



Guida per gli sviluppatori, versione 1

AWS IoT Greengrass



AWS IoT Greengrass: Guida per gli sviluppatori, versione 1

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

I marchi e l'immagine commerciale di Amazon non possono essere utilizzati in relazione a prodotti o servizi che non siano di Amazon, in una qualsiasi modalità che possa causare confusione tra i clienti o in una qualsiasi modalità che denigri o discrediti Amazon. Tutti gli altri marchi non di proprietà di Amazon sono di proprietà delle rispettive aziende, che possono o meno essere associate, collegate o sponsorizzate da Amazon.

Table of Contents

.....	xxi
Che cos'è AWS IoT Greengrass?	1
AWS IoT Greengrass Software di base	3
AWS IoT Greengrass Versioni software principali	4
AWS IoT Greengrass gruppi	14
Dispositivi in AWS IoT Greengrass	16
SDKs	19
Piattaforme supportate e requisiti	20
AWS IoT Greengrass download	34
AWS IoT Greengrass Software di base	34
AWS IoT Greengrass software snap	42
AWS IoT Greengrass Software Docker	43
AWS IoT Greengrass SDK principale	45
Runtime e librerie di Machine Learning supportati	46
AWS IoT Greengrass Software ML SDK	47
Ci piacerebbe conoscere la tua opinione	47
Installare il software AWS IoT Greengrass Core	47
Scaricare ed estrarre un file tar.gz	48
Eseguire lo script di installazione del dispositivo Greengrass	48
Installazione da un repository APT	48
Esegui AWS IoT Greengrass in un contenitore Docker	51
Corri AWS IoT Greengrass in un attimo	51
Archiviazione di un'installazione del software core	63
Configura il AWS IoT Greengrass core	65
AWS IoT Greengrass file di configurazione principale	66
Gli endpoint del servizio devono corrispondere al tipo di certificato	128
Connessione alla porta 443 o tramite un proxy di rete	130
Configurazione di una directory di scrittura	140
Configurazione delle impostazioni MQTT	144
Attivazione del rilevamento automatico dell'IP	162
Lancio di Greengrass all'avvio del sistema	166
Consulta anche	167
AWS IoT Greengrass V1 politica di manutenzione	168
AWS IoT Greengrass schema di controllo delle versioni	168

Fasi del ciclo di vita del software Core AWS IoT Greengrass	169
Politica di manutenzione per AWS IoT Greengrass il software Core	169
Pianificazione della fase di manutenzione	170
Pianificazione della rinuncia	170
Politica di supporto per le funzioni Lambda	170
Politica di supporto per AWS IoT Device Tester per AWS IoT Greengrass V1	171
Fine del programma di manutenzione	171
Fine della manutenzione per le immagini AWS IoT Greengrass Docker del software Core v1.x	43
Fine della manutenzione del repository APT del software Core v1.x AWS IoT Greengrass ..	173
Fine della manutenzione per il software AWS IoT Greengrass Core v1.11.x Snap	173
Iniziare con AWS IoT Greengrass	174
Scopri come iniziare	174
Requisiti	177
Crea un Account AWS	178
Registrati per un Account AWS	179
Crea un utente con accesso amministrativo	179
Avvio rapido: configurazione dispositivo Greengrass	181
Requisiti	181
Esecuzione della configurazione del dispositivo Greengrass	182
Risoluzione dei problemi	186
Opzioni di configurazione del dispositivo Greengrass	187
Modulo 1: impostazione dell'ambiente per Greengrass	197
Impostazione di un Raspberry Pi	198
Configurazione di un' EC2 istanza Amazon	206
Configurazione di altri dispositivi	212
Modulo 2: Installazione del software AWS IoT Greengrass Core	215
Fornitura di AWS IoT qualsiasi cosa da utilizzare come core Greengrass	216
Crea un gruppo Greengrass	220
Installa ed esegui AWS IoT Greengrass sul dispositivo principale	221
Modulo 3 (parte 1): funzioni Lambda su AWS IoT Greengrass	228
Creare e impacchettare una funzione Lambda	228
Configurare la funzione Lambda per AWS IoT Greengrass	234
Distribuire configurazioni cloud su un dispositivo core	237
Verifica che la funzione Lambda sia in esecuzione sul dispositivo principale	238
Modulo 3 (parte 2): funzioni Lambda su AWS IoT Greengrass	239

Crea e impacchetta la funzione Lambda	240
Configura funzioni Lambda di lunga durata per AWS IoT Greengrass	244
Prova le funzioni Lambda di lunga durata	245
Testa le funzioni Lambda su richiesta	248
Modulo 4: Interazione con i dispositivi client in un gruppo AWS IoT Greengrass	252
Creare dispositivi client in un AWS IoT Greengrass gruppo	254
Configurazione delle sottoscrizioni	257
Installa il SDK per dispositivi AWS IoT for Python	258
Test delle comunicazioni	264
Modulo 5: interazione con dispositivi ombra	269
Configurazione di dispositivi e sottoscrizioni	270
Scarica i file richiesti	273
Test delle comunicazioni (sincronizzazione dispositivi disattivata)	273
Test delle comunicazioni (sincronizzazione dispositivi attivata)	277
Modulo 6: Accesso ad altri AWS servizi	278
Configurazione del ruolo del gruppo	280
Creare e configurare la funzione Lambda	282
Configurazione delle sottoscrizioni	285
Test delle comunicazioni	286
Modulo 7: simulazione dell'integrazione di sicurezza hardware	288
Installazione di SoftHSM	289
Configurazione di SoftHSM	289
Importazione della chiave privata	290
Configurazione del core Greengrass	292
Test della configurazione	295
Consulta anche	296
Aggiornamenti OTA del software AWS IoT Greengrass Core	297
Requisiti	297
Autorizzazioni IAM per gli aggiornamenti OTA	299
Considerazioni	301
Agente di aggiornamento OTA di Greengrass	302
Integrazione con i sistemi di inizializzazione	303
Rigenerazione gestita con aggiornamenti OTA	303
Creare un aggiornamento OTA.	305
CreateSoftwareUpdateJob API	309
Distribuisci gruppi AWS IoT Greengrass	312

Distribuzione di gruppi (console)	313
Distribuzione di gruppi (API)	314
Ottenere l'ID del gruppo	316
Panoramica del modello di oggetti del gruppo	317
Gruppi	317
Versioni del gruppo	318
Componenti del gruppo	319
Aggiornamento dei gruppi	320
Consulta anche	321
Ottenere le notifiche di distribuzione	322
Evento di modifica dello stato della distribuzione del gruppo	323
Prerequisiti per la creazione di regole EventBridge	324
Configurazione delle notifiche di distribuzione (console)	325
Configurazione delle notifiche di distribuzione (CLI)	326
Configurazione delle notifiche di distribuzione (AWS CloudFormation)	327
Consulta anche	327
Reimpostazione delle distribuzioni	327
Reimposta le distribuzioni dalla console AWS IoT	328
Reimposta le distribuzioni con l'API AWS IoT Greengrass	329
Consulta anche	330
Creazione di distribuzioni in blocco	330
Prerequisiti	331
Creazione e caricamento del file di input della distribuzione di massa	331
Crea e configura un ruolo di esecuzione IAM per distribuzioni di massa	334
Autorizzazione per l'accesso al bucket S3 da parte del ruolo di esecuzione	337
Distribuzione dei gruppi	338
Test della distribuzione	341
Risoluzione dei problemi di distribuzione di massa	342
Consulta anche	345
Esegui funzioni Lambda locali	346
SDKs	347
Migrazione delle funzioni Lambda basate sul cloud	350
Riferimento alle funzioni in base all'alias o alla versione	351
Controllo dell'esecuzione della funzione Greengrass Lambda	352
Impostazioni di configurazione specifiche del gruppo	352
Esecuzione di una funzione Lambda come utente root	356

Considerazioni sulla scelta della containerizzazione delle funzioni Lambda	358
Impostazione dell'identità di accesso predefinita per le funzioni Lambda in un gruppo	362
Impostazione della containerizzazione predefinita per le funzioni Lambda in un gruppo	363
Flussi di comunicazione	364
Comunicazione tramite messaggi MQTT	364
Altri flussi di comunicazione	365
Recupero dell'argomento di input (o oggetto)	366
Configurazione del ciclo di vita	368
Eseguibili Lambda	370
Creare un file eseguibile Lambda	371
Esegui AWS IoT Greengrass in un contenitore Docker	373
Prerequisiti	374
Ottieni l'immagine del AWS IoT Greengrass contenitore da Amazon ECR	376
Creazione e configurazione del gruppo e del core Greengrass	379
Esegui AWS IoT Greengrass localmente	379
Configurazione della containerizzazione "No container" per il gruppo	383
Distribuisci le funzioni Lambda nel contenitore Docker	384
(Facoltativo) Implementa i dispositivi client che interagiscono con Greengrass nel contenitore Docker	384
Arresto del contenitore Docker AWS IoT Greengrass	384
AWS IoT Greengrass Risoluzione dei problemi in un contenitore Docker	384
Accesso alle risorse locali	388
Tipi di risorse supportati	388
Requisiti	390
Risorse di volume nella directory /proc	390
Autorizzazione di accesso ai file dell'owner del gruppo	391
Consulta anche	391
Utilizzo della CLI	391
Creazione di risorse locali	392
Creazione della funzione Greengrass	394
Aggiungere la funzione Lambda al gruppo	395
Risoluzione dei problemi	397
Utilizzo della console	399
Prerequisiti	399
Creare un pacchetto di distribuzione della funzione Lambda	400
Creare e pubblicare una funzione Lambda	401

Aggiungere la funzione Lambda al gruppo	404
Aggiungere una risorsa locale al gruppo	405
Aggiunta di sottoscrizioni al gruppo	406
Distribuzione del gruppo.	407
Test dell'accesso alle risorse locali	408
Esecuzione dell'inferenza di Machine Learning	412
Come funziona AWS IoT Greengrass l'inferenza ML	412
Risorse di Machine Learning	413
Origini di modello supportate	413
Requisiti	416
Runtime e librerie per inferenza ML	416
SageMaker Runtime di deep learning AI Neo	417
MXNet controllo delle versioni	417
MXNet su Raspberry Pi	417
TensorFlow limitazioni relative al servizio dei modelli su Raspberry Pi	418
Accesso alle risorse di Machine Learning	418
Autorizzazioni di accesso per risorse di Machine Learning	419
Definizione delle autorizzazioni di accesso per le funzioni Lambda (console)	422
Definizione delle autorizzazioni di accesso per le funzioni Lambda (API)	423
Accesso alle risorse di machine learning dal codice della funzione Lambda	426
Risoluzione dei problemi	427
Consulta anche	429
Come configurare l'inferenza di Machine Learning	429
Prerequisiti	430
Configurare il dispositivo Raspberry Pi	431
Installa il MXNet framework	433
Creazione di un pacchetto del modello	433
Creare e pubblicare una funzione Lambda	434
Aggiungere la funzione Lambda al gruppo	437
Aggiunta di risorse al gruppo	439
Aggiunta di una sottoscrizione al gruppo	442
Distribuzione del gruppo.	442
Esecuzione del test dell'app	444
Passaggi successivi	448
Configurazione di un dispositivo Intel Atom	448
Configurazione di un NVIDIA Jetson TX2	451

Come configurare l'inferenza di Machine Learning ottimizzata	456
Prerequisiti	430
Configurare il dispositivo Raspberry Pi	458
Installazione di Neo Deep Learning Runtime	460
Creare una funzione Lambda di inferenza	460
Aggiungere la funzione Lambda al gruppo	464
Aggiunta della risorsa del modello ottimizzato Neo al gruppo	466
Aggiunta della risorsa del dispositivo della telecamera al gruppo	468
Aggiunta di sottoscrizioni al gruppo	470
Distribuzione del gruppo.	470
Test dell'esempio	471
Configurazione di un dispositivo Intel Atom	472
Configurazione di un NVIDIA Jetson TX2	475
Risoluzione dei problemi di AWS IoT Greengrass inferenza ML	445
Passaggi successivi	482
Gestione dei flussi di dati	483
Flusso di lavoro della gestione dei flussi	484
Requisiti	486
Sicurezza dei dati	487
Sicurezza dei dati locali	487
Autenticazione client	488
Consulta anche	489
Configurazione di Stream Manager	489
Parametri di Stream Manager	489
Configurazione delle impostazioni (console)	492
Configurazione delle impostazioni (CLI)	495
Consulta anche	506
Utilizzalo StreamManagerClient per lavorare con gli stream	506
Creazione del flusso di messaggi	507
Aggiunta di un messaggio	512
Lettura di messaggi	518
Visualizzazione dell'elenco di flussi	521
Descrizione del flusso di messaggi	522
Aggiorna il flusso di messaggi	525
Eliminazione del flusso di messaggi	529
Consulta anche	531

Esporta le configurazioni per le destinazioni supportate Cloud AWS	531
Esportazione di flussi di dati (Console)	549
Prerequisiti	549
Creare un pacchetto di distribuzione della funzione Lambda	552
Creazione di una funzione Lambda	555
Aggiunta di una funzione al gruppo	557
Abilitazione di Stream Manager	558
Configurazione della registrazione locale	559
Distribuzione del gruppo.	559
Eseguire il test dell'applicazione	561
Consulta anche	562
Esportazione di flussi di dati (CLI)	562
Prerequisiti	563
Creare un pacchetto di distribuzione della funzione Lambda	566
Creazione di una funzione Lambda	570
Creazione della versione e della definizione della funzione	572
Creazione di una definizione e di una versione di logger	574
Ottenimento dell'ARN della versione di definizione del core	575
Creazione di una versione del gruppo	576
Crea distribuzione	577
Eseguire il test dell'applicazione	578
Consulta anche	580
Distribuzione dei segreti nel core	581
Crittografia dei segreti	582
Requisiti	583
Specificare la chiave privata per la crittografia dei segreti	584
AWS IoT Greengrass Consenti di ottenere valori segreti	586
Consulta anche	587
Utilizzo delle risorse segrete	588
Creazione e gestione di segreti	588
Utilizzo dei segreti locali	593
Come creare una risorsa segreta (console)	596
Prerequisiti	597
Crea un segreto di Secrets Manager	598
Aggiunta di una risorsa segreta a un gruppo	599
Creare un pacchetto di distribuzione della funzione Lambda	600

Creazione di una funzione Lambda	602
Aggiunta della funzione al gruppo	604
Collegamento della risorsa segreta alla funzione	606
Aggiunta di sottoscrizioni al gruppo	606
Distribuzione del gruppo.	607
Test della funzione Lambda	608
Consulta anche	609
Integrazione con servizi e protocolli tramite i connettori	610
Requisiti	611
Utilizzo dei connettori Greengrass	612
Parametri di configurazione	614
Parametri utilizzati per l'accesso alle risorse di gruppo	615
Aggiornamento dei parametri del connettore	615
Input e output	616
Argomenti di input	616
Supporto per la containerizzazione	617
Aggiornamento delle versioni dei connettori	618
Registrazione	619
AWS-connettori Greengrass forniti	619
CloudWatch Metriche	622
Device Defender	639
Distribuzione dell'applicazione Docker	645
IoT Analytics	688
Adattatore di protocollo IP Ethernet IoT	705
IoT SiteWise	710
Kinesis Firehose	726
Feedback ML	745
Classificazione delle immagini ML	762
Rilevamento di oggetti ML	789
Adattatore di protocollo Modbus-RTU	806
Adattatore di protocollo Modbus-TCP	825
Raspberry Pi GPIO	831
Stream seriale	842
ServiceNow MetricBase Integrazione	855
SNS	871
Integrazione Splunk	883

Notifiche Twilio	897
Nozioni di base sui connettori (console)	914
Prerequisiti	916
Crea un segreto di Secrets Manager	916
Aggiunta di una risorsa segreta a un gruppo	918
Aggiunta di un connettore al gruppo	918
Creare un pacchetto di distribuzione della funzione Lambda	919
Creazione di una funzione Lambda	921
Aggiunta di una funzione al gruppo	923
Aggiunta di sottoscrizioni al gruppo	923
Distribuzione del gruppo.	924
Test della soluzione	925
Consulta anche	927
Nozioni di base sui connettori (CLI)	927
Prerequisiti	929
Crea un segreto di Secrets Manager	930
Creazione della versione e della definizione della risorsa	931
Creazione della versione e della definizione del connettore	932
Creare un pacchetto di distribuzione della funzione Lambda	933
Creazione di una funzione Lambda	935
Creazione della versione e della definizione della funzione	937
Creazione della versione e della definizione dell'abbonamento	938
Creazione di una versione del gruppo	939
Crea distribuzione	941
Test della soluzione	942
Consulta anche	943
API Greengrass Discovery RESTful	945
Richiesta	945
Risposta	946
Autorizzazione alla scoperta	947
Documenti di risposta di individuazione di esempio	947
Sicurezza	950
Panoramica della AWS IoT Greengrass sicurezza	951
Flusso di lavoro di connessione del dispositivo	952
Configurazione della sicurezza AWS IoT Greengrass	953
Principal di sicurezza	954

Sottoscrizioni gestite nel flusso di lavoro di messaggistica MQTT	957
Supporto TLS per le suite di cifratura	957
Protezione dei dati	960
Crittografia dei dati	961
Integrazione della sicurezza hardware	964
Autenticazione e autorizzazione del dispositivo	985
Certificati X.509	986
AWS IoT politiche	988
AWS IoT Politica minima per il dispositivo principale	991
Gestione dell'identità e degli accessi	995
Destinatari	995
Autenticazione con identità	996
Gestione dell'accesso con policy	999
Consulta anche	1002
Come AWS IoT Greengrass funziona con IAM	1002
Ruolo del servizio Greengrass	1011
Ruolo del gruppo Greengrass	1019
Prevenzione del problema "confused deputy" tra servizi	1029
Esempi di policy basate su identità	1030
Risoluzione dei problemi di identità e accesso	1034
Convalida della conformità	1037
Resilienza	1038
Sicurezza dell'infrastruttura	1039
Analisi della configurazione e delle vulnerabilità	1040
Endpoint VPC (AWS PrivateLink)	1041
Considerazioni sugli endpoint AWS IoT Greengrass VPC	1041
Crea un endpoint VPC di interfaccia per AWS IoT Greengrass le operazioni del piano di controllo	1042
Creazione di una policy per gli endpoint VPC per AWS IoT Greengrass	1042
Best practice di sicurezza	1043
Concedere autorizzazioni minime possibili	1043
Non codificare le credenziali nelle funzioni Lambda	1043
Non registrare informazioni riservate	1044
Creare sottoscrizioni mirate	1044
Tenere sincronizzato l'orologio del dispositivo	1045
Gestione dell'autenticazione dei dispositivi con il core Greengrass	1045

Consulta anche	1046
Registrazione di log e monitoraggio	1047
Strumenti di monitoraggio	1047
Consulta anche	1048
Monitoraggio con i log AWS IoT Greengrass	1048
Accesso ai log CloudWatch	1048
Accesso ai log del file system	1050
Configurazione della registrazione predefinita	1051
Configurare la registrazione per AWS IoT Greengrass	1052
Limitazioni di registrazione	1055
CloudTrail registri	1057
Registrazione delle chiamate AWS IoT Greengrass API con AWS CloudTrail	1057
AWS IoT Greengrass informazioni in CloudTrail	1058
Comprensione delle AWS IoT Greengrass voci dei file di registro	1059
Consulta anche	1062
Raccolta di dati di telemetria sanitaria del sistema	1062
Configurazione delle impostazioni di telemetria	1065
Iscrizione per ricevere dati di telemetria	1070
Risoluzione dei problemi di AWS IoT Greengrass telemetria	1076
Chiamata dell'API di controllo sanitario locale	1077
Ottieni informazioni sanitarie per tutti i lavoratori	1077
Ottieni informazioni sanitarie su determinati lavoratori	1078
Informazioni sulla salute dei lavoratori	1081
Tagging delle risorse Greengrass	1084
Nozioni di base sui tag	1084
Supporto del tagging (console)	1084
Supporto del tagging (API)	1085
Utilizzo dei tag con policy IAM	1087
Policy IAM di esempio	1088
Consulta anche	1090
AWS CloudFormation supporto per AWS IoT Greengrass	1091
Creare risorse	1091
Distribuzione delle risorse	1092
Modello di esempio	1093
Supportato s Regione AWS	1106
Utilizzo di AWS IoT Device Tester per AWS IoT Greengrass V1	1107

AWS IoT Greengrass suite di qualificazione	1107
Suite di test personalizzate	1108
Versioni supportate di AWS IoT Device Tester for AWS IoT Greengrass V1	1108
Versioni IDT non supportate per AWS IoT Greengrass	1109
Usa IDT per gestire la suite di AWS IoT Greengrass qualifiche	1114
Versioni della suite di test	1115
Descrizioni dei gruppi di test	1116
Prerequisiti	1121
Configura il tuo dispositivo per eseguire test IDT	1130
Configurare le impostazioni IDT	1153
Esegui la suite AWS IoT Greengrass di qualificazione	1168
Informazioni su risultati e log	1173
Usa IDT per sviluppare ed eseguire le tue suite di test	1177
Scarica la versione più recente di IDT per AWS IoT Greengrass.	1121
Flusso di lavoro per la creazione della suite	1178
Tutorial: crea ed esegui la suite di test IDT di esempio	1179
Tutorial: Sviluppa una semplice suite di test IDT	1184
Crea file di configurazione della suite di test IDT	1193
Configurare la macchina a stati IDT	1201
Crea eseguibili per test case IDT	1224
Usa il contesto IDT	1232
Configurare le impostazioni per i test runner	1236
Esegui il debug ed esegui suite di test personalizzate	1248
Esamina i risultati e i registri dei test IDT	1251
Metriche di utilizzo IDT	1257
IDT per la risoluzione dei problemi AWS IoT Greengrass	1264
Codici di errore	1264
Risoluzione di IDT per errori AWS IoT Greengrass	1287
Politica di supporto per AWS IoT Device Tester per AWS IoT Greengrass V1	1292
Risoluzione dei problemi	1294
AWS IoT Greengrass Questioni principali	1294
Errore: nel file di configurazione manca il CaPath, CertPath o. KeyPath Il processo daemon	
Greengrass con [pid = <pid>] è terminato.	1296
Errore: impossibile analizzare/<greengrass-root> /config/config.json.	1297
Errore: si è verificato un errore durante la generazione della configurazione TLS:	
ErrUnknown URIScheme	1297

