.2.0	Diese Version enthält Version 2.2.0 der Greengras s Nucleus-Komponente und von Updates AWS bereitges tellte 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 Leistungsverbesser ungen und Fehlerkorrekturen.	13. Juni 2021
Neue SageMaker AI Edge Manager-Komponente veröffentlicht	Version 1.0.0 der Amazon SageMaker AI Edge Manager-Komponente ist verfügbar für AWS IoT Greengrass. Diese Komponente installie rt die SageMaker AI Edge Manager-Agent-Binärdatei auf Greengrass-Core-Geräten.	10. Juni 2021
Typen von Komponenten	Informationen zu Komponent entypen wurden in hinzugefü gt AWS IoT Greengras s. 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 GGV2 Q 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.	1. 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 China 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 Namen und die Greengrass Nucleus-Version abzurufen. AWS-Region	23. Februar 2021

Manuelle Installation	Es wurden Informationen zur manuellen Erstellung der erforderlichen AWS Ressourcen oder zur Installation hinter einer Firewall oder einem Netzwerk-Proxy hinzugefügt. 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 Softwareupdate Core v2.0.4	Diese Version stellt Version 2.0.4 der Greengrass Nucleus-Komponente bereit. Sie enthält den neuen `greengrassDataPlanePort` 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 Geräteval idierung 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 Bereitste llungen. Diese Funktionen erleichtern Ihnen die Entwicklu ng 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.