Errore: l'avvio di Runtime non è riuscito: impossibile avviare i worker: test del container scaduto.	1298
<address>Errore: Impossibile richiamare PutLogEvents su Cloudwatch locale, LogGroup:/ GreengrassSystem/connection_manager, errore:: invio della richiesta non riuscito causato da: Post http RequestError://<path>/cloudwatch/logs/: dial tcp: getsockopt: connessione rifiutata, risposta: {}.	1298
<region><account-id><function-name><version><file-name>Errore: impossibile creare il server a causa di: failed to load group: chmod/<greengrass-root>/aws:lambda: :function:/ ggc/deployment/lambda/arn: nessun file o directory di questo tipo.	1299
Il software AWS IoT Greengrass Core non si avvia dopo il passaggio dall'esecuzione senza containerizzazione all'esecuzione in un contenitore Greengrass.	1299
Errore: la dimensione di spool deve essere almeno 262144 byte.	1299
Errore: [ERROR]-Cloud messaging error: si è verificato un errore durante il tentativo di pubblicare un messaggio. {"errorString": "operation timed out"}	1300
<version>Errore: container_linux.go:344: l'avvio del processo del contenitore ha causato «process_linux.go:424: l'init del contenitore ha causato\" rootfs_linux.go:64: mount\\\"\\\\ _ipc.sock: autorizzazione negata\\\" \"\"». greengrass/ggc/socket/greengrass_ipc.sock\\\" to rootfs \\\"/greengrass/ggc/packages rootfs/merged\\\" at \\\"/greengrass_ipc.sock\\\" caused \\ \\\"stat /greengrass/ggc/socket/greengrass	1301
Errore: daemon Greengrass in esecuzione con PID: <process-id>. Alcuni componenti di sistema non sono stati avviati. Controlla se ci sono errori in 'runtime.log'.	1301
La shadow del dispositivo non si sincronizza con il cloud.	1035
ERRORE: non in grado di accettare la connessione TCP. accept tcp [::]:8000: accept4: troppi file aperti.	1302
Errore: errore di esecuzione runtime: impossibile avviare il container lambda. container_linux.go:259: l'avvio del processo del container ha causato "process_linux.go:345: container init caused \"rootfs_linux.go:50: preparing rootfs caused \\\"permission denied\\ \\\"\".	1302
Avviso: [WARN] - [5] GK Remote: errore durante il recupero dei dati della chiave pubblica: la chiave privata per non è ErrPrincipalNotConfigured impostata. MqttCertificate	1302
<account-id><role-name><region>Errore: autorizzazione negata durante il tentativo di utilizzare il ruolo arn:aws:iam: :role/ per accedere a s3 url https://- greengrass-updates.s3. <region><architecture><distribution-version>.amazonaws.com/ core/ /greengrass-core- .tar.gz.	1035
Il AWS IoT Greengrass core è configurato per utilizzare un proxy di rete e la funzione Lambda non può effettuare connessioni in uscita.	1303

Il core si trova in un ciclo infinito di connessione-disconnessione. Il file runtime.log contiene una serie di voci di connessione e disconnessione.	1304
Errore: impossibile avviare il container lambda. container_linux.go: 259: l'avvio del processo del container ha causato "process_linux.go: 345: container init caused\"rootfs_linux.go: 62: mounting \" proc\\\" to rootfs \"	1305
[ERRORE] -errore di esecuzione in fase di esecuzione: impossibile avviare il contenitore lambda. <ggc-path>{"errorString": «impossibile inizializzare i supporti dei contenitori: impossibile mascherare la radice di greengrass nella directory superiore di overlay: impossibile creare il dispositivo maschera nella directory: il file esiste»}	1305
[ERRORE] -Distribuzione non riuscita. {"DeploymentId": <deployment-id>«, «errorString»: «Processo di test del contenitore con <pid>pid non riuscito: stato del processo del contenitore: stato di uscita 1"}	1306
Errore: [ERROR] -Errore di esecuzione di runtime: impossibile avviare il contenitore lambda. <ggc-version><ggc-version>{"errorString": «impossibile inizializzare i supporti del contenitore: impossibile creare l'overlay fs per il contenitore: montaggio dell'overlay su/////<ggc-version><ggc-version><ggc-version>/dns:/greengrass/ggc/packages/rootfs/merged failed: failed to mount with args source=\"no_source\" dest= \"/greengrass/ggc/packagesrootfs/merged\" fstype=\"overlay\" flags=\"0\" data=\"lowerdir=/greengrass/ggc/packages//rootfs/work\\»: troppi livelli di collegamenti upperdir=/greengrass/ggc/packages simbolici \"} rootfs/upper,workdir=/greengrass/ggc/packages	1307
Errore: [DEBUG] - Impossibile ottenere route. Eliminare il messaggio.	1308
Errore: [Errno 24] Troppi file aperti <lambda-function>, [Errno 24] Troppi file aperti	1308
Errore: il server ds non è riuscito ad avviare l'ascolto di socket: listen unix<ggc-path>/ggc/socket/greengrass_ipc.sock: bind: argomento non valido	1308
[INFORMAZIONI] (Copier) aws.greengrass. StreamManager: stdout. Causato da: com.fasterxml.jackson.databind. JsonMappingException: Instant supera l'istante minimo o massimo	1309
GPG error: https://dnw9lb6lzp2d8.cloudfront.net stable InRelease: The following signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass Master Key	1309
Problemi relativi alla distribuzione	1310
La distribuzione corrente non funziona e si desidera ripristinare una distribuzione funzionante precedente.	1311
Visualizzi un errore 403 Forbidden sulla distribuzione nei log.	1313
Si verifica un ConcurrentDeployment errore quando si esegue il comando create-deployment per la prima volta.	1314

Errore: Greengrass non è autorizzato ad assumere il ruolo di servizio associato a questo account, oppure l'errore: Non riuscito: il ruolo di servizio TES non è associato a questo account.	1035
Errore: impossibile eseguire il passaggio di download nella distribuzione. errore durante il download: errore durante il download del file di definizione del gruppo:... x509: il certificato è scaduto o non è ancora valido	1314
La distribuzione non viene completata.	1314
Errore: impossibile trovare gli eseguibili java o java8 oppure l'errore: Implementazione <deployment-id>di tipo NewDeployment per gruppo <group-id>non riuscita errore: worker with <worker-id>failed to initialize with reason La versione Java installata deve essere maggiore o uguale a 8	1315
La distribuzione non viene completata e il file runtime.log contiene più voci "wait 1s for container to stop".	1316
La distribuzione non viene completata e runtime.log mostra il seguente messaggio di errore: "[ERROR]-Greengrass deployment error: failed to report deployment status back to cloud {"deploymentId": "<deployment-id>", "errorString": "Failed to initiate PUT, endpoint: https://<deployment-status>, error: Put https://<deployment-status>: proxyconnect tcp: x509: certificate signed by unknown authority"}"	1316
<path>Errore: la distribuzione <deployment-id>del tipo NewDeployment per il gruppo <group-id>non è riuscita. Errore durante l'elaborazione. group config non è valido: 112 o [119 0] non dispongono dell'autorizzazione rw sul file:	1317
Errore: < list-of-function-arns > sono configurati per l'esecuzione come root ma Greengrass non è configurato per eseguire funzioni Lambda con autorizzazioni root.	1317
Errore: distribuzione <deployment-id>di tipo NewDeployment per gruppo <group-id>non riuscita Errore di distribuzione di Greengrass: impossibile eseguire la fase di download nella distribuzione. errore durante l'elaborazione: impossibile caricare il file di gruppo scaricato: impossibile trovare l'UID in base al nome utente, UserName: ggc_user: user: unknown user ggc_user.	1318
Errore: errore di esecuzione [ERROR]-runtime: impossibile avviare il container lambda. {"errorString": "failed to initialize container mounts: failed to mask greengrass root in overlay upper dir: failed to create mask device at directory <ggc-path>: file exists"}	1318
Errore: distribuzione <deployment-id>di tipo NewDeployment per gruppo <group-id>non riuscita errore: avvio del processo non riuscito: container_linux.go:259: l'avvio del processo contenitore ha causato «process_linux.go:250: l'esecuzione del processo exec setns per init ha causato\" wait: nessun processo figlio\" ».	1319

Errore: [<host-prefix>WARN] -MQTT [client] dial tcp: lookup -ats.iot. <region>.amazonaws.com: nessun host di questo tipo... [ERROR] -Errore di implementazione di Greengrass: impossibile riportare lo stato della distribuzione al cloud... net/http: richiesta annullata in attesa della connessione (Client.Timeout superato in attesa delle intestazioni)	1319
Problemi relativi alla creazione del gruppo e della funzione	1320
Errore: la configurazione 'IsolationMode' per il gruppo non è valida.	1320
Errore: la configurazione 'IsolationMode' per la funzione con arn non <function-arn>è valida.	1321
Errore: la MemorySize configurazione per la funzione con arn <function-arn>non è consentita in =. IsolationMode NoContainer	1321
Errore: la configurazione di Access Sysfs per la funzione con arn <function-arn>non è consentita in =. IsolationMode NoContainer	1321
Errore: la MemorySize configurazione per la funzione con arn <function-arn>è richiesta in IsolationMode =. GreengrassContainer	1321
Errore: la funzione <function-arn>si riferisce a una risorsa <resource-type>di tipo non consentito in IsolationMode =NoContainer.	1322
Errore: la configurazione dell'esecuzione per la funzione con arn <function-arn> non è consentita.	1322
Problemi di individuazione	1322
Errore: il dispositivo è membro di troppi gruppi, è possibile che i dispositivi non siano presenti in più di 10 gruppi	1323
Problemi relativi alle risorse di Machine Learning	1323
MLModelProprietario non valido: GroupOwnerSetting viene fornito nella risorsa del modello ML, ma non è presente GroupOwner o non GroupPermission è presente	427
NoContainer la funzione non può configurare l'autorizzazione quando si collegano risorse di Machine Learning. <function-arn>si riferisce alla risorsa Machine Learning <resource-id>con autorizzazione <ro/rw> nella politica di accesso alle risorse.	427
La funzione <function-arn>si riferisce alla risorsa di Machine Learning <resource-id>con autorizzazione mancante in entrambe ResourceAccessPolicy le risorse OwnerSetting.	428
La funzione <function-arn>si riferisce alla risorsa Machine Learning <resource-id>con autorizzazione\ "rw\», mentre l'impostazione del proprietario della risorsa consente GroupPermission solo\ "ro\».	428
NoContainer La funzione <function-arn>si riferisce alle risorse del percorso di destinazione annidato.	428

Lambda <function-arn> ottiene l'accesso alla risorsa <resource-id> condividendo lo stesso ID del proprietario del gruppo	429
AWS IoT Greengrass problemi principali di Docker	1325
Errore: opzioni sconosciute: -. no-include-email	385
Attenzione: IPv4 è disattivata. Networking non funzionerà.	385
Errore: un firewall sta bloccando la condivisione di file tra finestre e contenitori.	385
Errore: si è verificato un errore (AccessDeniedException) durante la chiamata dell' GetAuthorizationToken operazione: User: arn:aws:iam: :user/ <account-id><user-name>non è autorizzato a eseguire: ecr: on resource: * GetAuthorizationToken	385
Errore: impossibile creare container per il servizio greengrass: Conflict. Il nome del contenitore «/aws-iot-greengrass" è già in uso.	1327
Errore: [FATAL]-Non è riuscito a reimpostare lo spazio dei nomi del montaggio del thread a causa di un errore inaspettato: "operation not permitted". Per mantenere la coerenza, GGC si arresterà e dovrà essere riavviato manualmente.	1328
Risoluzione dei problemi con i log	1328
Risoluzione dei problemi di storage	1330
Risoluzione dei problemi relativi ai messaggi	1330
Risoluzione dei problemi di timeout della sincronizzazione shadow	1331
Seleziona Re:post AWS	1332
Cronologia dei documenti	1333
Aggiornamenti precedenti	1357

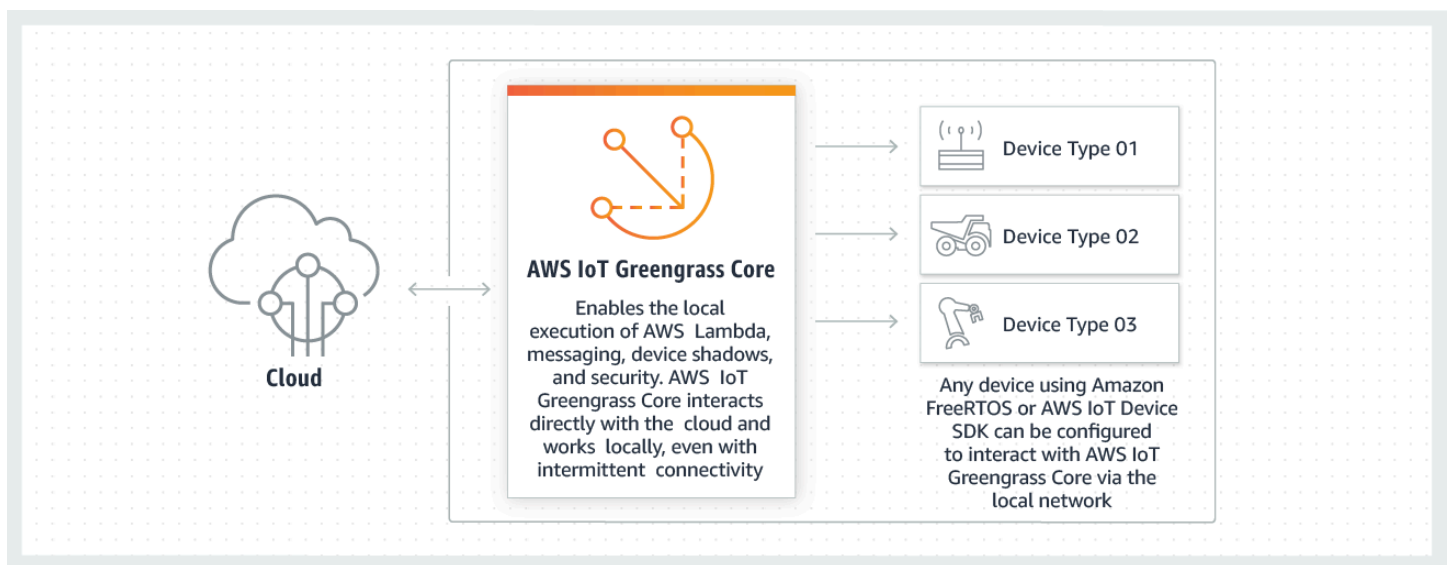
AWS IoT Greengrass Version 1 è entrato nella fase di estensione della vita utile il 30 giugno 2023. Per ulteriori informazioni, consulta la [politica AWS IoT Greengrass V1 di manutenzione](#). Dopo questa data, AWS IoT Greengrass V1 non rilascerà aggiornamenti che forniscano funzionalità, miglioramenti, correzioni di bug o patch di sicurezza. I dispositivi che funzionano AWS IoT Greengrass V1 non subiranno interruzioni e continueranno a funzionare e a connettersi al cloud. Ti consigliamo vivamente di eseguire la [migrazione a AWS IoT Greengrass Version 2](#), che aggiunge [nuove importanti funzionalità](#) e [supporto per piattaforme aggiuntive](#).

Le traduzioni sono generate tramite traduzione automatica. In caso di conflitto tra il contenuto di una traduzione e la versione originale in Inglese, quest'ultima prevarrà.

Che cos'è AWS IoT Greengrass?

AWS IoT Greengrass è un software che estende le funzionalità cloud ai dispositivi locali. Consente ai dispositivi di raccogliere e analizzare i dati più vicini all'origine delle informazioni, reagire autonomamente a eventi locali e comunicare in modo sicuro tra di loro sulle reti locali. I dispositivi locali possono anche comunicare in modo sicuro AWS IoT Core ed esportare i dati IoT in Cloud. AWS IoT Greengrass gli sviluppatori possono utilizzare AWS Lambda funzioni e [connettori](#) predefiniti per creare applicazioni serverless che vengono distribuite sui dispositivi per l'esecuzione locale.

Il diagramma seguente mostra l'architettura di base di AWS IoT Greengrass



AWS IoT Greengrass consente ai clienti di creare dispositivi IoT e logica applicativa. In particolare, AWS IoT Greengrass fornisce la gestione basata sul cloud della logica applicativa che viene eseguita sui dispositivi. Le funzioni e i connettori Lambda distribuiti localmente vengono attivati da eventi locali, messaggi dal cloud o altre fonti.

In AWS IoT Greengrass, i dispositivi comunicano in modo sicuro su una rete locale e scambiano messaggi tra loro senza doversi connettere al cloud. AWS IoT Greengrass fornisce un gestore di messaggi pub/sub locale in grado di bufferizzare in modo intelligente i messaggi in caso di interruzione della connettività, in modo da preservare i messaggi in entrata e in uscita sul cloud.

AWS IoT Greengrass protegge i dati degli utenti:

- Attraverso l'autenticazione e l'autorizzazione sicura dei dispositivi.
- Attraverso la connettività sicura nella rete locale.

- Tra i dispositivi locali e il cloud.

Le credenziali di sicurezza dei dispositivi funzionano in gruppo fino alla revoca, anche in caso di interruzione della connettività al cloud, in modo che i dispositivi possano continuare a comunicare in modo sicuro a livello locale.

AWS IoT Greengrass fornisce over-the-air aggiornamenti sicuri delle funzioni Lambda.

AWS IoT Greengrass è composto da:

- Distribuzioni software
 - AWS IoT Greengrass Software di base
 - AWS IoT Greengrass SDK principale
- Servizio cloud
 - AWS IoT Greengrass API
- Funzionalità
 - Runtime Lambda
 - Implementazione shadow
 - Gestore messaggistica
 - Gestione di gruppo
 - Servizio di individuazione
 - Over-the-air agente di aggiornamento
 - Stream manager
 - Accesso alle risorse locali
 - Inferenza del Machine Learning locale
 - Secrets Manager locale
 - Connettori con integrazione predefinita con servizi, protocolli e software

Argomenti

- [AWS IoT Greengrass Software di base](#)
- [AWS IoT Greengrass gruppi](#)
- [Dispositivi in AWS IoT Greengrass](#)
- [SDKs](#)

- [Piattaforme supportate e requisiti](#)
- [AWS IoT Greengrass download](#)
- [Ci piacerebbe conoscere la tua opinione](#)
- [Installare il software AWS IoT Greengrass Core](#)
- [Configura il AWS IoT Greengrass core](#)

AWS IoT Greengrass Software di base

Il software AWS IoT Greengrass Core offre le seguenti funzionalità:

- Implementazione ed esecuzione locale di connettori e funzioni Lambda.
- Elabora i flussi di dati localmente con esportazioni automatiche verso Cloud AWS
- Messaggistica MQTT sulla rete locale tra dispositivi, connettori e funzioni Lambda utilizzando abbonamenti gestiti.
- Messaggistica MQTT tra AWS IoT dispositivi, connettori e funzioni Lambda utilizzando abbonamenti gestiti.
- Connessioni sicure tra dispositivi e Cloud AWS utilizzo dell'autenticazione e dell'autorizzazione dei dispositivi.
- Sincronizzazione shadow locale dei dispositivi. Le ombre possono essere configurate per la sincronizzazione con Cloud AWS
- Accesso controllato alle risorse volume e dispositivo locali.
- Distribuzione di modelli di machine learning formati nel cloud per l'esecuzione di un'inferenza locale.
- Rilevamento automatico dell'indirizzo IP che permette ai dispositivi di scoprire il dispositivo Greengrass core.
- Distribuzione centralizzata di una configurazione del gruppo, nuova o aggiornata. Dopo il download dei dati di configurazione, il dispositivo core viene riavviato automaticamente.
- Aggiornamenti software sicuri over-the-air (OTA) delle funzioni Lambda definite dall'utente.
- Archiviazione sicura e crittografata dei segreti locali e accesso controllato tramite connettori e funzioni Lambda.

AWS IoT Greengrass le istanze principali vengono configurate in AWS IoT Greengrass APIs modo da creare e aggiornare le definizioni di AWS IoT Greengrass gruppo archiviate nel cloud.

AWS IoT Greengrass Versioni software principali

AWS IoT Greengrass fornisce diverse opzioni per l'installazione del software AWS IoT Greengrass Core, inclusi i file di download tar.gz, uno script di avvio rapido e apt installazioni su piattaforme Debian supportate. Per ulteriori informazioni, consulta [the section called “Installare il software AWS IoT Greengrass Core”](#).

Le seguenti schede descrivono le novità e le modifiche apportate alle versioni del software AWS IoT Greengrass Core.

GGC v1.11

1.11.6

Correzione di bug e miglioramenti:

- Maggiore resilienza in caso di improvvisa perdita di alimentazione durante un'implementazione.
- È stato risolto un problema a causa del quale il danneggiamento dei dati dello stream manager poteva impedire l'avvio del software AWS IoT Greengrass Core.
- È stato risolto un problema per cui i nuovi dispositivi client non potevano connettersi al core in determinati scenari.
- È stato risolto un problema a causa del quale i nomi degli stream manager non potevano contenere .log.

1.11.5

Correzione di bug e miglioramenti:

- Miglioramenti delle prestazioni generali e correzioni di bug.

1.11.4

Correzione di bug e miglioramenti:

- È stato risolto un problema con lo stream manager che impediva gli aggiornamenti al AWS IoT Greengrass software Core v1.11.3. Se utilizzi stream manager per esportare dati nel cloud, ora puoi utilizzare un aggiornamento OTA per aggiornare una versione precedente v1.x del software Core alla v1.11.4. AWS IoT Greengrass
- Miglioramenti delle prestazioni generali e correzioni di bug.

1.11.3

Correzione di bug e miglioramenti:

- È stato risolto un problema che impediva al software AWS IoT Greengrass Core, eseguito in un attimo, di rispondere in un attimo su un dispositivo Ubuntu dopo un'improvvisa interruzione dell'alimentazione del dispositivo.
- È stato risolto un problema che causava un ritardo nella consegna dei messaggi MQTT a funzioni Lambda di lunga durata.
- È stato risolto un problema che impediva l'invio corretto dei messaggi MQTT quando il `maxWorkItemCount` valore era impostato su un valore maggiore di. 1024
- È stato risolto un problema che faceva sì che l'agente di aggiornamento OTA ignorasse il `KeepAlive` periodo MQTT specificato nella `keepAlive` proprietà in. [config.json](#)
- Miglioramenti delle prestazioni generali e correzioni di bug.

Important

Se utilizzi stream manager per esportare dati nel cloud, non eseguire l'aggiornamento al software AWS IoT Greengrass Core v1.11.3 da una versione precedente v1.x. Se stai abilitando lo stream manager per la prima volta, ti consigliamo vivamente di installare prima la versione più recente del software Core. AWS IoT Greengrass

1.11.1

Correzione di bug e miglioramenti:

- È stato risolto un problema che causava un maggiore utilizzo della memoria per lo stream manager.
- È stato risolto un problema che causava il ripristino del numero di sequenza dello stream 0 se il dispositivo principale Greengrass era spento per un periodo più lungo del periodo specificato time-to-live (TTL) dei dati del flusso.
- È stato risolto un problema che impediva allo stream manager di interrompere correttamente i tentativi di esportare i dati in. Cloud AWS

1.11.0

Nuove caratteristiche:

- Un agente di telemetria sul core di Greengrass raccoglie dati di telemetria locali e li pubblica su. Cloud AWS Per recuperare i dati di telemetria per un'ulteriore elaborazione, i clienti possono creare una EventBridge regola Amazon e iscriversi a un target. Per ulteriori

informazioni, consulta [Raccolta di dati di telemetria sanitaria del sistema](#) dai dispositivi principali. AWS IoT Greengrass

- Un'API HTTP locale restituisce un'istantanea dello stato corrente dei processi di lavoro locali avviati da. AWS IoT Greengrass Per ulteriori informazioni, consulta [Chiamare l'API per il controllo sanitario locale](#).
- Un [gestore di flussi](#) esporta automaticamente i dati in Amazon S3 e. AWS IoT SiteWise

I nuovi [parametri dello stream manager](#) consentono di aggiornare gli stream esistenti e sospendere o riprendere l'esportazione dei dati.

- Support per l'esecuzione di funzioni Lambda di Python 3.8.x sul core.
- Una nuova `ggDaemonPort` proprietà da utilizzare per configurare il numero di porta IPC principale di Greengrass. [config.json](#) Il numero di porta predefinito è 8000.

Una nuova `systemComponentAuthTimeout` proprietà [config.json](#) che si utilizza per configurare il timeout per l'autenticazione IPC di base di Greengrass. Il timeout predefinito è di 5000 millisecondi.

- Il numero massimo di AWS IoT dispositivi per AWS IoT Greengrass gruppo è stato aumentato da 200 a 2500.

È stato aumentato il numero massimo di abbonamenti per gruppo da 1000 a 10000.

Per ulteriori informazioni, consulta [Endpoint e quote per AWS IoT Greengrass](#).

Correzione di bug e miglioramenti:

- Ottimizzazione generale che può ridurre l'utilizzo della memoria dei processi di servizio Greengrass.
- Un nuovo parametro di configurazione di runtime (`mountAllBlockDevices`) consente a Greengrass di utilizzare i bind mount per montare tutti i dispositivi a blocchi in un contenitore dopo aver configurato OverlayFS. Questa funzionalità ha risolto un problema che causava il fallimento della distribuzione di Greengrass se `/usr` non rientrava nella `/` gerarchia.
- È stato risolto un problema che causava un errore di AWS IoT Greengrass base se `/tmp` si trattava di un collegamento simbolico.
- È stato risolto un problema che consentiva all'agente di distribuzione Greengrass di rimuovere gli artefatti del modello di machine learning inutilizzati dalla cartella. `mlmodel_public`
- Miglioramenti delle prestazioni generali e correzioni di bug.

Extended life versions

1.10.5

Correzione di bug e miglioramenti:

- Miglioramenti delle prestazioni generali e correzioni di bug.

1.10.4

Correzione di bug e miglioramenti:

- È stato risolto un problema che impediva al software AWS IoT Greengrass Core, eseguito in un attimo, di rispondere in un attimo su un dispositivo Ubuntu dopo un'improvvisa interruzione dell'alimentazione del dispositivo.
- È stato risolto un problema che causava un ritardo nella consegna dei messaggi MQTT a funzioni Lambda di lunga durata.
- È stato risolto un problema che impediva l'invio corretto dei messaggi MQTT quando il `maxWorkItemCount` valore era impostato su un valore maggiore di. 1024
- È stato risolto un problema che faceva sì che l'agente di aggiornamento OTA ignorasse il KeepAlive periodo MQTT specificato nella keepAlive proprietà in. [config.json](#)
- Miglioramenti delle prestazioni generali e correzioni di bug.

1.10.3

Correzione di bug e miglioramenti:

- Una nuova `systemComponentAuthTimeout` proprietà [config.json](#) che si utilizza per configurare il timeout per l'autenticazione IPC di base di Greengrass. Il timeout predefinito è di 5000 millisecondi.
- È stato risolto un problema che causava un maggiore utilizzo della memoria per lo stream manager.

1.10.2

Correzione di bug e miglioramenti:

- Una nuova `mqttOperationTimeout` proprietà in [config.json](#) che potete utilizzare per impostare il timeout per le operazioni di pubblicazione, sottoscrizione e annullamento dell'iscrizione nelle connessioni MQTT con. AWS IoT Core
- Miglioramenti delle prestazioni generali e correzioni di bug.

1.10.1

Correzione di bug e miglioramenti:

- [Stream manager](#) è più resiliente al danneggiamento dei dati dei file.
- Risolto un problema che causava un errore di montaggio di sysfs sui dispositivi che utilizzavano Linux kernel 5.1 e versioni successive.
- Miglioramenti delle prestazioni generali e correzioni di bug.

1.10.0

Nuove caratteristiche:

- Un gestore di flussi che elabora i flussi di dati localmente e li esporta automaticamente. Cloud AWS Questa funzionalità richiede Java 8 sul dispositivo core Greengrass. Per ulteriori informazioni, consulta [Gestione dei flussi di dati](#).
- Un nuovo connettore di distribuzione dell'applicazione Docker Greengrass che esegue un'applicazione Docker su un dispositivo core. Per ulteriori informazioni, consulta [the section called "Distribuzione dell'applicazione Docker"](#).
- Un nuovo SiteWise connettore IoT che invia i dati dei dispositivi industriali dai server OPC-UA alle proprietà degli asset. AWS IoT SiteWise Per ulteriori informazioni, consulta [the section called "IoT SiteWise"](#).
- Le funzioni Lambda eseguite senza containerizzazione possono accedere alle risorse di machine learning del gruppo Greengrass. Per ulteriori informazioni, consulta [the section called "Accesso alle risorse di Machine Learning"](#).
- Support per sessioni persistenti MQTT con AWS IoT. Per ulteriori informazioni, consulta [the section called "Sessioni persistenti MQTT con AWS IoT Core"](#).
- Il traffico MQTT locale può viaggiare su una porta diversa dalla porta predefinita 8883. Per ulteriori informazioni, consulta [the section called "Porta MQTT per la messaggistica locale"](#).
- Nuove queueFullPolicy opzioni nel [AWS IoT Greengrass Core SDK](#) per una pubblicazione affidabile dei messaggi dalle funzioni Lambda.
- Support per l'esecuzione delle funzioni Lambda di Node.js 12.x sul core.
- Over-the-air Gli aggiornamenti (OTA) con integrazione della sicurezza hardware possono essere configurati con OpenSSL 1.1.
- Miglioramenti delle prestazioni generali e correzioni di bug.

1.9.4

Correzione di bug e miglioramenti:

- Miglioramenti delle prestazioni generali e correzioni di bug.

1.9.3

Nuove caratteristiche:

- Support per ARMv6L. AWS IoT Greengrass Il software di base v1.9.3 o successivo può essere installato su distribuzioni Raspbian su architetture ARMv6L (ad esempio, su dispositivi Raspberry Pi Zero).
- Aggiornamenti OTA sulla porta 443 con ALPN. I core Greengrass che utilizzano la porta 443 per il traffico MQTT ora supportano gli aggiornamenti software over-the-air (OTA). AWS IoT Greengrass utilizza l'estensione TLS Application Layer Protocol Network (ALPN) per abilitare queste connessioni. Per ulteriori informazioni, consulta [Aggiornamenti OTA del software AWS IoT Greengrass Core](#) e [the section called “Connessione alla porta 443 o tramite un proxy di rete”](#).

Correzione di bug e miglioramenti:

- Risolve un bug introdotto nella versione 1.9.0 che impediva alle funzioni Lambda di Python 2.7 di inviare payload binari ad altre funzioni Lambda.
- Miglioramenti delle prestazioni generali e correzioni di bug.

1.9.2

Nuove caratteristiche:

- Support per [OpenWrt](#). AWS IoT Greengrass Il software di base v1.9.2 o successivo può essere installato su OpenWrt distribuzioni con architetture Armv8 () e ARMv7L. AArch64 OpenWrt Attualmente, non supporta l'inferenza ML.

1.9.1

Correzione di bug e miglioramenti:

- Correzione di un bug introdotto nella v1.9.0 che invia messaggi da c loud contenenti i caratteri jolly nell'argomento.

1.9.0

Nuove caratteristiche:

- Support per i runtime Lambda di Python 3.7 e Node.js 8.10. Le funzioni Lambda che utilizzano i runtime Python 3.7 e Node.js 8.10 ora possono essere eseguite su un core. AWS IoT Greengrass (AWS IoT Greengrass continua a supportare i runtime Python 2.7 e Node.js 6.10.)
- Connessioni MQTT ottimizzate. Il core Greengrass stabilisce meno connessioni con AWS IoT Core. Questa modifica è in grado di ridurre i costi operativi per le spese basate sul numero di connessioni.
- Chiave Elliptic Curve (EC) per il server MQTT locale. Il server locale MQTT supporta le chiavi EC in aggiunta alle chiavi RSA. Il certificato del server MQTT ha una firma RSA SHA-256, indipendentemente dal tipo di chiave. Per ulteriori informazioni, consulta [the section called “Principal di sicurezza”](#).

Correzione di bug e miglioramenti:

- Miglioramenti delle prestazioni generali e correzioni di bug.

1.8.4

È stato risolto un problema relativo alla sincronizzazione shadow e alla riconnessione di Device Certificate Manager (DCM).

Miglioramenti delle prestazioni generali e correzioni di bug.

1.8.3

Miglioramenti delle prestazioni generali e correzioni di bug.

1.8.2

Miglioramenti delle prestazioni generali e correzioni di bug.

1.8.1

Miglioramenti delle prestazioni generali e correzioni di bug.

1.8.0

Nuove caratteristiche:

- Identità di accesso predefinita configurabile per le funzioni Lambda nel gruppo. Questa impostazione a livello di gruppo determina le autorizzazioni predefinite utilizzate per eseguire le funzioni Lambda. Puoi impostare l'ID utente, l'ID gruppo o entrambi. Le singole funzioni Lambda possono sovrascrivere l'identità di accesso predefinita del relativo gruppo.

Per ulteriori informazioni, consulta [the section called “Impostazione dell'identità di accesso predefinita per le funzioni Lambda in un gruppo”](#).

- Traffico HTTPS sulla porta 443. La comunicazione HTTPS può essere configurata per viaggiare sulla porta 443 anziché sulla porta predefinita 8443. Ciò integra il AWS IoT Greengrass supporto per l'estensione TLS Application Layer Protocol Network (ALPN) e consente a tutto il traffico di messaggistica Greengrass, sia MQTT che HTTPS, di utilizzare la porta 443. Per ulteriori informazioni, consulta [the section called “Connessione alla porta 443 o tramite un proxy di rete”](#).
- Client IDs per AWS IoT le connessioni con nome prevedibile. Questa modifica abilita il supporto per AWS IoT Device Defender e gli [eventi del ciclo di vita AWS IoT](#), in modo da poter ricevere notifiche per eventi di connessione, disconnessione, sottoscrizione e annullamento sottoscrizione. La denominazione prevedibile semplifica inoltre la creazione di una logica relativa alla connessione IDs (ad esempio, la creazione di modelli di [policy di sottoscrizione](#) basati sugli attributi dei certificati). Per ulteriori informazioni, consulta [the section called “Client IDs per connessioni MQTT con AWS IoT”](#).

Correzione di bug e miglioramenti:

- È stato risolto un problema relativo alla sincronizzazione shadow e alla riconnessione di Device Certificate Manager (DCM).
- Miglioramenti delle prestazioni generali e correzioni di bug.

1.7.1

Nuove caratteristiche:

- I connettori Greengrass forniscono un'integrazione integrata con l'infrastruttura locale, i protocolli dei dispositivi e altri servizi cloud. AWS Per ulteriori informazioni, consulta [Integrazione con servizi e protocolli tramite i connettori](#).
- AWS IoT Greengrass si estende AWS Secrets Manager ai dispositivi principali, il che rende disponibili password, token e altri segreti per i connettori e le funzioni Lambda. I segreti vengono crittografati mentre sono in transito e quando sono inattivi. Per ulteriori informazioni, consulta [Distribuzione dei segreti nel core](#).
- Supporto delle soluzioni hardware di sicurezza affidabili. Per ulteriori informazioni, consulta [the section called “Integrazione della sicurezza hardware”](#).
- Impostazioni di isolamento e autorizzazione che consentono alle funzioni Lambda di funzionare senza contenitori Greengrass e di utilizzare le autorizzazioni di un utente e un gruppo specificati. Per ulteriori informazioni, consulta [the section called “Controllo dell'esecuzione della funzione Greengrass Lambda”](#).

- Puoi eseguirlo AWS IoT Greengrass in un contenitore Docker (su Windows, macOS o Linux) configurando il tuo gruppo Greengrass per l'esecuzione senza containerizzazione. Per ulteriori informazioni, consulta [the section called “Esegui AWS IoT Greengrass in un contenitore Docker”](#).
- Messaggistica MQTT sulla porta 443 con Application Layer Protocol Negotiation (ALPN) o la connessione tramite un proxy di rete. Per ulteriori informazioni, consulta [the section called “Connessione alla porta 443 o tramite un proxy di rete”](#).
- Il runtime di deep learning SageMaker AI Neo, che supporta modelli di machine learning ottimizzati dal compilatore di deep learning AI Neo. SageMaker Per ulteriori maggiori informazioni su Neo Deep Learning Runtime, consulta [the section called “Runtime e librerie per inferenza ML”](#).
- Supporto di Raspbian Stretch (2018-06-27) su dispositivi core Raspberry Pi.

Correzione di bug e miglioramenti:

- Miglioramenti delle prestazioni generali e correzioni di bug.

In aggiunta, le seguenti caratteristiche sono disponibili con questo rilascio:

- Il AWS IoT Device Tester for AWS IoT Greengrass, che puoi utilizzare per verificare il funzionamento dell'architettura della CPU, della configurazione del kernel e dei driver. AWS IoT Greengrass Per ulteriori informazioni, consulta [Utilizzo di AWS IoT Device Tester per AWS IoT Greengrass V1](#).
- I pacchetti AWS IoT Greengrass Core software, AWS IoT Greengrass Core SDK e AWS IoT Greengrass Machine Learning SDK possono essere scaricati tramite Amazon. CloudFront Per ulteriori informazioni, consulta [the section called “AWS IoT Greengrass download”](#).

1.6.1

Nuove caratteristiche:

- Eseguibili Lambda che eseguono codice binario sul core Greengrass. Usa il nuovo AWS IoT Greengrass Core SDK per C per scrivere eseguibili Lambda in C e C++. Per ulteriori informazioni, consulta [the section called “Eseguibili Lambda”](#).
- Cache opzionale per i messaggi di storage locale con caratteristiche di persistenza anche in caso di riavvio. Puoi configurare le impostazioni di storage per i messaggi MQTT in coda per l'elaborazione. Per ulteriori informazioni, consulta [the section called “Coda di messaggi MQTT”](#).
- Intervallo massimo di riconnessione configurabile nel caso in cui il dispositivo core sia scollegato. Per ulteriori informazioni, consulta la proprietà

`mqttMaxConnectionRetryInterval` in [the section called “AWS IoT Greengrass file di configurazione principale”](#).

- Accesso della risorsa alla directory /proc dell'host. Per ulteriori informazioni, consulta [Accesso alle risorse locali](#).
- Directory di scrittura configurabile. Il software AWS IoT Greengrass Core può essere distribuito in posizioni di sola lettura e lettura-scrittura. Per ulteriori informazioni, consulta [the section called “Configurazione di una directory di scrittura”](#).

Correzione di bug e miglioramenti:

- Miglioramento delle performance relative alla pubblicazione di messaggi all'interno di Greengrass Core e tra dispositivi e il dispositivo principale.
- Riduzione delle risorse di calcolo necessarie per elaborare i log generati dalle funzioni Lambda definite dall'utente.

1.5.0

Nuove caratteristiche:

- AWS IoT Greengrass L'inferenza del Machine Learning (ML) è generalmente disponibile. È possibile eseguire l'inferenza di Machine Learning localmente sui dispositivi AWS IoT Greengrass usando modelli che vengono compilati e addestrati nel cloud. Per ulteriori informazioni, consulta [Esecuzione dell'inferenza di Machine Learning](#).
- Le funzioni Greengrass Lambda ora supportano i dati binari come payload di input, oltre a JSON. [Per utilizzare questa funzionalità, è necessario eseguire l'aggiornamento alla versione 1.1.0 di AWS IoT Greengrass Core SDK, che è possibile scaricare dalla pagina dei download di Core SDK.AWS IoT Greengrass](#)

Correzione di bug e miglioramenti:

- Riduzione complessiva dell'impatto sulla memoria.
- Miglioramenti delle prestazioni per l'invio di messaggi al cloud.
- Miglioramenti delle prestazioni e della stabilità per l'agente di download, Device Certificate Manager e l'agente di aggiornamento OTA.
- Correzioni di bug minori.

1.3.0

Nuove caratteristiche:

- Over-the-air Agente di aggiornamento (OTA) in grado di gestire i processi di aggiornamento Greengrass distribuiti sul cloud. L'agente è disponibile nella nuova directory `/greengrass/ota`. Per ulteriori informazioni, consulta [Aggiornamenti OTA del software AWS IoT Greengrass Core](#).
- La caratteristica di accesso alle risorse locali permette alle funzioni Lambda Greengrass di accedere alle risorse locali come volumi e dispositivi periferici. Per ulteriori informazioni, consulta [Accedi alle risorse locali con funzioni e connettori Lambda](#).

1.1.0

Nuove caratteristiche:

- AWS IoT Greengrass I gruppi distribuiti possono essere ripristinati eliminando funzioni, sottoscrizioni e configurazioni Lambda. Per ulteriori informazioni, consulta [the section called "Reimpostazione delle distribuzioni"](#).
- Support per i runtime Node.js 6.10 e Java 8 Lambda, oltre a Python 2.7.

Per migrare dalla versione precedente del core: AWS IoT Greengrass

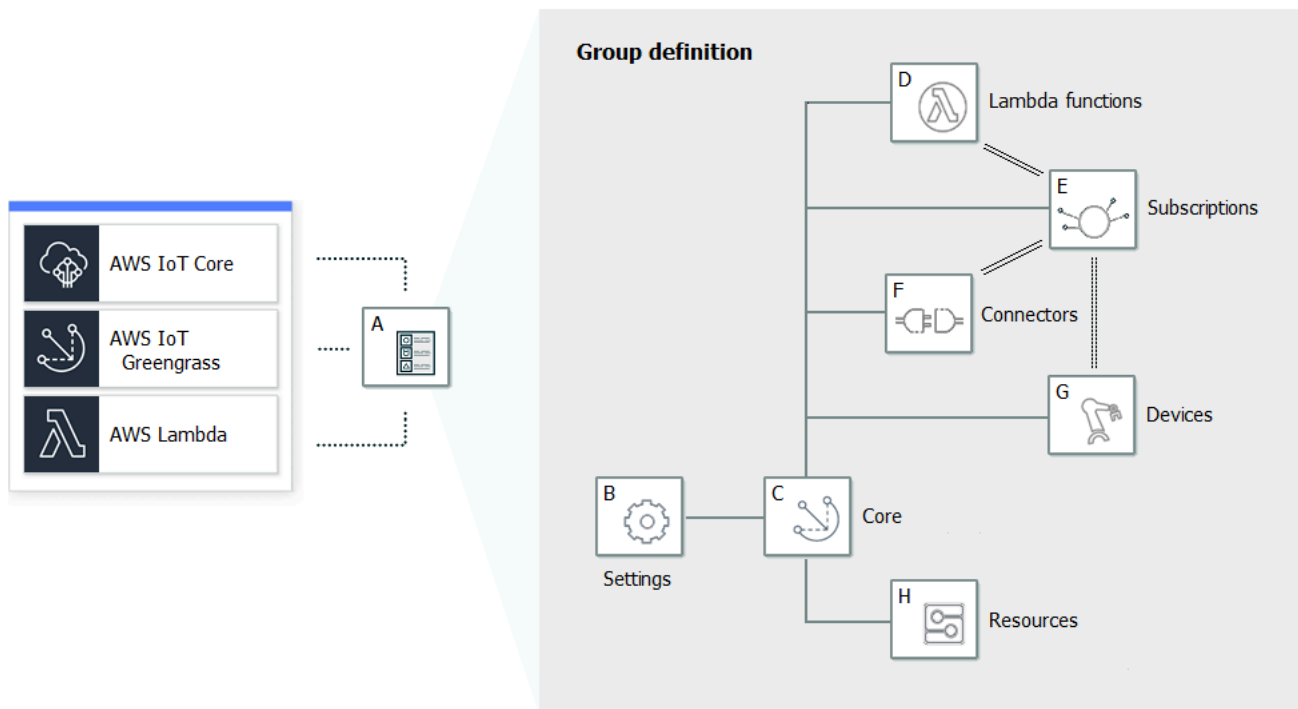
- Copia i certificati dalla cartella `/greengrass/configuration/certs` a `/greengrass/certs`.
- Copia `/greengrass/configuration/config.json` su `/greengrass/config/config.json`.
- Esegui `/greengrass/ggc/core/greengrassd` invece di `/greengrass/greengrassd`.
- Distribuisci il gruppo al nuovo core.

1.0.0

Versione iniziale

AWS IoT Greengrass gruppi

Un gruppo Greengrass è un insieme di impostazioni e componenti, ad esempio un nucleo Greengrass, dispositivi e sottoscrizioni. I gruppi vengono utilizzati per definire una portata di interazione. Ad esempio, un gruppo può rappresentare un piano di un edificio, un camion o un intero sito minerario. Il seguente diagramma mostra i componenti che possono far parte di un gruppo Greengrass.



Nello schema precedente:

A: Definizione del gruppo Greengrass

Informazioni sulle impostazioni di gruppo e sui componenti.

B: Impostazioni del gruppo Greengrass

Ciò include:

- Ruolo del gruppo Greengrass
- Configurazione dell'autorità di certificazione e della connessione locale.
- Informazioni sulla connettività di base di Greengrass.
- Ambiente di runtime Lambda predefinito. Per ulteriori informazioni, consulta [the section called “Impostazione della containerizzazione predefinita per le funzioni Lambda in un gruppo”](#).
- CloudWatch e configurazione dei log locali. Per ulteriori informazioni, consulta [the section called “Monitoraggio con i log AWS IoT Greengrass”](#).

C: Greengrass core

La AWS IoT cosa (dispositivo) che rappresenta il nucleo di Greengrass. Per ulteriori informazioni, consulta [the section called “Configura il AWS IoT Greengrass core”](#).

D: Definizione della funzione Lambda

Un elenco di funzioni Lambda eseguite localmente sul core, con i dati di configurazione associati. Per ulteriori informazioni, consulta [Esegui funzioni Lambda locali](#).

E: definizione di sottoscrizione

Un elenco di sottoscrizioni che consentono le comunicazioni tramite messaggi MQTT. Una sottoscrizione definisce:

- Un'origine e una destinazione del messaggio. Questi possono essere dispositivi client, funzioni Lambda AWS IoT Core, connettori e il servizio shadow locale.
- Un argomento (o oggetto) utilizzato per filtrare i messaggi.

Per ulteriori informazioni, consulta [the section called “Sottoscrizioni gestite nel flusso di lavoro di messaggistica MQTT”](#).

F: definizione di un connettore

Un elenco di connettori eseguiti in locale nel core, con i dati di configurazione associati. Per ulteriori informazioni, consulta [Integrazione con servizi e protocolli tramite i connettori](#).

G: definizione di dispositivo

Un elenco di AWS IoT elementi (noti come dispositivi o dispositivi client) che fanno parte del gruppo Greengrass, con i dati di configurazione associati. Per ulteriori informazioni, consulta [the section called “Dispositivi in AWS IoT Greengrass”](#).

H: definizione di risorsa

Un elenco delle risorse locali, delle risorse di Machine Learning e delle risorse segrete su Greengrass core, con i relativi dati di configurazione. Per ulteriori informazioni, consulta [Accesso alle risorse locali](#), [Esecuzione dell'inferenza di Machine Learning](#) e [Distribuzione dei segreti nel core](#).

Una volta implementate, la definizione del gruppo Greengrass, le funzioni Lambda, i connettori, le risorse e la tabella di sottoscrizione vengono copiate nel dispositivo principale. Per ulteriori informazioni, consulta [Distribuisci gruppi AWS IoT Greengrass](#).

Dispositivi in AWS IoT Greengrass

Un gruppo Greengrass può contenere due tipi di AWS IoT dispositivi:

Core Greengrass

Un core Greengrass è un dispositivo che esegue il software AWS IoT Greengrass Core, che gli consente di comunicare direttamente con AWS IoT Core e con il AWS IoT Greengrass servizio. Un core dispone del proprio certificato del dispositivo utilizzato per l'autenticazione con AWS IoT Core. Dispone di un'ombra del dispositivo e di una voce nel AWS IoT Core registro. I core Greengrass eseguono un runtime Lambda locale, un agente di distribuzione e un tracker di indirizzi IP che invia le informazioni sull'indirizzo IP al AWS IoT Greengrass servizio per consentire ai dispositivi client di scoprire automaticamente il gruppo e le informazioni di connessione principali. Per ulteriori informazioni, consulta [the section called “Configura il AWS IoT Greengrass core”](#).

Note

Un gruppo Greengrass deve contenere un solo core.

Dispositivo client

I dispositivi client (chiamati anche dispositivi connessi, dispositivi Greengrass o dispositivi) sono dispositivi che si connettono a un core Greengrass tramite MQTT. Dispongono del proprio certificato di AWS IoT Core autenticazione, di un dispositivo shadow e di una voce nel registro. AWS IoT Core I dispositivi client possono eseguire [FreerTOS](#) o utilizzare [Device SDK AWS IoT Greengrass o Discovery API per ottenere informazioni di rilevamento utilizzate per connettersi e autenticarsi con il core AWS IoT dello](#) stesso gruppo Greengrass. Per informazioni su come utilizzare la AWS IoT console per creare e configurare un dispositivo client per, consulta [AWS IoT Greengrass the section called “Modulo 4: Interazione con i dispositivi client in un gruppo AWS IoT Greengrass”](#) Oppure, per esempi che mostrano come utilizzare per creare e configurare un dispositivo client AWS IoT Greengrass, consulta [create-device-definition](#) la sezione AWS CLI Command Reference. AWS CLI

In un gruppo Greengrass, è possibile creare abbonamenti che consentono ai dispositivi client di comunicare tramite MQTT con funzioni Lambda, connettori e altri dispositivi client del gruppo e AWS IoT Core con o il servizio shadow locale. I messaggi MQTT vengono instradati tramite il core. Se il dispositivo principale perde la connettività al cloud, i dispositivi client possono continuare a comunicare sulla rete locale. Le dimensioni dei dispositivi client possono variare, da dispositivi più piccoli basati su microcontrollori a dispositivi di grandi dimensioni. Attualmente, un

gruppo Greengrass può contenere fino a 2.500 dispositivi client. Un dispositivo client può far parte di un massimo di 10 gruppi.

Note

OPC-UA è uno standard di scambio di informazioni per la comunicazione industriale. [Per implementare il supporto per OPC-UA sul core Greengrass, puoi utilizzare il connettore IoT. SiteWise](#) Il connettore invia i dati dei dispositivi industriali dai server OPC-UA alle proprietà degli asset in AWS IoT SiteWise

La tabella seguente mostra come questi tipi di dispositivi sono correlati tra loro.

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

Il dispositivo AWS IoT Greengrass principale archivia i certificati in due posizioni:

- Certificato dispositivo core in `/greengrass-root/certs` In genere, il certificato del dispositivo core è denominato `hash.cert.pem` (ad esempio `86c84488a5.cert.pem`). Questo certificato viene utilizzato dal AWS IoT client per l'autenticazione reciproca quando il core si connette ai AWS IoT Greengrass servizi AWS IoT Core and.
- Certificato server MQTT in `/greengrass-root/ggc/var/state/server`. Il certificato del server MQTT è denominato `server.crt`. Questo certificato viene utilizzato per l'autenticazione reciproca tra il server MQTT locale (nel core Greengrass) e i dispositivi Greengrass.

Note

`greengrass-root` rappresenta il percorso in cui il software AWS IoT Greengrass Core è installato sul dispositivo. In genere, questa è la directory `/greengrass`.

SDKs

I seguenti, AWS forniti, SDKs vengono utilizzati per lavorare con AWS IoT Greengrass:

AWS SDK

Usa l' AWS SDK per creare applicazioni che interagiscono con qualsiasi AWS servizio, tra cui Amazon S3, Amazon DynamoDB e altro ancora. AWS IoT AWS IoT Greengrass Nel contesto di AWS IoT Greengrass, puoi utilizzare l' AWS SDK nelle funzioni Lambda distribuite per effettuare chiamate dirette a qualsiasi servizio. AWS Per ulteriori informazioni, consulta [AWS SDKs](#).

Note

Le operazioni specifiche di Greengrass disponibili in sono disponibili anche nell'[AWS IoT Greengrass API](#) e. AWS SDKs [AWS CLI](#)

AWS IoT SDK del dispositivo

Il AWS IoT Device SDK aiuta i dispositivi a connettersi a AWS IoT Core e. AWS IoT Greengrass Per ulteriori informazioni, consulta [AWS IoT Device SDKs nella AWS IoT Developer Guide](#).

I dispositivi client possono utilizzare qualsiasi piattaforma AWS IoT Device SDK v2 per scoprire informazioni di connettività per un core Greengrass. Le informazioni sulla connettività includono:

- I IDs gruppi Greengrass a cui appartiene il dispositivo client.
- Gli indirizzi IP del core Greengrass in ogni gruppo. Questi sono anche chiamati endpoint principali.
- Il certificato CA di gruppo, utilizzato dai dispositivi per l'autenticazione reciproca con il core. Per ulteriori informazioni, consulta [the section called “Flusso di lavoro di connessione del dispositivo”](#).

Note

Nella versione 1 del AWS IoT dispositivo SDKs, solo le piattaforme C++ e Python forniscono supporto di discovery integrato.

AWS IoT Greengrass SDK principale

Il AWS IoT Greengrass Core SDK consente alle funzioni Lambda di interagire con il core Greengrass, pubblicare messaggi AWS IoT, interagire con il servizio shadow locale, richiamare altre funzioni Lambda distribuite e accedere a risorse segrete. Questo SDK viene utilizzato dalle funzioni Lambda eseguite su AWS IoT Greengrass un core. Per ulteriori informazioni, consulta [AWS IoT Greengrass SDK principale](#).

AWS IoT Greengrass SDK per Machine Learning

Il AWS IoT Greengrass Machine Learning SDK consente alle funzioni Lambda di utilizzare modelli di machine learning distribuiti nel core di Greengrass come risorse di apprendimento automatico. Questo SDK viene utilizzato dalle funzioni Lambda che vengono eseguite su AWS IoT Greengrass un core e interagiscono con un servizio di inferenza locale. Per ulteriori informazioni, consulta [AWS IoT Greengrass SDK per il Machine Learning](#).

Piattaforme supportate e requisiti

Le schede seguenti elencano le piattaforme e i requisiti supportati per il software Core. AWS IoT Greengrass

Note

È possibile scaricare il software AWS IoT Greengrass Core dalla sezione Download del [software AWS IoT Greengrass principale](#).

GGC v1.11

Piattaforme supportate:

- Architettura: Armv7l
 - Sistema operativo: Linux
 - Sistema operativo: Linux ([OpenWrt](#))
- Architettura: Armv8 () AArch64
 - Sistema operativo: Linux
 - Sistema operativo: Linux ([OpenWrt](#))
- Architettura: Armv6l
 - Sistema operativo: Linux
- Architettura: x86_64
 - Sistema operativo: Linux
- Le piattaforme Windows, macOS e Linux possono essere eseguite AWS IoT Greengrass in un contenitore Docker. Per ulteriori informazioni, consulta [the section called “Esegui AWS IoT Greengrass in un contenitore Docker”](#).

Requisiti:

- Almeno 128 MB di spazio su disco disponibile per il software AWS IoT Greengrass Core. Se si utilizza l'[agente di aggiornamento OTA](#), il minimo è 400 MB.
- Almeno 128 MB di RAM assegnati al software AWS IoT Greengrass Core. Con [stream manager](#) abilitato, il valore minimo è 198 MB di RAM.

 Note

Lo Stream Manager è abilitato per impostazione predefinita se si utilizza l'opzione di creazione del gruppo predefinito sulla AWS IoT console per creare il gruppo Greengrass.

- Versione kernel Linux:
 - [È richiesta la versione 4.4 o successiva del kernel Linux per supportare l'esecuzione AWS IoT Greengrass con contenitori.](#)

- È richiesta la versione 3.17 o successiva del kernel Linux per supportare l'esecuzione AWS IoT Greengrass senza contenitori. In questa configurazione, la containerizzazione predefinita della funzione Lambda per il gruppo Greengrass deve essere impostata su Nessun contenitore. Per istruzioni, consulta [the section called “Impostazione della containerizzazione predefinita per le funzioni Lambda in un gruppo”](#).
- [GNU C Library \(glibc\)](#) versione 2.14 o successiva. OpenWrt le distribuzioni richiedono [musl C Library](#) versione 1.1.16 o successiva.
- La directory `/var/run` deve essere presente sul dispositivo.
- I file `/dev/stdin`, `/dev/stdout` e `/dev/stderr` devono essere disponibili.
- La protezione hardlink e softlink deve essere attivata sul dispositivo. Altrimenti, AWS IoT Greengrass può essere eseguito solo in modalità non sicura, utilizzando il flag. `-i`
- Le seguenti configurazioni del kernel Linux devono essere abilitate sul dispositivo:
 - Spazio dei nomi:
 - `CONFIG_IPC_NS`
 - `CONFIG_UTS_NS`
 - `CONFIG_USER_NS`
 - `CONFIG_PID_NS`
 - Cgroups:
 - `CONFIG_CGROUP_DEVICE`
 - `CONFIG_CGROUPS`
 - `CONFIG_MEMCG`

Il kernel deve supportare [cgroups](#). I seguenti requisiti si applicano all'esecuzione AWS IoT Greengrass con [contenitori](#):

- Il memory cgroup deve essere abilitato e montato per consentire di AWS IoT Greengrass impostare il limite di memoria per le funzioni Lambda.
- Il device cgroup deve essere abilitato e montato se le funzioni Lambda [con accesso alle risorse locali](#) vengono utilizzate per aprire file AWS IoT Greengrass sul dispositivo principale.
- Altri:
 - `CONFIG_POSIX_MQUEUE`
 - `CONFIG_OVERLAY_FS`
 - `CONFIG_HAVE_ARCH_SECCOMP_FILTER`

- CONFIG_SECCOMP_FILTER
 - CONFIG_KEYS
 - CONFIG_SECCOMP
 - CONFIG_SHMEM
- Il certificato root per Amazon S3 AWS IoT deve essere presente nell'archivio di fiducia del sistema.
 - [Stream Manager](#) richiede il runtime Java 8 e un minimo di 70 MB di RAM oltre ai requisiti di memoria del software AWS IoT Greengrass Core di base. Lo Stream Manager è abilitato per impostazione predefinita quando si utilizza l'opzione di creazione del gruppo predefinito sulla AWS IoT console. Stream manager non è supportato nelle OpenWrt distribuzioni.
 - Librerie che supportano il [AWS Lambda runtime](#) richiesto dalle funzioni Lambda che desideri eseguire localmente. Le librerie obbligatorie devono essere installate sul core e aggiunte alla variabile di ambiente PATH. È possibile installare più librerie nello stesso core.
 - [Python](#) versione 3.8 per funzioni che utilizzano il runtime Python 3.8.
 - [Python](#) versione 3.7 per funzioni che usano il runtime Python 3.7.
 - [Python](#) versione 2.7 per funzioni che usano il runtime Python 2.7.
 - [Node.js](#) versione 12.x per le funzioni che utilizzano il runtime Node.js 12.x.
 - [Java](#) versione 8 o successive per funzioni che usano il runtime Java 8.

 Note

L'esecuzione di Java su una OpenWrt distribuzione non è supportata ufficialmente. Tuttavia, se la tua OpenWrt build supporta Java, potresti essere in grado di eseguire funzioni Lambda create in Java sui tuoi dispositivi. OpenWrt

Per ulteriori informazioni sul AWS IoT Greengrass supporto per i runtime Lambda, consulta.

[Esegui funzioni Lambda locali](#)

- I seguenti comandi di shell (non le BusyBox varianti) sono richiesti dall'agente di [aggiornamento over-the-air \(OTA\)](#):
 - wget
 - realpath
 - tar

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

GGC v1.10

Piattaforme supportate:

- Architettura: Armv7l
 - Sistema operativo: Linux
 - Sistema operativo: Linux ([OpenWrt](#))
- Architettura: Armv8 () AArch64
 - Sistema operativo: Linux
 - Sistema operativo: Linux ([OpenWrt](#))
- Architettura: Armv6l
 - Sistema operativo: Linux
- Architettura: x86_64
 - Sistema operativo: Linux

- Le piattaforme Windows, macOS e Linux possono essere eseguite AWS IoT Greengrass in un contenitore Docker. Per ulteriori informazioni, consulta [the section called “Esegui AWS IoT Greengrass in un contenitore Docker”](#).

Requisiti:

- Almeno 128 MB di spazio su disco disponibile per il software AWS IoT Greengrass Core. Se si utilizza l'[agente di aggiornamento OTA](#), il minimo è 400 MB.
- Almeno 128 MB di RAM assegnati al software AWS IoT Greengrass Core. Con [stream manager](#) abilitato, il valore minimo è 198 MB di RAM.



Note

Lo Stream Manager è abilitato per impostazione predefinita se si utilizza l'opzione di creazione del gruppo predefinito sulla AWS IoT console per creare il gruppo Greengrass.

- Versione kernel Linux:
 - [È richiesta la versione 4.4 o successiva del kernel Linux per supportare l'esecuzione AWS IoT Greengrass con contenitori.](#)
 - È richiesta la versione 3.17 o successiva del kernel Linux per supportare l'esecuzione AWS IoT Greengrass senza contenitori. In questa configurazione, la containerizzazione predefinita della funzione Lambda per il gruppo Greengrass deve essere impostata su Nessun contenitore. Per istruzioni, consulta [the section called “Impostazione della containerizzazione predefinita per le funzioni Lambda in un gruppo”](#).
- [GNU C Library \(glibc\)](#) versione 2.14 o successiva. OpenWrt le distribuzioni richiedono [musl C Library](#) versione 1.1.16 o successiva.
- La directory `/var/run` deve essere presente sul dispositivo.
- I file `/dev/stdin`, `/dev/stdout` e `/dev/stderr` devono essere disponibili.
- La protezione hardlink e softlink deve essere attivata sul dispositivo. Altrimenti, AWS IoT Greengrass può essere eseguito solo in modalità non sicura, utilizzando il flag. `-i`
- Le seguenti configurazioni del kernel Linux devono essere abilitate sul dispositivo:
 - Spazio dei nomi:
 - `CONFIG_IPC_NS`
 - `CONFIG_UTS_NS`

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

Il kernel deve supportare [cgroups](#). I seguenti requisiti si applicano all'esecuzione AWS IoT Greengrass con [contenitori](#):

- Il memory cgroup deve essere abilitato e montato per consentire di AWS IoT Greengrass impostare il limite di memoria per le funzioni Lambda.
- Il device cgroup deve essere abilitato e montato se le funzioni Lambda [con accesso alle risorse locali](#) vengono utilizzate per aprire file AWS IoT Greengrass sul dispositivo principale.
- Altri:
 - CONFIG_POSIX_MQUEUE
 - CONFIG_OVERLAY_FS
 - CONFIG_HAVE_ARCH_SECCOMP_FILTER
 - CONFIG_SECCOMP_FILTER
 - CONFIG_KEYS
 - CONFIG_SECCOMP
 - CONFIG_SHMEM
- Il certificato root per Amazon S3 AWS IoT deve essere presente nell'archivio di fiducia del sistema.
- [Stream Manager](#) richiede il runtime Java 8 e un minimo di 70 MB di RAM oltre ai requisiti di memoria del software AWS IoT Greengrass Core di base. Lo Stream Manager è abilitato per impostazione predefinita quando si utilizza l'opzione di creazione del gruppo predefinito sulla AWS IoT console. Stream manager non è supportato nelle OpenWrt distribuzioni.
- Librerie che supportano il [AWS Lambda runtime](#) richiesto dalle funzioni Lambda che desideri eseguire localmente. Le librerie obbligatorie devono essere installate sul core e aggiunte alla variabile di ambiente PATH. È possibile installare più librerie nello stesso core.
 - [Python](#) versione 3.7 per funzioni che usano il runtime Python 3.7.
 - [Python](#) versione 2.7 per funzioni che usano il runtime Python 2.7.

- [Node.js](#) versione 12.x per le funzioni che utilizzano il runtime Node.js 12.x.
- [Java](#) versione 8 o successive per funzioni che usano il runtime Java 8.

 Note

L'esecuzione di Java su una OpenWrt distribuzione non è supportata ufficialmente. Tuttavia, se la tua OpenWrt build supporta Java, potresti essere in grado di eseguire funzioni Lambda create in Java sui tuoi dispositivi. OpenWrt

Per ulteriori informazioni sul AWS IoT Greengrass supporto per i runtime Lambda, consulta.

[Esegui funzioni Lambda locali](#)

- I seguenti comandi di shell (non le BusyBox varianti) sono richiesti dall'agente di [aggiornamento over-the-air \(OTA\)](#):

- wget
- realpath
- tar
- readlink
- basename
- dirname
- pidof
- df
- grep
- umount
- mv
- gzip
- mkdir
- rm
- ln
- cut
- cat

- `/bin/bash`

GGC v1.9

Piattaforme supportate:

- Architettura: Armv7l
 - Sistema operativo: Linux
 - Sistema operativo: Linux ([OpenWrt](#))
- Architettura: Armv8 () AArch64
 - Sistema operativo: Linux
 - Sistema operativo: Linux ([OpenWrt](#))
- Architettura: Armv6l
 - Sistema operativo: Linux
- Architettura: x86_64
 - Sistema operativo: Linux
- Le piattaforme Windows, macOS e Linux possono essere eseguite AWS IoT Greengrass in un contenitore Docker. Per ulteriori informazioni, consulta [the section called “Esegui AWS IoT Greengrass in un contenitore Docker”](#).

Requisiti:

- Almeno 128 MB di spazio su disco disponibile per il software AWS IoT Greengrass Core. Se si utilizza l'[agente di aggiornamento OTA](#), il minimo è 400 MB.
- Almeno 128 MB di RAM assegnati al software AWS IoT Greengrass Core.
- Versione kernel Linux:
 - [È richiesta la versione 4.4 o successiva del kernel Linux per supportare l'esecuzione AWS IoT Greengrass con contenitori.](#)
 - È richiesta la versione 3.17 o successiva del kernel Linux per supportare l'esecuzione AWS IoT Greengrass senza contenitori. In questa configurazione, la containerizzazione predefinita della funzione Lambda per il gruppo Greengrass deve essere impostata su Nessun contenitore. Per istruzioni, consulta [the section called “Impostazione della containerizzazione predefinita per le funzioni Lambda in un gruppo”](#).
- [GNU C Library \(glibc\)](#) versione 2.14 o successiva. OpenWrt le distribuzioni richiedono [musl C Library](#) versione 1.1.16 o successiva.
- La directory `/var/run` deve essere presente sul dispositivo.

- I file `/dev/stdin`, `/dev/stdout` e `/dev/stderr` devono essere disponibili.
- La protezione hardlink e softlink deve essere attivata sul dispositivo. Altrimenti, AWS IoT Greengrass può essere eseguito solo in modalità non sicura, utilizzando il flag. `-i`
- Le seguenti configurazioni del kernel Linux devono essere abilitate sul dispositivo:
 - Spazio dei nomi:
 - `CONFIG_IPC_NS`
 - `CONFIG_UTS_NS`
 - `CONFIG_USER_NS`
 - `CONFIG_PID_NS`
 - Cgroups:
 - `CONFIG_CGROUP_DEVICE`
 - `CONFIG_CGROUPS`
 - `CONFIG_MEMCG`

Il kernel deve supportare [cgroups](#). I seguenti requisiti si applicano all'esecuzione AWS IoT Greengrass con [contenitori](#):

- Il memory cgroup deve essere abilitato e montato per consentire di AWS IoT Greengrass impostare il limite di memoria per le funzioni Lambda.
- Il device cgroup deve essere abilitato e montato se le funzioni Lambda [con accesso alle risorse locali](#) vengono utilizzate per aprire file AWS IoT Greengrass sul dispositivo principale.
- Altri:
 - `CONFIG_POSIX_MQUEUE`
 - `CONFIG_OVERLAY_FS`
 - `CONFIG_HAVE_ARCH_SECCOMP_FILTER`
 - `CONFIG_SECCOMP_FILTER`
 - `CONFIG_KEYS`
 - `CONFIG_SECCOMP`
 - `CONFIG_SHMEM`
- Il certificato root per Amazon S3 AWS IoT deve essere presente nell'archivio di fiducia del sistema.

- Librerie che supportano il [AWS Lambda runtime](#) richiesto dalle funzioni Lambda che desideri eseguire localmente. Le librerie obbligatorie devono essere installate sul core e aggiunte alla variabile di ambiente PATH. È possibile installare più librerie nello stesso core.
- [Python](#) versione 2.7 per funzioni che usano il runtime Python 2.7.
- [Python](#) versione 3.7 per funzioni che usano il runtime Python 3.7.
- [Node.js](#) versione 6.10 o successive per funzioni che usano il runtime Node.js 6.10.
- [Node.js](#) versione 8.10 o successive per funzioni che usano il runtime Node.js 8.10.
- [Java](#) versione 8 o successive per funzioni che usano il runtime Java 8.

 Note

L'esecuzione di Java su una OpenWrt distribuzione non è supportata ufficialmente. Tuttavia, se la tua OpenWrt build supporta Java, potresti essere in grado di eseguire funzioni Lambda create in Java sui tuoi dispositivi. OpenWrt

Per ulteriori informazioni sul AWS IoT Greengrass supporto per i runtime Lambda, consulta.

[Esegui funzioni Lambda locali](#)

- I seguenti comandi di shell (non le BusyBox varianti) sono richiesti dall'agente di [aggiornamento over-the-air \(OTA\)](#):
 - `wget`
 - `realpath`
 - `tar`
 - `readlink`
 - `basename`
 - `dirname`
 - `pidof`
 - `df`
 - `grep`
 - `umount`
 - `mv`
 - `gzip`
 - `mkdir`

- rm
- ln
- cut
- cat

GGC v1.8

- Piattaforme supportate:
 - Architettura: ARMv7L; sistema operativo: Linux
 - Architettura: x86_64; Sistema operativo: Linux
 - Architettura: Armv8 (AArch64); Sistema operativo: Linux
 - Le piattaforme Windows, macOS e Linux possono essere eseguite AWS IoT Greengrass in un contenitore Docker. Per ulteriori informazioni, consulta [the section called “Esegui AWS IoT Greengrass in un contenitore Docker”](#).
 - [Le piattaforme Linux possono eseguire una versione AWS IoT Greengrass con funzionalità limitate utilizzando lo snap Greengrass, disponibile tramite Snapcraft](#). Per ulteriori informazioni, consulta [the section called “AWS IoT Greengrass software snap”](#).
- I seguenti elementi sono obbligatori:
 - Almeno 128 MB di spazio su disco disponibile per il software AWS IoT Greengrass Core. Se si utilizza l'[agente di aggiornamento OTA](#), il minimo è 400 MB.
 - Almeno 128 MB di RAM assegnati al software AWS IoT Greengrass Core.
 - Versione kernel Linux:
 - [È richiesta la versione 4.4 o successiva del kernel Linux per supportare l'esecuzione AWS IoT Greengrass con contenitori](#).
 - È richiesta la versione 3.17 o successiva del kernel Linux per supportare l'esecuzione AWS IoT Greengrass senza contenitori. In questa configurazione, la containerizzazione predefinita della funzione Lambda per il gruppo Greengrass deve essere impostata su Nessun contenitore. Per istruzioni, consulta [the section called “Impostazione della containerizzazione predefinita per le funzioni Lambda in un gruppo”](#).
 - [Libreria GNU C](#) (glibc) versione 2.14 o successiva.
 - La directory /var/run deve essere presente sul dispositivo.
 - I file /dev/stdin, /dev/stdout e /dev/stderr devono essere disponibili.

- La protezione hardlink e softlink deve essere attivata sul dispositivo. Altrimenti, AWS IoT Greengrass può essere eseguito solo in modalità non sicura, utilizzando il flag. `-i`
- Le seguenti configurazioni del kernel Linux devono essere abilitate sul dispositivo:

- Spazio dei nomi:
 - CONFIG_IPC_NS
 - CONFIG_UTS_NS
 - CONFIG_USER_NS
 - CONFIG_PID_NS
- Cgroups:
 - CONFIG_CGROUP_DEVICE
 - CONFIG_CGROUPS
 - CONFIG_MEMCG

Il kernel deve supportare [cgroups](#). I seguenti requisiti si applicano all'esecuzione AWS IoT Greengrass con [contenitori](#):

- Il memory cgroup deve essere abilitato e montato per consentire di AWS IoT Greengrass impostare il limite di memoria per le funzioni Lambda.
- Il device cgroup deve essere abilitato e montato se le funzioni Lambda [con accesso alle risorse locali](#) vengono utilizzate per aprire file AWS IoT Greengrass sul dispositivo principale.
- Altri:
 - CONFIG_POSIX_MQUEUE
 - CONFIG_OVERLAY_FS
 - CONFIG_HAVE_ARCH_SECCOMP_FILTER
 - CONFIG_SECCOMP_FILTER
 - CONFIG_KEYS
 - CONFIG_SECCOMP
 - CONFIG_SHMEM
- Il certificato root per Amazon S3 AWS IoT deve essere presente nell'archivio di fiducia del sistema.
- I seguenti elementi sono obbligatori in base a condizioni:

- Librerie che supportano il [AWS Lambda runtime](#) richiesto dalle funzioni Lambda che desideri eseguire localmente. Le librerie obbligatorie devono essere installate sul core e aggiunte alla variabile di ambiente PATH. È possibile installare più librerie nello stesso core.
- [Python](#) versione 2.7 per funzioni che usano il runtime Python 2.7.
- [Node.js](#) versione 6.10 o successive per funzioni che usano il runtime Node.js 6.10.
- [Java](#) versione 8 o successive per funzioni che usano il runtime Java 8.
- I seguenti comandi di shell (non le BusyBox varianti) sono richiesti dall'[agente di aggiornamento over-the-air \(OTA\)](#):
 - `wget`
 - `realpath`
 - `tar`
 - `readlink`
 - `basename`
 - `dirname`
 - `pidof`
 - `df`
 - `grep`
 - `umount`
 - `mv`
 - `gzip`
 - `mkdir`
 - `rm`
 - `ln`
 - `cut`
 - `cat`

Per informazioni sulle AWS IoT Greengrass quote (limiti), vedere [Service Quotas](#) in Riferimenti generali di Amazon Web Services

Per informazioni sui prezzi, consulta [Prezzi di AWS IoT Greengrass](#) e [Prezzi di AWS IoT Core](#).

AWS IoT Greengrass download

È possibile utilizzare le informazioni riportate di seguito per trovare e scaricare il software da utilizzare con AWS IoT Greengrass.

Argomenti

- [AWS IoT Greengrass Software di base](#)
- [AWS IoT Greengrass software snap](#)
- [AWS IoT Greengrass Software Docker](#)
- [AWS IoT Greengrass SDK principale](#)
- [Runtime e librerie di Machine Learning supportati](#)
- [AWS IoT Greengrass Software ML SDK](#)

AWS IoT Greengrass Software di base

Il software AWS IoT Greengrass Core estende AWS le funzionalità su un dispositivo AWS IoT Greengrass principale, permettendo ai dispositivi locali di agire localmente sui dati che generano.

v1.11

1.11.6

Correzione di bug e miglioramenti:

- Maggiore resilienza in caso di improvvisa perdita di alimentazione durante un'implementazione.
- È stato risolto un problema a causa del quale il danneggiamento dei dati dello stream manager poteva impedire l'avvio del software AWS IoT Greengrass Core.
- È stato risolto un problema per cui i nuovi dispositivi client non potevano connettersi al core in determinati scenari.
- È stato risolto un problema a causa del quale i nomi degli stream manager non potevano contenere `.log`.

1.11.5

Correzione di bug e miglioramenti:

- Miglioramenti delle prestazioni generali e correzioni di bug.

1.11.4

Correzione di bug e miglioramenti:

- È stato risolto un problema con lo stream manager che impediva gli aggiornamenti al AWS IoT Greengrass software Core v1.11.3. Se utilizzi stream manager per esportare dati nel cloud, ora puoi utilizzare un aggiornamento OTA per aggiornare una versione precedente v1.x del software Core alla v1.11.4. AWS IoT Greengrass
- Miglioramenti delle prestazioni generali e correzioni di bug.

1.11.3

Correzione di bug e miglioramenti:

- È stato risolto un problema che impediva al software AWS IoT Greengrass Core, eseguito in un attimo, di rispondere in un attimo su un dispositivo Ubuntu dopo un'improvvisa interruzione dell'alimentazione del dispositivo.
- È stato risolto un problema che causava un ritardo nella consegna dei messaggi MQTT a funzioni Lambda di lunga durata.
- È stato risolto un problema che impediva l'invio corretto dei messaggi MQTT quando il `maxWorkItemCount` valore era impostato su un valore maggiore di. 1024
- È stato risolto un problema che faceva sì che l'agente di aggiornamento OTA ignorasse il KeepAlive periodo MQTT specificato nella keepAlive proprietà in. [config.json](#)
- Miglioramenti delle prestazioni generali e correzioni di bug.

Important

Se utilizzi stream manager per esportare dati nel cloud, non eseguire l'aggiornamento al software AWS IoT Greengrass Core v1.11.3 da una versione precedente v1.x. Se stai abilitando lo stream manager per la prima volta, ti consigliamo vivamente di installare prima la versione più recente del software Core. AWS IoT Greengrass

1.11.1

Correzione di bug e miglioramenti:

- È stato risolto un problema che causava un maggiore utilizzo della memoria per lo stream manager.

- È stato risolto un problema che causava il ripristino del numero di sequenza dello stream 0 se il dispositivo principale Greengrass era spento per un periodo più lungo del periodo specificato time-to-live (TTL) dei dati del flusso.
- È stato risolto un problema che impediva allo stream manager di interrompere correttamente i tentativi di esportare i dati in Cloud AWS

1.11.0

Nuove caratteristiche:

- Un agente di telemetria sul core di Greengrass raccoglie dati di telemetria locali e li pubblica su Cloud AWS. Per recuperare i dati di telemetria per un'ulteriore elaborazione, i clienti possono creare una EventBridge regola Amazon e iscriversi a un target. Per ulteriori informazioni, consulta [Raccolta di dati di telemetria sanitaria del sistema](#) dai dispositivi principali. AWS IoT Greengrass
- Un'API HTTP locale restituisce un'istantanea dello stato corrente dei processi di lavoro locali avviati da AWS IoT Greengrass. Per ulteriori informazioni, consulta [Chiamare l'API per il controllo sanitario locale](#).
- Un [gestore di flussi](#) esporta automaticamente i dati in Amazon S3 e AWS IoT SiteWise

I nuovi [parametri dello stream manager](#) consentono di aggiornare gli stream esistenti e sospendere o riprendere l'esportazione dei dati.

- Support per l'esecuzione di funzioni Lambda di Python 3.8.x sul core.
- Una nuova `ggDaemonPort` proprietà da utilizzare per configurare il numero di porta IPC principale di Greengrass. [config.json](#) Il numero di porta predefinito è 8000.

Una nuova `systemComponentAuthTimeout` proprietà [config.json](#) che si utilizza per configurare il timeout per l'autenticazione IPC di base di Greengrass. Il timeout predefinito è di 5000 millisecondi.

- Il numero massimo di AWS IoT dispositivi per AWS IoT Greengrass gruppo è stato aumentato da 200 a 2500.

È stato aumentato il numero massimo di abbonamenti per gruppo da 1000 a 10000.

Per ulteriori informazioni, consulta [Endpoint e quote per AWS IoT Greengrass](#).

Correzione di bug e miglioramenti:

- Ottimizzazione generale che può ridurre l'utilizzo della memoria dei processi di servizio Greengrass.

- Un nuovo parametro di configurazione di runtime (`mountAllBlockDevices`) consente a Greengrass di utilizzare i bind mount per montare tutti i dispositivi a blocchi in un contenitore dopo aver configurato OverlayFS. Questa funzionalità ha risolto un problema che causava il fallimento della distribuzione di Greengrass se `/usr` non rientrava nella `/` gerarchia.
- È stato risolto un problema che causava un errore di AWS IoT Greengrass base se `/tmp` si trattava di un collegamento simbolico.
- È stato risolto un problema che consentiva all'agente di distribuzione Greengrass di rimuovere gli artefatti del modello di machine learning inutilizzati dalla cartella `mlmodel_public`.
- Miglioramenti delle prestazioni generali e correzioni di bug.

[Per installare il software AWS IoT Greengrass Core sul dispositivo principale, scaricate il pacchetto per l'architettura e il sistema operativo \(OS\) in uso, quindi seguite i passaggi indicati nella Guida introduttiva.](#)

 Tip

AWS IoT Greengrass fornisce anche altre opzioni per l'installazione del software AWS IoT Greengrass Core. Ad esempio, è possibile utilizzare la [configurazione del dispositivo Greengrass](#) per configurare l'ambiente e installare la versione più recente del software AWS IoT Greengrass Core. Oppure, sulle piattaforme Debian supportate, è possibile utilizzare il [gestore di pacchetti APT](#) per installare o aggiornare il AWS IoT Greengrass software Core. Per ulteriori informazioni, consulta [the section called “Installare il software AWS IoT Greengrass Core”](#).

Architettura	Sistema operativo	Link
Arm v8 () AArch64	Linux	Scarica
Arm v8 () AArch64	Linux () OpenWrt	Scarica
Armv7l	Linux	Scarica
Armv7l	Linux (OpenWrt)	Scarica
Armv6l	Linux	Scarica

Architettura	Sistema operativo	Link
x86_64	Linux	Scarica

Extended life versions

1.10.5

Le nuove caratteristiche di v1.10:

- Un gestore di flussi che elabora i flussi di dati localmente e li esporta automaticamente. Cloud AWS Questa funzionalità richiede Java 8 sul dispositivo core Greengrass. Per ulteriori informazioni, consulta [Gestione dei flussi di dati](#).
- Un nuovo connettore di distribuzione dell'applicazione Docker Greengrass che esegue un'applicazione Docker su un dispositivo core. Per ulteriori informazioni, consulta [the section called "Distribuzione dell'applicazione Docker"](#).
- Un nuovo SiteWise connettore IoT che invia i dati dei dispositivi industriali dai server OPC-UA alle proprietà degli asset. AWS IoT SiteWise Per ulteriori informazioni, consulta [the section called "IoT SiteWise"](#).
- Le funzioni Lambda eseguite senza containerizzazione possono accedere alle risorse di machine learning del gruppo Greengrass. Per ulteriori informazioni, consulta [the section called "Accesso alle risorse di Machine Learning"](#).
- Support per sessioni persistenti MQTT con AWS IoT. Per ulteriori informazioni, consulta [the section called "Sessioni persistenti MQTT con AWS IoT Core"](#).
- Il traffico MQTT locale può viaggiare su una porta diversa dalla porta predefinita 8883. Per ulteriori informazioni, consulta [the section called "Porta MQTT per la messaggistica locale"](#).
- Nuove queueFullPolicy opzioni nel [AWS IoT Greengrass Core SDK](#) per una pubblicazione affidabile dei messaggi dalle funzioni Lambda.
- Support per l'esecuzione delle funzioni Lambda di Node.js 12.x sul core.

Correzione di bug e miglioramenti:

- Over-the-air Gli aggiornamenti (OTA) con integrazione della sicurezza hardware possono essere configurati con OpenSSL 1.1.
- [Stream manager](#) è più resiliente al danneggiamento dei dati dei file.
- Risolto un problema che causava un errore di montaggio di sysfs sui dispositivi che utilizzavano Linux kernel 5.1 e versioni successive.

- Una nuova `mqttOperationTimeout` proprietà in [config.json](#) che consente di impostare il timeout per le operazioni di pubblicazione, sottoscrizione e annullamento dell'iscrizione nelle connessioni MQTT con AWS IoT Core
- È stato risolto un problema che causava un maggiore utilizzo della memoria per lo stream manager.
- Una nuova `systemComponentAuthTimeout` proprietà [config.json](#) che si utilizza per configurare il timeout per l'autenticazione IPC di base di Greengrass. Il timeout predefinito è di 5000 millisecondi.
- È stato risolto un problema che faceva sì che l'agente di aggiornamento OTA ignorasse il KeepAlive periodo MQTT specificato nella proprietà in `keepAlive` [config.json](#)
- È stato risolto un problema che impediva l'invio corretto dei messaggi MQTT quando il `maxWorkItemCount` valore era impostato su un valore maggiore di 1024
- È stato risolto un problema che causava un ritardo nella consegna dei messaggi MQTT a funzioni Lambda di lunga durata.
- È stato risolto un problema che impediva al software AWS IoT Greengrass Core, eseguito in un attimo, di rispondere in un attimo su un dispositivo Ubuntu dopo un'improvvisa interruzione dell'alimentazione del dispositivo.
- Miglioramenti delle prestazioni generali e correzioni di bug.

Per installare il software AWS IoT Greengrass Core sul tuo dispositivo principale, scarica il pacchetto per la tua architettura e il tuo sistema operativo (OS), quindi segui i passaggi indicati nella [Guida introduttiva](#).

Architettura	Sistema operativo	Link
Arm v8 () AArch64	Linux	Scarica
Arm v8 () AArch64	Linux () OpenWrt	Scarica
Armv7l	Linux	Scarica
Armv7l	Linux (OpenWrt)	Scarica
Armv6l	Linux	Scarica
x86_64	Linux	Scarica

1.9.4

Le nuove caratteristiche di v1.9:

- Support per i runtime Lambda di Python 3.7 e Node.js 8.10. Le funzioni Lambda che utilizzano i runtime Python 3.7 e Node.js 8.10 ora possono essere eseguite su un core. AWS IoT Greengrass (AWS IoT Greengrass continua a supportare i runtime Python 2.7 e Node.js 6.10.)
- Connessioni MQTT ottimizzate. Il core Greengrass stabilisce meno connessioni con AWS IoT Core. Questa modifica è in grado di ridurre i costi operativi per le spese basate sul numero di connessioni.
- Chiave Elliptic Curve (EC) per il server MQTT locale. Il server locale MQTT supporta le chiavi EC in aggiunta alle chiavi RSA. Il certificato del server MQTT ha una firma RSA SHA-256, indipendentemente dal tipo di chiave. Per ulteriori informazioni, consulta [the section called “Principal di sicurezza”](#).
- Support per [OpenWrt](#). AWS IoT Greengrass Il software di base v1.9.2 o successivo può essere installato su OpenWrt distribuzioni con architetture Armv8 () e ARMv7L. AArch64 OpenWrt Attualmente, non supporta l'inferenza ML.
- Support per ARMv6L. AWS IoT Greengrass Il software di base v1.9.3 o successivo può essere installato su distribuzioni Raspbian su architetture ARMv6L (ad esempio, su dispositivi Raspberry Pi Zero).
- Aggiornamenti OTA sulla porta 443 con ALPN. I core Greengrass che utilizzano la porta 443 per il traffico MQTT ora supportano gli aggiornamenti software over-the-air (OTA). AWS IoT Greengrass utilizza l'estensione TLS Application Layer Protocol Network (ALPN) per abilitare queste connessioni. Per ulteriori informazioni, consulta [Aggiornamenti OTA del software AWS IoT Greengrass Core](#) e [the section called “Connessione alla porta 443 o tramite un proxy di rete”](#).

Per installare il software AWS IoT Greengrass Core sul dispositivo principale, scaricate il pacchetto per l'architettura e il sistema operativo (OS) in uso, quindi seguite i passaggi indicati nella [Guida introduttiva](#).

Architettura	Sistema operativo	Link
Arm v8 () AArch64	Linux	Scarica
Arm v8 () AArch64	Linux () OpenWrt	Scarica

Architettura	Sistema operativo	Link
Armv7l	Linux	Scarica
Armv7l	Linux (OpenWrt)	Scarica
Armv6l	Linux	Scarica
x86_64	Linux	Scarica

1.8.4

- Nuove caratteristiche:
 - Identità di accesso predefinita configurabile per le funzioni Lambda nel gruppo. Questa impostazione a livello di gruppo determina le autorizzazioni predefinite utilizzate per eseguire le funzioni Lambda. Puoi impostare l'ID utente, l'ID gruppo o entrambi. Le singole funzioni Lambda possono sovrascrivere l'identità di accesso predefinita del relativo gruppo. Per ulteriori informazioni, consulta [the section called “Impostazione dell'identità di accesso predefinita per le funzioni Lambda in un gruppo”](#).
 - Traffico HTTPS sulla porta 443. La comunicazione HTTPS può essere configurata per viaggiare sulla porta 443 anziché sulla porta predefinita 8443. Ciò integra il AWS IoT Greengrass supporto per l'estensione TLS Application Layer Protocol Network (ALPN) e consente a tutto il traffico di messaggistica Greengrass, sia MQTT che HTTPS, di utilizzare la porta 443. Per ulteriori informazioni, consulta [the section called “Connessione alla porta 443 o tramite un proxy di rete”](#).
 - Client IDs per AWS IoT le connessioni con nome prevedibile. Questa modifica abilita il supporto per AWS IoT Device Defender e gli [eventi del ciclo di vita AWS IoT](#), in modo da poter ricevere notifiche per eventi di connessione, disconnessione, sottoscrizione e annullamento sottoscrizione. La denominazione prevedibile semplifica inoltre la creazione di una logica relativa alla connessione IDs (ad esempio, la creazione di modelli di [policy di sottoscrizione](#) basati sugli attributi dei certificati). Per ulteriori informazioni, consulta [the section called “Client IDs per connessioni MQTT con AWS IoT”](#).

Correzione di bug e miglioramenti:

- È stato risolto un problema relativo alla sincronizzazione shadow e alla riconnessione di Device Certificate Manager (DCM).
- Miglioramenti delle prestazioni generali e correzioni di bug.

Per installare il software AWS IoT Greengrass Core sul dispositivo principale, scaricate il pacchetto per l'architettura e il sistema operativo (OS) in uso, quindi seguite i passaggi indicati nella [Guida introduttiva](#).

Architettura	Sistema operativo	Link
Arm v8 () AArch64	Linux	Scarica
Armv7l	Linux	Scarica
x86_64	Linux	Scarica

Scaricando questo software accetti l'[Accordo di licenza del software Greengrass Core](#).

Per informazioni su altre opzioni per l'installazione del software AWS IoT Greengrass Core sul dispositivo, consulta [the section called “Installare il software AWS IoT Greengrass Core”](#)

AWS IoT Greengrass software snap

AWS IoT Greengrass snap 1.11.x consente di eseguire una versione limitata AWS IoT Greengrass tramite comodi pacchetti software, insieme a tutte le dipendenze necessarie, in un ambiente containerizzato.

Note

Lo AWS IoT Greengrass snap è disponibile per il software Core v1.11.x. AWS IoT Greengrass AWS IoT Greengrass non fornisce uno snap per la v1.10.x. Le versioni non supportate non ricevono correzioni di bug o aggiornamenti.
Lo AWS IoT Greengrass snap non supporta i connettori e l'inferenza di machine learning (ML).

Per ulteriori informazioni, consulta [the section called “Corri AWS IoT Greengrass in un attimo”](#).

AWS IoT Greengrass Software Docker

AWS fornisce un Dockerfile e immagini Docker che semplificano l'esecuzione AWS IoT Greengrass in un contenitore Docker.

Dockerfile

I Dockerfile contengono il codice sorgente per la creazione di immagini di container personalizzate. AWS IoT Greengrass L'immagine può essere modificata per essere eseguita su diverse architetture di piattaforma o per ridurne le dimensioni. Per le istruzioni, consulta il file README.

Scarica la versione del software AWS IoT Greengrass Core di destinazione.

v1.11

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

Extended life versions

v1.10

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

v1.9

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

v1.8

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

immagine Docker

Le immagini Docker hanno il software AWS IoT Greengrass Core e le dipendenze installati nelle immagini di base di Amazon Linux 2 (x86_64) e Alpine Linux (x86_64, ARMv7L o). AArch64 È possibile utilizzare immagini predefinite per iniziare a sperimentare AWS IoT Greengrass.

Important

Il 30 giugno 2022, AWS IoT Greengrass è terminata la manutenzione delle immagini Docker del software AWS IoT Greengrass Core v1.x pubblicate su Amazon Elastic

Container Registry (Amazon ECR) e Docker Hub. Puoi continuare a scaricare queste immagini Docker da Amazon ECR e Docker Hub fino al 30 giugno 2023, ovvero 1 anno dopo la fine della manutenzione. Tuttavia, le immagini Docker del software AWS IoT Greengrass Core v1.x non ricevono più patch di sicurezza o correzioni di bug dopo la fine della manutenzione il 30 giugno 2022. Se esegui un carico di lavoro di produzione che dipende da queste immagini Docker, ti consigliamo di creare le tue immagini Docker utilizzando i Dockerfile forniti. AWS IoT Greengrass Per ulteriori informazioni, consulta [AWS IoT Greengrass Version 1 politica di manutenzione](#).

Scarica un'immagine predefinita da [Docker Hub](#) o Amazon Elastic Container Registry (Amazon ECR).

- Per Docker Hub, usa il **version** tag per scaricare una versione specifica dell'immagine Greengrass Docker. Per trovare i tag per tutte le immagini disponibili, controlla la pagina Tag nell'Hub Docker.
- Per Amazon ECR, usa il `latest` tag per scaricare l'ultima versione disponibile dell'immagine Greengrass Docker. Per ulteriori informazioni sull'elenco delle versioni di immagini disponibili e sul download di immagini da Amazon ECR, consulta [Esecuzione AWS IoT Greengrass in un contenitore Docker](#).

Warning

A partire dalla v1.11.6 del software AWS IoT Greengrass Core, le immagini Greengrass Docker non includono più Python 2.7, perché Python 2.7 end-of-life è arrivato nel 2020 e non riceve più aggiornamenti di sicurezza. Se scegli di eseguire l'aggiornamento a queste immagini Docker, ti consigliamo di verificare che le tue applicazioni funzionino con le nuove immagini Docker prima di distribuire gli aggiornamenti sui dispositivi di produzione. Se hai bisogno di Python 2.7 per la tua applicazione che utilizza un'immagine Greengrass Docker, puoi modificare il Dockerfile Greengrass per includere Python 2.7 per la tua applicazione.

AWS IoT Greengrass non fornisce immagini Docker per il software Core v1.11.1. AWS IoT Greengrass

Note

Per impostazione predefinita, le immagini `alpine-aarch64` e `alpine-armv7l` possono essere eseguite solo su host basati su ARM. Per eseguire queste immagini su un host x86, è possibile installare [QEMU](#) e montare le librerie QEMU sull'host. Per esempio:

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

AWS IoT Greengrass SDK principale

Le funzioni Lambda utilizzano il AWS IoT Greengrass Core SDK per interagire con il AWS IoT Greengrass core a livello locale. Ciò consente alle funzioni Lambda implementate di:

- Scambia messaggi MQTT con. AWS IoT Core
- Scambia messaggi MQTT con connettori, dispositivi client e altre funzioni Lambda nel gruppo Greengrass.
- Interagire con il servizio shadow locale.
- Richiama altre funzioni Lambda locali.
- Accesso a [risorse segrete](#).
- Interagire con [stream manager](#).

Scarica il AWS IoT Greengrass Core SDK per la tua lingua o piattaforma da. GitHub

- [AWS IoT Greengrass SDK principale per Java](#)
- [AWS IoT Greengrass Core SDK per Node.js](#)
- [AWS IoT Greengrass SDK principale per Python](#)
- [AWS IoT Greengrass Core SDK per C](#)

Per ulteriori informazioni, consulta [AWS IoT Greengrass SDK principale](#).

Runtime e librerie di Machine Learning supportati

Per [eseguire l'inferenza](#) su un core Greengrass, è necessario installare il runtime o la libreria di Machine Learning per il tipo di modello ML.

AWS IoT Greengrass supporta i seguenti tipi di modelli ML. Utilizzare questi collegamenti per trovare informazioni su come installare il runtime o la libreria per il tipo di modello e la piattaforma del dispositivo.

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

Esempi di Machine Learning

AWS IoT Greengrass fornisce esempi che è possibile utilizzare con i runtime e le librerie ML supportati. Questi esempi vengono rilasciati con l'[Accordo di licenza del software Greengrass Core](#).

Deep learning runtime (DLR)

Scaricare l'esempio per la piattaforma del proprio dispositivo:

- Esempio DLR per [Raspberry Pi](#)
- [Esempio DLR per NVIDIA Jetson TX2](#)
- Esempio DLR per [Intel Atom](#)

Per un tutorial che utilizza l'esempio DLR, consulta [the section called “Come configurare l'inferenza di Machine Learning ottimizzata”](#).

MXNet

Scaricare l'esempio per la piattaforma del proprio dispositivo:

- MXNet [esempio per Raspberry Pi](#)
- MXNet esempio per [NVIDIA Jetson TX2](#)
- MXNet [esempio per Intel Atom](#)

Per un tutorial che utilizza l' MXNet esempio, vedi [the section called “Come configurare l'inferenza di Machine Learning”](#).

TensorFlow

Scarica l'[esempio Tensorflow](#) per la piattaforma del tuo dispositivo. Questo esempio funziona con Raspberry Pi, NVIDIA Jetson TX2 e Intel Atom.

AWS IoT Greengrass Software ML SDK

[AWS IoT Greengrass SDK per il Machine Learning](#) Consente alle funzioni Lambda create di utilizzare un modello di machine learning locale e inviare dati al connettore [ML Feedback](#) per il caricamento e la pubblicazione.

v1.1.0

- [Python 3.7](#).

v1.0.0

- [Python 2.7](#).

Ci piacerebbe conoscere la tua opinione

Appreziamo il tuo feedback. [Per contattarci, visita AWS re:POST e usa il tag.AWS IoT Greengrass](#)

Installare il software AWS IoT Greengrass Core

Il software AWS IoT Greengrass Core estende AWS le funzionalità su un dispositivo AWS IoT Greengrass principale, permettendo ai dispositivi locali di agire localmente sui dati che generano.

AWS IoT Greengrass offre diverse opzioni per l'installazione del software AWS IoT Greengrass Core:

- [Scaricare ed estrarre un file tar.gz](#).
- [Eseguire lo script di installazione dispositivo Greengrass](#).
- [Installare da un repository APT](#).

AWS IoT Greengrass fornisce anche ambienti containerizzati che eseguono il software AWS IoT Greengrass Core.

- [Esegui AWS IoT Greengrass in un contenitore Docker.](#)
- [Esegui AWS IoT Greengrass in un attimo.](#)

Scarica ed estrai il pacchetto software AWS IoT Greengrass Core

Scegliete il software AWS IoT Greengrass Core per la vostra piattaforma da scaricare come file tar.gz ed estrarlo sul dispositivo. Puoi scaricare le versioni recenti del software. Per ulteriori informazioni, consulta [the section called “AWS IoT Greengrass Software di base”](#).

Eseguire lo script di installazione del dispositivo Greengrass

Esegui la configurazione del dispositivo Greengrass per configurare il dispositivo, installare l'ultima versione del software AWS IoT Greengrass Core e implementare una funzione Hello World Lambda in pochi minuti. Per ulteriori informazioni, consulta [the section called “Avvio rapido: configurazione dispositivo Greengrass”](#).

Installa il software AWS IoT Greengrass Core da un repository APT

Important

A partire dall'11 febbraio 2022, non è più possibile installare o aggiornare il software AWS IoT Greengrass Core da un repository APT. Sui dispositivi in cui è stato aggiunto il AWS IoT Greengrass repository, è necessario [rimuovere il repository dall'elenco delle fonti](#). I dispositivi che eseguono il software dal repository APT continueranno a funzionare normalmente. Si consiglia di aggiornare il software AWS IoT Greengrass Core utilizzando i file [tar](#).

Il repository APT fornito da AWS IoT Greengrass include i seguenti pacchetti:

- `aws-iot-greengrass-core`. Installa il software AWS IoT Greengrass Core.

- `aws-iot-greengrass-keyring`. Installa le chiavi GnuPG (GPG) usate per firmare l'archivio dei pacchetti. AWS IoT Greengrass

Scaricando questo software accetti l'[Accordo di licenza del software Greengrass Core](#).

Argomenti

- [Utilizzare gli script systemd per gestire il ciclo di vita del daemon Greengrass](#)
- [Disinstallate il software di base utilizzando il repository APT AWS IoT Greengrass](#)
- [Rimuovi i sorgenti del repository software AWS IoT Greengrass principale](#)

Utilizzare gli script systemd per gestire il ciclo di vita del daemon Greengrass

Il `aws-iot-greengrass-core` pacchetto installa anche systemd script che è possibile utilizzare per gestire il ciclo di vita del software Core (daemon). AWS IoT Greengrass

- Per avviare il demone Greengrass durante l'avvio:

```
systemctl enable greengrass.service
```

- Per avviare il demone Greengrass:

```
systemctl start greengrass.service
```

- Arrestare il daemon Greengrass.

```
systemctl stop greengrass.service
```

- Per verificare lo stato del demone Greengrass:

```
systemctl status greengrass.service
```

Disinstallate il software di base utilizzando il repository APT AWS IoT Greengrass

Quando si disinstalla il software di AWS IoT Greengrass base, è possibile scegliere se conservare o rimuovere le informazioni di configurazione del software di AWS IoT Greengrass base, come i certificati dei dispositivi, le informazioni sul gruppo e i file di registro.

Per disinstallare il software di AWS IoT Greengrass base e conservare le informazioni di configurazione

- Esegui il comando seguente per rimuovere i pacchetti software AWS IoT Greengrass principali e conservare le informazioni di configurazione nella `/greengrass` cartella.

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

Per disinstallare il software di AWS IoT Greengrass base e rimuovere le informazioni di configurazione

1. Esegui il comando seguente per rimuovere i pacchetti software AWS IoT Greengrass principali e rimuovere le informazioni di configurazione da `/greengrass` folder.

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

2. Rimuovi il repository del software AWS IoT Greengrass principale dall'elenco dei sorgenti. Per ulteriori informazioni, consulta [Rimuovi i sorgenti del repository software AWS IoT Greengrass principale](#).

Rimuovi i sorgenti del repository software AWS IoT Greengrass principale

È possibile rimuovere i sorgenti del AWS IoT Greengrass core software repository quando non è più necessario installare o aggiornare il software di AWS IoT Greengrass base dal repository APT. Dopo l'11 febbraio 2022, è necessario rimuovere il repository dall'elenco delle fonti per evitare errori durante l'esecuzione. `apt update`

Per rimuovere il repository APT dall'elenco delle fonti

- Eseguite i seguenti comandi per rimuovere il repository del software AWS IoT Greengrass principale dall'elenco dei sorgenti.

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

Esegui AWS IoT Greengrass in un contenitore Docker

AWS IoT Greengrass fornisce un Dockerfile e immagini Docker che semplificano l'esecuzione del software AWS IoT Greengrass Core in un contenitore Docker. Per ulteriori informazioni, consulta [the section called “AWS IoT Greengrass Software Docker”](#).

Note

È inoltre possibile eseguire un'applicazione Docker su un dispositivo principale Greengrass. Aggiunto il supporto per il [connettore di distribuzione dell'applicazione Greengrass Docker](#).

Esegui in un attimo AWS IoT Greengrass

AWS IoT Greengrass snap 1.11.x consente di eseguire una versione limitata di AWS IoT Greengrass tramite comodi pacchetti software, insieme a tutte le dipendenze necessarie, in un ambiente containerizzato.

[Il 31 dicembre 2023, AWS IoT Greengrass terminerà la manutenzione per la versione AWS IoT Greengrass principale del software 1.11.x Snap pubblicata su \[snapcraft.io\]\(#\)](#). I dispositivi che attualmente eseguono Snap continueranno a funzionare fino a nuovo avviso. Tuttavia, lo Snap AWS IoT Greengrass principale non riceverà più patch di sicurezza o correzioni di bug al termine della manutenzione.

Concetti relativi a

Di seguito sono riportati i concetti essenziali di snap per aiutarti a capire come utilizzare lo AWS IoT Greengrass snap:

[Canale](#)

Un componente snap che definisce quale versione di uno snap è installata e monitorata per gli aggiornamenti. Gli snap vengono aggiornati automaticamente alla versione più recente del canale corrente.

[Interfaccia](#)

Un componente snap che consente l'accesso a risorse, come reti e file utente.

Per eseguire lo AWS IoT Greengrass snap, è necessario collegare le seguenti interfacce. Nota che greengrass-support-no-container deve essere prima connessa e mai disconnessa.

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

Le altre interfacce sono opzionali. Se le funzioni Lambda richiedono l'accesso a risorse specifiche, potrebbe essere necessario connettersi alle interfacce appropriate.

Aggiorna

Gli snap vengono aggiornati automaticamente. Il snapd demone è il gestore di pacchetti snap che verifica la presenza di aggiornamenti quattro volte al giorno per impostazione predefinita. Ogni controllo di aggiornamento è chiamato aggiornamento. Quando si verifica un aggiornamento, il demone si arresta, lo snap viene aggiornato e quindi il demone si riavvia.

[Per ulteriori informazioni, consulta il sito Web di Snapcraft.](#)

Cosa c'è di nuovo con AWS IoT Greengrass snap v1.11.x

Di seguito vengono descritte le novità e le modifiche apportate alla versione 1.11.x dello snap. AWS IoT Greengrass

- Questa versione supporta solo l'snap_daemonutente, esposto come ID utente (UID) e gruppo (GID). 584788
- Questa versione supporta solo funzioni Lambda non containerizzate.

Important

Poiché le funzioni Lambda non containerizzate devono condividere lo stesso utente snap_daemon (), le funzioni Lambda non sono isolate l'una dall'altra. Per ulteriori

informazioni, vedere [Controllo dell'esecuzione delle funzioni di Greengrass Lambda utilizzando](#) la configurazione specifica del gruppo.

- Questa versione supporta i runtime C, C++, Java 8, Node.js 12.x, Python 2.7, Python 3.7 e Python 3.8.

Note

Per evitare runtime Python ridondanti, le funzioni Lambda di Python 3.7 eseguono effettivamente il runtime di Python 3.8.

AWS IoT Greengrass Iniziare con snap

La seguente procedura ti aiuta a installare e configurare lo AWS IoT Greengrass snap sul tuo dispositivo.

Requisiti

Per eseguire lo AWS IoT Greengrass snap, devi fare quanto segue:

- Esegui lo AWS IoT Greengrass snap su una distribuzione Linux supportata, come Ubuntu, Linux Mint, Debian e Fedora.
- Installa il snapd demone sul tuo dispositivo. Il snapd demone che include snap lo strumento gestisce l'ambiente snap sul tuo dispositivo.

Per l'elenco delle distribuzioni Linux supportate e le istruzioni di installazione, consulta [Installazione di snapd](#) nella documentazione di Snap.

Installa e configura lo snap AWS IoT Greengrass

Il seguente tutorial mostra come installare e configurare lo AWS IoT Greengrass snap sul tuo dispositivo.

Note

- Sebbene questo tutorial utilizzi un' EC2 istanza Amazon (x86 t2.micro Ubuntu 20.04), puoi eseguire lo AWS IoT Greengrass snap con hardware fisico, come un Raspberry Pi.

- Il snapd demone è preinstallato su Ubuntu.

1. Installa lo `core18` snap eseguendo il seguente comando nel terminale del tuo dispositivo:

```
sudo snap install core18
```

Lo `core18` snap è uno [snap di base](#) che fornisce un ambiente di runtime con librerie di uso comune. Questo snap è stato creato da [Ubuntu 18.04 LTS](#).

2. Effettua l'aggiornamento `snapd` eseguendo il seguente comando:

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

3. Esegui il `snap list` comando per verificare se hai installato lo AWS IoT Greengrass snap.

Il seguente esempio di risposta mostra che `snapd` è installato, ma `aws-iot-greengrass` non lo è.

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

4. Scegli una delle seguenti opzioni per installare AWS IoT Greengrass snap 1.11.x.

- Per installare lo AWS IoT Greengrass snap, esegui il seguente comando:

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

Risposta di esempio:

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

- Per migrare da una versione precedente alla v1.11.x o eseguire l'aggiornamento all'ultima versione di patch disponibile, esegui il comando seguente:

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

Come altri snap, lo AWS IoT Greengrass snap utilizza i canali per gestire le versioni minori. Gli snap vengono aggiornati automaticamente all'ultima versione disponibile del canale corrente. Ad esempio, se lo specifichi `--channel=1.11.x`, lo AWS IoT Greengrass snap viene aggiornato alla v1.11.5.

È possibile eseguire il `snap info aws-iot-greengrass` comando per ottenere l'elenco dei canali disponibili per. AWS IoT Greengrass

Risposta di esempio:

```
name:      aws-iot-greengrass
summary:   AWS supported software that extends cloud capabilities to local devices.
publisher: Amazon Web Services (aws#)
store-url: https://snapcraft.io/aws-iot-greengrass
contact:   https://repost.aws/tags/TA4ckIed1sR4enZBey29rKTg/aws-io-t-greengrass
license:   Proprietary
description: |
  AWS IoT Greengrass seamlessly extends AWS onto edge devices so they can act
  locally on the data
  they generate, while still using the cloud for management, analytics, and durable
  storage.
  AWS IoT Greengrass snap v1.11.0 enables you to run a limited version of AWS IoT
  Greengrass with
  all necessary dependencies in a containerized environment.
  The AWS IoT Greengrass snap doesn't support connectors and machine learning (ML)
  inference.
  By downloading this software you agree to the Greengrass Core Software License
  Agreement
  (https://s3-us-west-2.amazonaws.com/greengrass-release-license/greengrass-
  license-v1.pdf).
  For more information, see Run AWS IoT Greengrass in a snap
  (https://docs.aws.amazon.com/greengrass/latest/developerguide/install-
  ggc.html#gg-snap-support) in
  the AWS IoT Greengrass Developer.
  If you need help, try the AWS IoT Greengrass tag on AWS re:Post
  (https://repost.aws/tags/TA4ckIed1sR4enZBey29rKTg/aws-io-t-greengrass) or connect
  with an AWS IQ expert
  (https://iq.aws.amazon.com/services/aws/greengrass).
snap-id:   SRDuhPJGj4XPxFNNZQKOTvURAp0wxKnd
channels:
  latest/stable: 1.11.3 2021-06-15 (59) 111MB -
```



```
latest/candidate: 1.11.3 2021-06-14 (59) 111MB -
latest/beta:      1.11.3 2021-06-14 (59) 111MB -
latest/edge:      1.11.3 2021-06-14 (59) 111MB -
1.11.x/stable:    1.11.3 2021-06-15 (59) 111MB -
1.11.x/candidate: 1.11.3 2021-06-15 (59) 111MB -
1.11.x/beta:      1.11.3 2021-06-15 (59) 111MB -
1.11.x/edge:      1.11.3 2021-06-15 (59) 111MB -
```

5. Per accedere a risorse specifiche di cui le tue funzioni Lambda hanno bisogno, puoi connetterti a interfacce aggiuntive.

Esegui il comando seguente per ottenere l'elenco delle interfacce supportate da AWS IoT Greengrass snap:

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

Risposta di esempio:

Interface	Notes	Plug	Slot
camera	-	aws-iot-greengrass:camera	-
dvb	-	aws-iot-greengrass:dvb	-
gpio	-	aws-iot-greengrass:gpio	-
gpio-memory-control	-	aws-iot-greengrass:gpio-memory-control	-
greengrass-support	-	aws-iot-greengrass:greengrass-support-no-container	-
:greengrass-support	-	aws-iot-greengrass:greengrass-support-no-container	-
hardware-observe	-	aws-iot-greengrass:hardware-observe	-
:hardware-observe	manual	aws-iot-greengrass:hardware-observe	-
hardware-random-control	-	aws-iot-greengrass:hardware-random-control	-
home	-	aws-iot-greengrass:home-for-greengrassd	-
home	-	aws-iot-greengrass:home-for-hooks	:home
hugepages-control	manual	aws-iot-greengrass:hugepages-control	-
:hugepages-control	manual	aws-iot-greengrass:hugepages-control	-
i2c	-	aws-iot-greengrass:i2c	-

iio		aws-iot-greengrass:iio	-
-			
joystick		aws-iot-greengrass:joystick	-
-			
log-observe		aws-iot-greengrass:log-observe	:log-
observe	manual		
mount-observe		aws-iot-greengrass:mount-observe	
:mount-observe	manual		
network		aws-iot-greengrass:network	
:network	-		
network-bind		aws-iot-greengrass:network-bind	
:network-bind	-		
network-control		aws-iot-greengrass:network-control	
:network-control	-		
opengl		aws-iot-greengrass:opengl	
:opengl	-		
optical-drive		aws-iot-greengrass:optical-drive	
:optical-drive	-		
process-control		aws-iot-greengrass:process-control	
:process-control	-		
raw-usb		aws-iot-greengrass:raw-usb	-
-			
removable-media		aws-iot-greengrass:removable-media	-
-			
serial-port		aws-iot-greengrass:serial-port	-
-			
spi		aws-iot-greengrass:spi	-
-			
system-observe		aws-iot-greengrass:system-observe	
:system-observe	-		

Se vedi un trattino (-) nella colonna Slot, l'interfaccia corrispondente non è connessa.

6. Segui [Installazione del software AWS IoT Greengrass Core](#) per creare un AWS IoT oggetto, un gruppo Greengrass, risorse di sicurezza che consentono comunicazioni sicure con AWS IoT e il file di configurazione del software AWS IoT Greengrass Core. Il file di configurazione contiene configurazioni specifiche del core di Greengrass, come la posizione dei file dei certificati e l'endpoint dei dati del AWS IoT dispositivo. `config.json`

 Note

Se hai scaricato il file su un altro dispositivo, segui questo [passaggio](#) per trasferire i file sul dispositivo AWS IoT Greengrass principale.

7. Per prima AWS IoT Greengrass cosa, assicurati di aggiornare il file [config.json](#), come illustrato di seguito:
 - Sostituisci ogni istanza di *certificateId* con l'ID del certificato nel nome del certificato e dei file chiave.
 - Se hai scaricato un certificato Amazon Root CA diverso da Amazon Root CA 1, sostituisci ogni istanza di *AmazonRootCA1.pem* con il nome del file Amazon root CA.

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

8. Esegui il comando seguente per aggiungere il AWS IoT Greengrass certificato e i file di configurazione:

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

Implementazione di una funzione Lambda

Questa sezione mostra come implementare una funzione Lambda gestita dal cliente sullo AWS IoT Greengrass snap.

Important

AWS IoT Greengrass snap v1.11 supporta solo funzioni Lambda non containerizzate.

1. Esegui il comando seguente per avviare il demone: AWS IoT Greengrass

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

Risposta di esempio:

```
Started.
```

Note

Se si verifica un errore, è possibile utilizzare il `snap run` comando per un messaggio di errore dettagliato. Per ulteriori informazioni sulla risoluzione dei problemi, vedere [errore: impossibile eseguire le seguenti attività: - Esegui il comando di servizio «start» per i servizi \["greengrassd"\] di snap "" \(\[start snap. aws-iot-greengrass aws-iot-greengrass.greengrassd.service\] non riuscito con exit status 1: Job for snap. aws-iot-greengrass.greengrassd.service non è riuscito perché il processo di controllo è terminato con un codice di errore. Vedi «systemctl status snap». aws-iot-greengrass.greengrassd.service» e «journalctl -xe» per i dettagli.\)](#).

2. Esegui il comando seguente per confermare che il demone è in esecuzione:

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

Risposta di esempio:

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

3. Segui il [Modulo 3 \(parte 1\): Lambda funziona AWS IoT Greengrass](#) per creare e distribuire una funzione Hello World Lambda. Tuttavia, prima di distribuire la funzione Lambda, completa il passaggio successivo.
4. Assicurati che la tua funzione Lambda venga eseguita come `snap_daemon` utente e in modalità senza contenitore. Per aggiornare le impostazioni del tuo gruppo Greengrass, procedi come segue nella AWS IoT Greengrass console:
 - a. Accedi alla AWS IoT Greengrass console.
 - b. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).
 - c. In Gruppi Greengrass, scegli il gruppo target.
 - d. Nella pagina di configurazione del gruppo, nel riquadro di navigazione, scegli la scheda Funzioni Lambda.
 - e. In Ambiente di runtime della funzione Lambda predefinito, scegliete Modifica ed effettuate le seguenti operazioni:
 - i. Per Utente e gruppo di sistema predefiniti, scegliete Un altro ID utente/ID di gruppo, quindi immettete sia **584788** l'ID utente di sistema (numero) che l'ID del gruppo di sistema (numero).
 - ii. Per la containerizzazione predefinita della funzione Lambda, scegli Nessun contenitore.
 - iii. Seleziona Salva.

Arresto del demone AWS IoT Greengrass

È possibile utilizzare il `snap stop` comando per interrompere un servizio.

Per fermare il AWS IoT Greengrass demone, esegui il seguente comando:

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

Il comando dovrebbe tornare. Stopped .

Per verificare se lo snap è stato interrotto correttamente, esegui il seguente comando:

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

Risposta di esempio:

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

Disinstallazione dello snap AWS IoT Greengrass

Per disinstallare lo AWS IoT Greengrass snap, esegui il seguente comando:

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

Risposta di esempio:

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

Risoluzione dei problemi relativi allo AWS IoT Greengrass snap

Utilizza le seguenti informazioni per risolvere i problemi relativi allo AWS IoT Greengrass snap.

Errori relativi all'autorizzazione ottenuta e negata.

Soluzione: gli errori di autorizzazione negata sono spesso dovuti alla mancanza di interfacce. Per l'elenco delle interfacce mancanti e informazioni dettagliate sulla risoluzione dei problemi, puoi utilizzare lo `snappy-debug` strumento.

Esegui il comando seguente per installare lo strumento.

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

Risposta di esempio:

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

Esegui il `sudo snappy-debug` comando in una sessione terminale separata. L'operazione continua fino a quando non si verifica un errore di autorizzazione negata.

Ad esempio, se la funzione Lambda tenta di leggere un file nella `$HOME` directory, potresti ottenere la seguente risposta:

```
INFO: Following '/var/log/syslog'. If have dropped messages, use:  
INFO: $ sudo journalctl --output=short --follow --all | sudo snappy-debug  
kernel.printk_ratelimit = 0
```

```
= AppArmor =  
Time: Dec 6 04:48:26  
Log: apparmor="DENIED" operation="mknod" profile="snap.aws-iot-greengrass.greengrassd"  
name="/home/ubuntu/my-file.txt" pid=12345 comm="touch" requested_mask="c"  
denied_mask="c" fsuid=0 ouid=0  
File: /home/ubuntu/my-file.txt (write)  
Suggestion:  
* add 'home' to 'plugins'
```

Questo esempio mostra che la creazione del `/home/ubuntu/my-file.txt` file ha causato l'errore di autorizzazione. Suggerisce inoltre di aggiungere `home` a `plugins`. Tuttavia, questo suggerimento non è applicabile. Agli `home-for-greengrassd` and `home-for-hooks` plug viene concesso solo l'accesso in sola lettura.

Per maggiori informazioni, consulta [The snappy-debug snap nella documentazione](#) di Snap.

errore: impossibile eseguire le seguenti attività: - Esegui il comando di servizio «start» per i servizi ["greengrassd"] di snap "" ([start snap. aws-iot-greengrass aws-iot-greengrass.greengrassd.service] non riuscito con exit status 1: Job for snap. aws-iot-greengrass.greengrassd.service non è riuscito perché il processo di controllo è terminato con un codice di errore. Vedi «systemctl status snap. aws-iot-greengrass.greengrassd.service» e «journalctl -xe» per i dettagli.)

Soluzione: questo errore potrebbe verificarsi quando il `snap start aws-iot-greengrass` comando non riesce ad avviare il software Core. AWS IoT Greengrass

Per ulteriori informazioni sulla risoluzione dei problemi, esegui il comando seguente:

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

Risposta di esempio:

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

Questo esempio mostra che AWS IoT Greengrass non è stato possibile trovare il `config.json` file. È possibile controllare i file di configurazione e di certificato.

`/non var/snap/aws-iot-greengrass/current/ggc-write-directory/packages/1.11.5/rootfs/merged` è un percorso assoluto o è un collegamento simbolico.

Soluzione: lo AWS IoT Greengrass snap supporta solo funzioni Lambda non containerizzate.

Assicurati di eseguire le funzioni Lambda in modalità `no container`. Per ulteriori informazioni, consulta

[Considerazioni sulla scelta della containerizzazione delle funzioni Lambda nella Guida per gli sviluppatori.](#) AWS IoT Greengrass Version 1

Il demone snapd non è riuscito a riavviarsi dopo aver eseguito il comando `sudo snap refresh snapd`.

Soluzione: segui i passaggi da 6 a 8 [Installa e configura lo snap AWS IoT Greengrass](#) per aggiungere il AWS IoT Greengrass certificato e i file di configurazione allo snap. AWS IoT Greengrass

Archivia e installazione del software AWS IoT Greengrass Core

Quando esegui l'aggiornamento a una nuova versione del software AWS IoT Greengrass Core, puoi archiviare la versione attualmente installata. In questo modo, l'ambiente di installazione corrente resta a disposizione e puoi testare una nuova versione del software sullo stesso hardware. Questo semplifica inoltre l'esecuzione del rollback alla versione archiviata per qualsiasi motivo.

Per archiviare l'installazione corrente e installare una nuova versione

1. Scaricare il pacchetto di installazione del [software AWS IoT Greengrass Core](#) che si desidera utilizzare per l'aggiornamento.
2. Copiare il pacchetto nel dispositivo core di destinazione. Per istruzioni che mostrano come trasferire i file, consulta questa [fase](#).

Note

I certificati, le chiavi e il file di configurazione correnti verranno copiati nella nuova installazione in seguito.

Eseguire i comandi nelle fasi seguenti nel terminale del dispositivo core:

3. Assicurarsi che il daemon Greengrass sia stato arrestato sul dispositivo core.
 - a. Per controllare se il daemon è in esecuzione:

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

Se l'output contiene una voce `root` per `/greengrass/ggc/packages/ggc-version/bin/daemon`, allora il daemon è in esecuzione.

 Note

Questa procedura è stata scritta presupponendo che il software AWS IoT Greengrass Core sia installato nella `/greengrass` directory.

- b. Per arrestare il daemon:

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

4. Sposta la directory principale di Greengrass in una directory diversa.

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

5. Decomprimi il nuovo software sul dispositivo core. Sostituite i *version* segnaposto *os-architecture* e nel comando.

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

6. Copia i certificati, le chiavi e il file di configurazione archiviati nella nuova installazione.

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

7. Avvia il daemon:

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

Ora è possibile effettuare una distribuzione del gruppo per testare la nuova installazione. Se si verifica un errore, è possibile ripristinare l'installazione archiviata.

Per ripristinare l'installazione archiviata

1. Arresta il daemon.
2. Elimina la nuova directory `/greengrass`.
3. Riporta la directory `/greengrass_backup` in `/greengrass`.
4. Avvia il daemon.

Configura il AWS IoT Greengrass core

Un AWS IoT Greengrass core è un AWS IoT oggetto (dispositivo) che funge da hub o gateway in ambienti edge. Come gli altri AWS IoT dispositivi, nel registro esiste un core, dispone di un'ombra del dispositivo e utilizza un certificato del dispositivo per l'autenticazione con AWS IoT Core e AWS IoT Greengrass. Il dispositivo core esegue il software AWS IoT Greengrass Core, che consente di gestire i processi locali per i gruppi Greengrass, come la comunicazione, la sincronizzazione shadow e lo scambio di token.

Il software AWS IoT Greengrass Core offre le seguenti funzionalità:

- Distribuzione ed esecuzione locale di connettori e funzioni Lambda.
- Elabora i flussi di dati localmente con esportazioni automatiche verso Cloud AWS
- Messaggistica MQTT sulla rete locale tra dispositivi, connettori e funzioni Lambda utilizzando abbonamenti gestiti.
- Messaggistica MQTT tra AWS IoT dispositivi, connettori e funzioni Lambda utilizzando abbonamenti gestiti.
- Connessioni sicure tra dispositivi e Cloud AWS utilizzo dell'autenticazione e dell'autorizzazione dei dispositivi.
- Sincronizzazione shadow locale dei dispositivi. Le ombre possono essere configurate per la sincronizzazione con Cloud AWS
- Accesso controllato alle risorse volume e dispositivo locali.
- Distribuzione di modelli di machine learning formati nel cloud per l'esecuzione di un'inferenza locale.
- Rilevamento automatico dell'indirizzo IP che permette ai dispositivi di scoprire il dispositivo Greengrass core.
- Distribuzione centralizzata di una configurazione del gruppo, nuova o aggiornata. Dopo il download dei dati di configurazione, il dispositivo core viene riavviato automaticamente.
- Aggiornamenti software sicuri over-the-air (OTA) delle funzioni Lambda definite dall'utente.
- Archiviazione sicura e crittografata dei segreti locali e accesso controllato tramite connettori e funzioni Lambda.

AWS IoT Greengrass file di configurazione principale

Il file di configurazione per il software AWS IoT Greengrass Core è `config.json`. Si trova nella directory `/greengrass-root/config`.

Note

`greengrass-root` rappresenta il percorso in cui il software AWS IoT Greengrass Core è installato sul dispositivo. In genere, questa è la directory `/greengrass`.

Se si utilizza l'opzione di creazione del gruppo predefinito dalla AWS IoT Greengrass console, il `config.json` file viene distribuito sul dispositivo principale in uno stato funzionante.

È possibile rivedere i contenuti di questo file eseguendo il seguente comando:

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

Di seguito è riportato un esempio del file `config.json`. Questa è la versione generata quando crei il core dalla AWS IoT Greengrass console.

GGC v1.11

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

```
{
  "cgroup": {
    "useSystemd": "yes"
  },
  "managedRespawn": false,
  "crypto": {
    "principals": {
      "SecretsManager": {
        "privateKeyPath": "file:///greengrass/certs/hash.private.key"
      },
      "IoTCertificate": {
        "privateKeyPath": "file:///greengrass/certs/hash.private.key",
        "certificatePath": "file:///greengrass/certs/hash.cert.pem"
      }
    },
    "caPath": "file:///greengrass/certs/root.ca.pem"
  },
  "writeDirectory": "/var/snap/aws-iot-greengrass/current/ggc-write-directory",
  "pidFileDirectory": "/var/snap/aws-iot-greengrass/current/pidFileDirectory"
}
```

Il file `config.json` supporta le seguenti proprietà:

coreThing

Campo	Descrizione	Note
caPath	Il percorso della CA AWS IoT principale relativo alla <i>/greengrass-root /</i> certs directory.	Per la compatibilità con le versioni precedenti alla 1.7.0. Questa proprietà viene ignorata quando l'oggetto è presente. crypto

 **Note**
Assicurati che gli [endpoint corrispondano al tipo di certificato](#).

Campo	Descrizione	Note
certPath	Il percorso del certificato del dispositivo core relativo alla directory <i>/greengrass-root/certs</i> .	Per la retrocompatibilità con le versioni precedenti alla 1.7.0. Questa proprietà viene ignorata quando l'oggetto è presente. <code>crypto</code>
keyPath	Il percorso della chiave privata core relativa alla directory <i>/greengrass-root/certs</i> .	Per la compatibilità con le versioni precedenti alla 1.7.0. Questa proprietà viene ignorata quando l'oggetto è presente. <code>crypto</code>
thingArn	L'Amazon Resource Name (ARN) dell' AWS IoT elemento che rappresenta il dispositivo AWS IoT Greengrass principale.	Trova l'ARN per il tuo core nella AWS IoT Greengrass console sotto Cores o eseguendo il comando aws greengrass get-core-definition-version _CLI .

Campo	Descrizione	Note
iotHost	Il tuo endpoint. AWS IoT	Trova l'endpoint nella AWS IoT console in Impostazioni o eseguendo il comando aws iot describe-endpoint --endpoint-type iot:Data-ATS CLI. Questo comando restituisce l'endpoint Amazon Trust Services (ATS). Per ulteriori informazioni, consulta la documentazione Autenticazione del server.  Note Assicurati che gli endpoint corrispon dano al tipo di certificato. Assicurati che i tuoi endpoint corrispon dano ai tuoi. Regione AWS

Campo	Descrizione	Note
ggHost	Il tuo AWS IoT Greengrass endpoint.	Questo è l'endpoint <code>iotHost</code> con il prefisso <code>host</code> sostituito da <code>greengrass</code> (ad esempio, <code>greengrass-ats.iot.region.amazonaws.com</code>). Usa lo stesso Regione AWS di <code>iotHost</code>.  Note Assicurati che gli endpoint corrispondano al tipo di certificato. Assicurati che i tuoi endpoint corrispondano ai tuoi Regione AWS
iotMqttPort	Facoltativo. Il numero di porta da utilizzare per la comunicazione MQTT con AWS IoT.	I valori validi sono 8883 e 443. Il valore predefinito è 8883. Per ulteriori informazioni, consulta Connessione alla porta 443 o tramite un proxy di rete .
iotHttpPort	Facoltativo. Il numero di porta utilizzato per creare le connessioni HTTPS a AWS IoT.	I valori validi sono 8443 e 443. Il valore predefinito è 8443. Per ulteriori informazioni, consulta Connessione alla porta 443 o tramite un proxy di rete .

Campo	Descrizione	Note
ggMqttPort	Facoltativo. Il numero di porta da utilizzare per la comunicazione MQTT sulla rete locale.	I valori validi sono compresi tra 1024 e 65535. Il valore predefinito è 8883. Per ulteriori informazioni, consulta the section called “Porta MQTT per la messaggistica locale” .
ggHttpPort	Facoltativo. Il numero di porta utilizzato per creare le connessioni HTTPS al servizio AWS IoT Greengrass.	I valori validi sono 8443 e 443. Il valore predefinito è 8443. Per ulteriori informazioni, consulta Connessione alla porta 443 o tramite un proxy di rete .
keepAlive	Facoltativo. Il periodo MQTT KeepAlive in secondi.	L'intervallo valido è 30-1200 secondi. Il valore predefinito è 600.
networkProxy	Facoltativo. Un oggetto che definisce un server proxy a cui connettersi.	Il server proxy può essere HTTP o HTTPS. Per ulteriori informazioni, consulta Connessione alla porta 443 o tramite un proxy di rete .
mqttOperationTimeout	Facoltativo. La quantità di tempo (in secondi) per consentire al core Greengrass di completare un'operazione di pubblicazione, sottoscrizione o annullamento dell'iscrizione nelle connessioni MQTT a AWS IoT Core.	Il valore predefinito è 5. Il valore minimo è 5.

Campo	Descrizione	Note
<code>ggDaemonPort</code>	Facoltativo. Il numero di porta IPC principale di Greengrass.	Questa proprietà è disponibile nella versione AWS IoT Greengrass 1.11.0 o successiva. I valori validi sono compresi tra 1024 e 65535. Il valore predefinito è 8000.
<code>systemComponentAuthTimeout</code>	Facoltativo. Il tempo (in millisecondi) necessario per consentire all'IPC principale di Greengrass di completare l'autenticazione.	Questa proprietà è disponibile nella versione 1.11.0 o successiva. AWS IoT Greengrass I valori validi sono compresi tra 500 e 5000. Il valore predefinito è 5000.

runtime

Campo	Descrizione	Note
<code>maxWorkItemCount</code>	Facoltativo. Il numero massimo di elementi di lavoro che il daemon Greengrass può elaborare alla volta. Gli elementi di lavoro che superano questo limite vengono ignorati. La coda degli elementi di lavoro è condivisa dai componenti di sistema, dalle funzioni Lambda definite dall'utente e dai connettori.	Il valore predefinito è 1024. Il valore massimo è limitato dall'hardware del dispositivo. L'aumento di questo valore aumenta la memoria utilizzata. AWS IoT Greengrass È possibile aumentare questo valore se si prevede che il core riceverà un intenso traffico di messaggi MQTT.

Campo	Descrizione	Note
<code>maxConcurrentLimit</code>	Facoltativo. Il numero massimo di worker Lambda non bloccati simultaneamente che il demone Greengrass può avere. È possibile specificare un numero intero diverso per sovrascrivere questo parametro.	Il valore predefinito è 25. Il valore minimo è definito da <code>lruSize</code> .
<code>lruSize</code>	Facoltativo. Definisce il valore minimo per <code>maxConcurrentLimit</code> .	Il valore predefinito è 25.
<code>mountAllBlockDevices</code>	Facoltativo. Consente di AWS IoT Greengrass utilizzare i bind mount per montare tutti i dispositivi a blocchi in un contenitore dopo aver configurato OverlayFS.	Questa proprietà è disponibile nella AWS IoT Greengrass versione 1.11.0 o successiva. I valori validi sono yes e no. Il valore predefinito è no. Imposta questo valore su yes se la tua <code>/usr</code> directory non è sotto la gerarchia. /
<code>postStartHealthCheckTimeout</code>	Facoltativo. Il tempo (in millisecondi) dopo l'avvio che il daemon Greengrass attende per il completamento del controllo dello stato.	Il timeout predefinito è 30 secondi (30000 ms).
<code>cgroup</code>		

Campo	Descrizione	Note
useSystemd	Indica se il dispositivo utilizza systemd .	I valori validi sono yes e no. Esegui lo script <code>check_ggc_dependencies</code> nel Modulo 1 per visualizzare se il dispositivo usa systemd.

crypto

crypto contiene proprietà che supportano lo storage di chiavi private in un modulo di sicurezza hardware (HSM) tramite PKCS#11 e storage segreti locali. Per ulteriori informazioni, consulta [the section called “Principal di sicurezza”](#), [the section called “Integrazione della sicurezza hardware”](#) e [Distribuzione dei segreti nel core](#). Sono supportate le configurazioni per l'archiviazione delle chiavi private su HSMs o nel file system.

Campo	Descrizione	Note
caPath	Il percorso assoluto verso la CA AWS IoT principale.	Deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code> .

Note

Assicurati che gli [endpoint corrispon](#)
[dano al tipo di](#)
[certificato](#).

PKCS11

OpenSSLEngine	Facoltativo. Il percorso assoluto del file .so del motore OpenSSL per abilitare il supporto PKCS#11 su OpenSSL.	Deve essere un percorso di un file nel file system. Questa proprietà è obbligato ria se si utilizza l'agente

Campo	Descrizione	Note
		di aggiornamento OTA Greengrass con sicurezza hardware. Per ulteriori informazioni, consulta the section called “Configurazione degli aggiornamenti OTA” .
P11Provider	Il percorso assoluto della libreria libdl-loadable dell'implementazione PKCS#11.	Deve essere un percorso di un file nel file system.
slotLabel	L'etichetta dello slot utilizzata per identificare il modulo hardware.	Deve essere conforme alle specifiche dell'etichetta PKCS#11.
slotUserPin	Il PIN utente utilizzato per autenticare il core Greengrass nel modulo.	Deve disporre delle autorizzazioni sufficienti a eseguire C_Sign con le chiavi private configurate.
principals		
IoTCertificate	Il certificato e la chiave privata che il core utilizza per effettuare le richieste a AWS IoT.	
IoTCertificate.privateKeyPath	Il percorso della chiave privata del core.	Per lo storage di file system, deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code>. Per lo storage HSM, deve essere un percorso PKCS#11 RFC 7512 che specifica l'etichetta dell'oggetto.

Campo	Descrizione	Note
IoTCertificate .certificatePath	Il percorso assoluto al certificato del dispositivo core.	Deve essere un URI di file nel formato: <i>file:///absolute/path/to/file</i> .
MQTTServerCertificate	Facoltativo. La chiave privata che il core utilizza insieme al certificato per fungere da gateway o server MQTT.	
MQTTServerCertificate .privateKeyPath	Il percorso della chiave privata del server MQTT locale.	Utilizza questo valore per specificare la tua chiave privata del server MQTT locale. Per lo storage di file system, deve essere un URI di file nel formato: <i>file:///absolute/path/to/file</i> . Per lo storage HSM, deve essere un percorso PKCS#11 RFC 7512 che specifica l'etichetta dell'oggetto. Se questa proprietà viene omessa, AWS IoT Greengrass ruota la chiave in base alle impostazioni di rotazione. Se viene specificata, il cliente sarà responsabile della rotazione della chiave.
SecretsManager	La chiave privata che protegge la chiave di dati utilizzata per la crittografia. Per ulteriori informazioni, consulta Distribuzione dei segreti nel core .	

Campo	Descrizione	Note
SecretsManager.privateKeyPath	Il percorso della chiave privata del Secrets Manager locale.	Solo una chiave RSA è supportata. Per lo storage di file system, deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code>. Per lo storage HSM, deve essere un percorso PKCS#11 RFC 7512 che specifica l'etichetta dell'oggetto. La chiave privata deve essere generata utilizzando il meccanismo di padding PKCS#1 v1.5.

Sono supportate anche le seguenti proprietà di configurazione:

Campo	Descrizione	Note
mqttMaxConnectionRetryInterval	Facoltativo. L'intervallo massimo (in secondi) tra tentativi di connessione MQTT se la connessione viene interrotta.	Specifica questo valore come numero intero senza segno. Il valore predefinito è 60.
managedRespawn	Facoltativo. Indica che l'agente OTA deve eseguire il codice personalizzato prima di un aggiornamento.	I valori validi sono <code>true</code> e <code>false</code> . Per ulteriori informazioni, consulta Aggiornamenti OTA del software AWS IoT Greengrass Core .

Campo	Descrizione	Note
writeDirectory	Facoltativo. La directory di scrittura in cui vengono AWS IoT Greengrass create tutte le risorse di lettura/scrittura.	Per ulteriori informazioni, consulta Configura una directory di scrittura per AWS IoT Greengrass .
pidFileDirectory	Facoltativo. AWS IoT Greengrass memorizza il relativo ID di processo (PID) in questa directory.	Il valore predefinito è /var/run.

Extended life versions

Le seguenti versioni del software AWS IoT Greengrass Core sono in [fase di estensione della vita utile](#). Queste informazioni sono incluse solo per riferimento.

GGC v1.10

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

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

Il file `config.json` supporta le seguenti proprietà:

coreThing

Campo	Descrizione	Note
caPath	Il percorso della CA AWS IoT principale relativo alla directory. <code>/greengrass-root/certs</code>	Per la compatibilità con le versioni precedenti alla 1.7.0. Questa proprietà viene ignorata quando l'oggetto è presente. <code>crypto</code>  Note Assicurati che gli endpoint corrispondano al tipo di certificato.
certPath	Il percorso del certificato del dispositivo core relativo alla directory <code>/greengrass-root/certs</code> .	Per la retrocompatibilità con le versioni precedenti alla 1.7.0. Questa proprietà viene ignorata quando l'oggetto è presente. <code>crypto</code>

Campo	Descrizione	Note
keyPath	Il percorso della chiave privata core relativa alla directory <i>/greengrass-root/certs</i> .	Per la compatibilità con le versioni precedenti alla 1.7.0. Questa proprietà viene ignorata quando l'oggetto è presente. <code>crypto</code>
thingArn	L'Amazon Resource Name (ARN) dell' AWS IoT elemento che rappresenta il dispositivo AWS IoT Greengrass principale.	Trova l'ARN per il tuo core nella AWS IoT Greengrass console sotto Cores o eseguendo il comando aws greengrass get-core-definition-version _CLI .

Campo	Descrizione	Note
iotHost	Il tuo endpoint. AWS IoT	Trova l'endpoint nella AWS IoT console in Impostazioni o eseguendo il comando aws iot describe-endpoint --endpoint-type iot:Data-ATS CLI. Questo comando restituisce l'endpoint Amazon Trust Services (ATS). Per ulteriori informazioni, consulta la documentazione Autenticazione del server.  Note Assicurati che gli endpoint corrispondano al tipo di certificato. Assicurati che i tuoi endpoint corrispondano ai tuoi. Regione AWS

Campo	Descrizione	Note
ggHost	Il tuo AWS IoT Greengrass endpoint.	Questo è l'endpoint <code>iotHost</code> con il prefisso <code>host</code> sostituito da <code>greengrass</code> (ad esempio, <code>greengrass-ats.iot</code> . region .amazonaws.com). Usa lo stesso Regione AWS di <code>iotHost</code>  Note Assicurati che gli endpoint corrispon dano al tipo di certificato. Assicurati che i tuoi endpoint corrispondano ai tuoi. Regione AWS
iotMqttPort	Facoltativo. Il numero di porta da utilizzare per la comunicazione MQTT con AWS IoT.	I valori validi sono 8883 e 443. Il valore predefinito è 8883. Per ulteriori informazioni, consulta Connessione alla porta 443 o tramite un proxy di rete .
iotHttpPort	Facoltativo. Il numero di porta utilizzato per creare le connessioni HTTPS a AWS IoT.	I valori validi sono 8443 e 443. Il valore predefinito è 8443. Per ulteriori informazioni, consulta Connessione alla porta 443 o tramite un proxy di rete .

Campo	Descrizione	Note
<code>ggMqttPort</code>	Facoltativo. Il numero di porta da utilizzare per la comunicazione MQTT sulla rete locale.	I valori validi sono compresi tra 1024 e 65535. Il valore predefinito è 8883. Per ulteriori informazioni, consulta the section called “Porta MQTT per la messaggistica locale” .
<code>ggHttpPort</code>	Facoltativo. Il numero di porta utilizzato per creare le connessioni HTTPS al servizio AWS IoT Greengrass.	I valori validi sono 8443 e 443. Il valore predefinito è 8443. Per ulteriori informazioni, consulta Connessione alla porta 443 o tramite un proxy di rete .
<code>keepAlive</code>	Facoltativo. Il periodo MQTT KeepAlive in secondi.	L'intervallo valido è 30-1200 secondi. Il valore predefinito è 600.
<code>networkProxy</code>	Facoltativo. Un oggetto che definisce un server proxy a cui connettersi.	Il server proxy può essere HTTP o HTTPS. Per ulteriori informazioni, consulta Connessione alla porta 443 o tramite un proxy di rete .
<code>mqttOperationTimeout</code>	Facoltativo. La quantità di tempo (in secondi) per consentire al core Greengrass di completare un'operazione di pubblicazione, sottoscrizione o annullamento dell'iscrizione nelle connessioni MQTT a AWS IoT Core.	Questa proprietà è disponibile a partire dalla AWS IoT Greengrass v1.10.2. Il valore predefinito è 5. Il valore minimo è 5.

runtime

Campo	Descrizione	Note
<code>maxWorkItemCount</code>	Facoltativo. Il numero massimo di elementi di lavoro che il daemon Greengrass può elaborare alla volta. Gli elementi di lavoro che superano questo limite vengono ignorati. La coda degli elementi di lavoro è condivisa dai componenti di sistema, dalle funzioni Lambda definite dall'utente e dai connettori.	Il valore predefinito è 1024. Il valore massimo è limitato dall'hardware del dispositivo. L'aumento di questo valore aumenta la memoria utilizzata. AWS IoT Greengrass. È possibile aumentare questo valore se si prevede che il core riceverà un intenso traffico di messaggi MQTT.
<code>maxConcurrentLimit</code>	Facoltativo. Il numero massimo di worker Lambda non bloccati simultaneamente che il demone Greengrass può avere. È possibile specificare un numero intero diverso per sovrascrivere questo parametro.	Il valore predefinito è 25. Il valore minimo è definito da <code>lruSize</code>.
<code>lruSize</code>	Facoltativo. Definisce il valore minimo per <code>maxConcurrentLimit</code>.	Il valore predefinito è 25.
<code>postStartHealthCheckTimeout</code>	Facoltativo. Il tempo (in millisecondi) dopo l'avvio che il daemon Greengrass attende per il completamento del controllo dello stato.	Il timeout predefinito è 30 secondi (30000 ms).

Campo	Descrizione	Note
cgroup		
useSystemd	Indica se il dispositivo utilizza systemd .	I valori validi sono yes e no. Esegui lo script <code>check_ggc_dependencies</code> nel Modulo 1 per visualizzare se il dispositivo usa systemd.
crypto		
crypto contiene proprietà che supportano lo storage di chiavi private in un modulo di sicurezza hardware (HSM) tramite PKCS#11 e storage segreti locali. Per ulteriori informazioni, consulta the section called “Principal di sicurezza”, the section called “Integrazione della sicurezza hardware” e Distribuzione dei segreti nel core. Sono supportate le configurazioni per l'archiviazione delle chiavi private sul HSMs o nel file system.		

Campo	Descrizione	Note
caPath	Il percorso assoluto verso la CA AWS IoT principale.	Deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code> .
PKCS11		
OpenSSL Engine	Facoltativo. Il percorso assoluto del file .so del motore OpenSSL	Deve essere un percorso di un file nel file system.

 Note

Assicurati che gli [endpoint corrispondano al tipo di certificato](#).

Campo	Descrizione	Note
	per abilitare il supporto PKCS#11 su OpenSSL.	Questa proprietà è obbligatoria se si utilizza l'agente di aggiornamento OTA Greengrass con sicurezza hardware. Per ulteriori informazioni, consulta the section called “Configurazione degli aggiornamenti OTA” .
P11Provider	Il percorso assoluto della libreria libdl-loadable dell'implementazione PKCS#11.	Deve essere un percorso di un file nel file system.
slotLabel	L'etichetta dello slot utilizzata per identificare il modulo hardware.	Deve essere conforme alle specifiche dell'etichetta PKCS#11.
slotUserPin	Il PIN utente utilizzato per autenticare il core Greengrass nel modulo.	Deve disporre delle autorizzazioni sufficienti a eseguire C_Sign con le chiavi private configurate.
principals		
IoTCertificate	Il certificato e la chiave privata che il core utilizza per effettuare le richieste a AWS IoT.	

Campo	Descrizione	Note
IoTCertificate .privateKeyPath	Il percorso della chiave privata del core.	Per lo storage di file system, deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code>. Per lo storage HSM, deve essere un percorso PKCS#11 RFC 7512 che specifica l'etichetta dell'oggetto.
IoTCertificate .certificatePath	Il percorso assoluto al certificato del dispositivo core.	Deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code> .
MQTTServerCertificate	Facoltativo. La chiave privata che il core utilizza insieme al certificato per fungere da gateway o server MQTT.	

Campo	Descrizione	Note
MQTTServerCertificatePrivateKeyPath	Il percorso della chiave privata del server MQTT locale.	Utilizza questo valore per specificare la tua chiave privata del server MQTT locale. Per lo storage di file system, deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code>. Per lo storage HSM, deve essere un percorso PKCS#11 RFC 7512 che specifica l'etichetta dell'oggetto. Se questa proprietà viene omessa, AWS IoT Greengrass ruota la chiave in base alle impostazioni di rotazione. Se viene specificata, il cliente sarà responsabile della rotazione della chiave.
SecretsManager	La chiave privata che protegge la chiave di dati utilizzata per la crittografia. Per ulteriori informazioni, consulta Distribuzione dei segreti nel core .	

Campo	Descrizione	Note
SecretsManager.privateKeyPath	Il percorso della chiave privata del Secrets Manager locale.	Solo una chiave RSA è supportata. Per lo storage di file system, deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code>. Per lo storage HSM, deve essere un percorso PKCS#11 RFC 7512 che specifica l'etichetta dell'oggetto. La chiave privata deve essere generata utilizzando il meccanismo di padding PKCS#1 v1.5.

Sono supportate anche le seguenti proprietà di configurazione:

Campo	Descrizione	Note
mqttMaxConnectionRetryInterval	Facoltativo. L'intervallo massimo (in secondi) tra tentativi di connessione MQTT se la connessione viene interrotta.	Specifica questo valore come numero intero senza segno. Il valore predefinito è 60.
managedRespawn	Facoltativo. Indica che l'agente OTA deve eseguire il codice personalizzato prima di un aggiornamento.	I valori validi sono <code>true</code> e <code>false</code> . Per ulteriori informazioni, consulta Aggiornamenti OTA del software AWS IoT Greengrass Core .

Campo	Descrizione	Note
writeDirectory	Facoltativo. La directory di scrittura in cui vengono AWS IoT Greengrass create tutte le risorse di lettura/scrittura.	Per ulteriori informazioni, consulta Configura una directory di scrittura per AWS IoT Greengrass .

GGC v1.9

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

Il file `config.json` supporta le seguenti proprietà:

coreThing

Campo	Descrizione	Note
caPath	Il percorso della CA AWS IoT principale relativo alla directory. <code>/greengrass-root/certs</code>	Per la compatibilità con le versioni precedenti alla 1.7.0. Questa proprietà viene ignorata quando l'oggetto è presente. <code>crypto</code>  Note Assicurati che gli endpoint corrispondano al tipo di certificato.
certPath	Il percorso del certificato del dispositivo core relativo alla directory <code>/greengrass-root/certs</code> .	Per la retrocompatibilità con le versioni precedenti alla 1.7.0. Questa proprietà viene ignorata quando l'oggetto è presente. <code>crypto</code>
keyPath	Il percorso della chiave privata core relativa alla directory <code>/greengrass-root/certs</code> .	Per la compatibilità con le versioni precedenti alla 1.7.0. Questa proprietà viene ignorata quando l'oggetto è presente. <code>crypto</code>
thingArn	L'Amazon Resource Name (ARN) dell' AWS IoT elemento che rappresenta il dispositivo AWS IoT Greengrass principale.	Trova l'ARN per il tuo core nella AWS IoT Greengrass console sotto Cores o eseguendo il comando aws greengrass get-core-

Campo	Descrizione	Note
		definition-version _CLI.
iotHost	Il tuo endpoint. AWS IoT	Trova l'endpoint nella AWS IoT console in Impostazioni o eseguendo il comando aws iot describe-endpoint --endpoint-type iot:Data-ATS CLI. Questo comando restituisce l'endpoint Amazon Trust Services (ATS). Per ulteriori informazioni, consulta la documentazione Autenticazione del server.  Note Assicurati che gli endpoint corrispondano al tipo di certificato. Assicurati che i tuoi endpoint corrispondano ai tuoi. Regione AWS

Campo	Descrizione	Note
ggHost	Il tuo AWS IoT Greengrass endpoint.	Questo è l'endpoint <code>iotHost</code> con il prefisso <code>host</code> sostituito da <code>greengrass</code> (ad esempio, <code>greengrass-ats.iot.region.amazonaws.com</code>). Usa lo stesso Regione AWS di <code>iotHost</code>  Note Assicurati che gli endpoint corrispon dano al tipo di certificato. Assicurati che i tuoi endpoint corrispondano ai tuoi. Regione AWS
iotMqttPort	Facoltativo. Il numero di porta da utilizzare per la comunicazione MQTT con AWS IoT.	I valori validi sono 8883 e 443. Il valore predefinito è 8883. Per ulteriori informazioni, consulta Connessione alla porta 443 o tramite un proxy di rete .
iotHttpPort	Facoltativo. Il numero di porta utilizzato per creare le connessioni HTTPS a AWS IoT.	I valori validi sono 8443 e 443. Il valore predefinito è 8443. Per ulteriori informazioni, consulta Connessione alla porta 443 o tramite un proxy di rete .

Campo	Descrizione	Note
<code>ggHttpPort</code>	Facoltativo. Il numero di porta utilizzato per creare le connessioni HTTPS al servizio AWS IoT Greengrass.	I valori validi sono 8443 e 443. Il valore predefinito è 8443. Per ulteriori informazioni, consulta Connessione alla porta 443 o tramite un proxy di rete .
<code>keepAlive</code>	Facoltativo. Il periodo MQTT KeepAlive in secondi.	L'intervallo valido è 30-1200 secondi. Il valore predefinito è 600.
<code>networkProxy</code>	Facoltativo. Un oggetto che definisce un server proxy a cui connettersi.	Il server proxy può essere HTTP o HTTPS. Per ulteriori informazioni, consulta Connessione alla porta 443 o tramite un proxy di rete .

runtime

Campo	Descrizione	Note
<code>maxConcurrentLimit</code>	Facoltativo. Il numero massimo di worker Lambda non bloccati simultaneamente che il demone Greengrass può avere. È possibile specificare un numero intero diverso per sovrascrivere questo parametro.	Il valore predefinito è 25. Il valore minimo è definito da <code>lruSize</code> .
<code>lruSize</code>	Facoltativo. Definisce il valore minimo	Il valore predefinito è 25.

Campo	Descrizione	Note
	<code>permaxConcurrentLimit</code> .	
<code>postStartHealthCheckTimeout</code>	Facoltativo. Il tempo (in millisecondi) dopo l'avvio che il daemon Greengrass attende per il completamento del controllo dello stato.	Il timeout predefinito è 30 secondi (30000 ms).
<code>cgroup</code>		
<code>useSystemd</code>	Indica se il dispositivo utilizza systemd .	I valori validi sono yes e no. Esegui lo script <code>check_ggc_dependencies</code> nel Modulo 1 per visualizzare se il dispositivo usa systemd.

crypto

L'oggetto `crypto` è aggiunto nella v1.7.0. Introduce proprietà che supportano lo storage di chiavi private in un modulo di sicurezza hardware (HSM) tramite PKCS#11 e storage segreti locali. Per ulteriori informazioni, consulta [the section called “Principal di sicurezza”](#), [the section called “Integrazione della sicurezza hardware”](#) e [Distribuzione dei segreti nel core](#). Sono supportate le configurazioni per l'archiviazione delle chiavi private sul HSMs o nel file system.

Campo	Descrizione	Note
<code>caPath</code>	Il percorso assoluto verso la CA AWS IoT principale.	Deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code> .

 **Note**

Assicurati che gli [endpoint corrispon](#)

Campo	Descrizione	Note
		dano al tipo di certificato.
PKCS11		
OpenSSL Engine	Facoltativo. Il percorso assoluto del file .so del motore OpenSSL per abilitare il supporto PKCS#11 su OpenSSL.	Deve essere un percorso di un file nel file system. Questa proprietà è obbligatoria se si utilizza l'agente di aggiornamento OTA Greengrass con sicurezza hardware. Per ulteriori informazioni, consulta the section called “Configurazione degli aggiornamenti OTA”.
P11Provider	Il percorso assoluto della libreria libdl-loadable dell'implementazione PKCS#11.	Deve essere un percorso di un file nel file system.
slotLabel	L'etichetta dello slot utilizzata per identificare il modulo hardware.	Deve essere conforme alle specifiche dell'etichetta PKCS#11.
slotUserPin	Il PIN utente utilizzato per autenticare il core Greengrass nel modulo.	Deve disporre delle autorizzazioni sufficienti a eseguire C_Sign con le chiavi private configurate.
principals		
IoTCertificate	Il certificato e la chiave privata che il core utilizza per effettuare le richieste a AWS IoT.	

Campo	Descrizione	Note
IoTCertificate .privateKeyPath	Il percorso della chiave privata del core.	Per lo storage di file system, deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code>. Per lo storage HSM, deve essere un percorso PKCS#11 RFC 7512 che specifica l'etichetta dell'oggetto.
IoTCertificate .certificatePath	Il percorso assoluto al certificato del dispositivo core.	Deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code> .
MQTTServerCertificate	Facoltativo. La chiave privata che il core utilizza insieme al certificato per fungere da gateway o server MQTT.	

Campo	Descrizione	Note
MQTTServerCertificatePrivateKeyPath	Il percorso della chiave privata del server MQTT locale.	Utilizza questo valore per specificare la tua chiave privata del server MQTT locale. Per lo storage di file system, deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code>. Per lo storage HSM, deve essere un percorso PKCS#11 RFC 7512 che specifica l'etichetta dell'oggetto. Se questa proprietà viene omessa, AWS IoT Greengrass ruota la chiave in base alle impostazioni di rotazione. Se viene specificata, il cliente sarà responsabile della rotazione della chiave.
SecretsManager	La chiave privata che protegge la chiave di dati utilizzata per la crittografia. Per ulteriori informazioni, consulta Distribuzione dei segreti nel core .	

Campo	Descrizione	Note
SecretsManager.privateKeyPath	Il percorso della chiave privata del Secrets Manager locale.	Solo una chiave RSA è supportata. Per lo storage di file system, deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code>. Per lo storage HSM, deve essere un percorso PKCS#11 RFC 7512 che specifica l'etichetta dell'oggetto. La chiave privata deve essere generata utilizzando il meccanismo di padding PKCS#1 v1.5.

Sono supportate anche le seguenti proprietà di configurazione.

Campo	Descrizione	Note
mqttMaxConnectionRetryInterval	Facoltativo. L'intervallo massimo (in secondi) tra tentativi di connessione MQTT se la connessione viene interrotta.	Specifica questo valore come numero intero senza segno. Il valore predefinito è 60.
managedRespawn	Facoltativo. Indica che l'agente OTA deve eseguire il codice personalizzato prima di un aggiornamento.	I valori validi sono <code>true</code> e <code>false</code> . Per ulteriori informazioni, consulta Aggiornamenti OTA del software AWS IoT Greengrass Core .

Campo	Descrizione	Note
writeDirectory	Facoltativo. La directory di scrittura in cui vengono AWS IoT Greengrass create tutte le risorse di lettura/scrittura.	Per ulteriori informazioni, consulta Configura una directory di scrittura per AWS IoT Greengrass .

GGC v1.8

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

Il `config.json` file supporta le seguenti proprietà.

coreThing

Campo	Descrizione	Note
caPath	Il percorso della CA AWS IoT principale relativo alla <i>/greengrass-root</i> / certs directory.	Per la compatibilità con le versioni precedenti alla 1.7.0. Questa proprietà viene ignorata quando l'oggetto è presente. crypto  Note Assicurati che gli endpoint corrispondano al tipo di certificato.
certPath	Il percorso del certificato del dispositivo core relativo alla directory <i>/greengrass-root</i> /certs.	Per la retrocompatibilità con le versioni precedenti alla 1.7.0. Questa proprietà viene ignorata quando l'oggetto è presente. crypto
keyPath	Il percorso della chiave privata core relativa alla directory <i>/greengrass-root</i> /certs.	Per la compatibilità con le versioni precedenti alla 1.7.0. Questa proprietà viene ignorata quando l'oggetto è presente. crypto
thingArn	L'Amazon Resource Name (ARN) dell' AWS IoT elemento che rappresenta il dispositivo AWS IoT Greengrass principale.	Trova l'ARN per il tuo core nella AWS IoT Greengrass console sotto Cores o eseguendo il comando aws greengrass get-core-

Campo	Descrizione	Note
		definition-version _CLI.
iotHost	Il tuo endpoint. AWS IoT	Trova l'endpoint nella AWS IoT console in Impostazioni o eseguendo il comando aws iot describe-endpoint --endpoint-type iot:Data-ATS CLI. Questo comando restituisce l'endpoint Amazon Trust Services (ATS). Per ulteriori informazioni, consulta la documentazione Autenticazione del server.  Note Assicurati che gli endpoint corrispondano al tipo di certificato. Assicurati che i tuoi endpoint corrispondano ai tuoi. Regione AWS

Campo	Descrizione	Note
ggHost	Il tuo AWS IoT Greengrass endpoint.	Questo è l'endpoint <code>iotHost</code> con il prefisso <code>host</code> sostituito da <code>greengrass</code> (ad esempio, <code>greengrass-ats.iot.region.amazonaws.com</code>). Usa lo stesso Regione AWS di <code>iotHost</code>  Note Assicurati che gli endpoint corrispondano al tipo di certificato. Assicurati che i tuoi endpoint corrispondano ai tuoi. Regione AWS
iotMqttPort	Facoltativo. Il numero di porta da utilizzare per la comunicazione MQTT con AWS IoT	I valori validi sono 8883 e 443. Il valore predefinito è 8883. Per ulteriori informazioni, consulta Connessione alla porta 443 o tramite un proxy di rete .
iotHttpPort	Facoltativo. Il numero di porta utilizzato per creare le connessioni HTTPS a AWS IoT.	I valori validi sono 8443 e 443. Il valore predefinito è 8443. Per ulteriori informazioni, consulta Connessione alla porta 443 o tramite un proxy di rete .

Campo	Descrizione	Note
<code>ggHttpPort</code>	Facoltativo. Il numero di porta utilizzato per creare le connessioni HTTPS al servizio AWS IoT Greengrass.	I valori validi sono 8443 e 443. Il valore predefinito è 8443. Per ulteriori informazioni, consulta Connessione alla porta 443 o tramite un proxy di rete .
<code>keepAlive</code>	Facoltativo. Il periodo MQTT KeepAlive in secondi.	L'intervallo valido è 30-1200 secondi. Il valore predefinito è 600.
<code>networkProxy</code>	Facoltativo. Un oggetto che definisce un server proxy a cui connettersi.	Il server proxy può essere HTTP o HTTPS. Per ulteriori informazioni, consulta Connessione alla porta 443 o tramite un proxy di rete .

runtime

Campo	Descrizione	Note
<code>cgroup</code>		
<code>useSystemd</code>	Indica se il dispositivo utilizza systemd .	I valori validi sono yes e no. Esegui lo script <code>check_ggc_dependencies</code> nel Modulo 1 per visualizzare se il dispositivo usa systemd.

crypto

L'oggetto `crypto` è aggiunto nella v1.7.0. Introduce proprietà che supportano lo storage di chiavi private in un modulo di sicurezza hardware (HSM) tramite PKCS#11 e storage segreti locali. Per ulteriori informazioni, consulta [the section called “Principal di sicurezza”](#), [the section](#)

called [“Integrazione della sicurezza hardware”](#) e [Distribuzione dei segreti nel core](#). Sono supportate le configurazioni per l'archiviazione delle chiavi private su HSMs o nel file system.

Campo	Descrizione	Note
caPath	Il percorso assoluto verso la CA AWS IoT principale.	Deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code> .  Note Assicurati che gli endpoint corrispondano al tipo di certificato.
PKCS11		
OpenSSLEngine	Facoltativo. Il percorso assoluto del file .so del motore OpenSSL per abilitare il supporto PKCS#11 su OpenSSL.	Deve essere un percorso di un file nel file system. Questa proprietà è obbligatoria se si utilizza l'agente di aggiornamento OTA Greengrass con sicurezza hardware. Per ulteriori informazioni, consulta the section called “Configurazione degli aggiornamenti OTA”.
P11Provider	Il percorso assoluto della libreria libdl-loadable dell'implementazione PKCS#11.	Deve essere un percorso di un file nel file system.

Campo	Descrizione	Note
<code>slotLabel</code>	L'etichetta dello slot utilizzato per identificare il modulo hardware.	Deve essere conforme alle specifiche dell'etichetta PKCS#11.
<code>slotUserPin</code>	Il PIN utente utilizzato per autenticare il core Greengrass nel modulo.	Deve disporre delle autorizzazioni sufficienti a eseguire <code>C_Sign</code> con le chiavi private configurate.
<code>principals</code>		
<code>IoTCertificate</code>	Il certificato e la chiave privata che il core utilizza per effettuare le richieste a AWS IoT.	
<code>IoTCertificate.privateKeyPath</code>	Il percorso della chiave privata del core.	Per lo storage di file system, deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code>. Per lo storage HSM, deve essere un percorso PKCS#11 RFC 7512 che specifica l'etichetta dell'oggetto.
<code>IoTCertificate.certificatePath</code>	Il percorso assoluto al certificato del dispositivo core.	Deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code> .
<code>MQTTServerCertificate</code>	Facoltativo. La chiave privata che il core utilizza insieme al certificato per fungere da gateway o server MQTT.	

Campo	Descrizione	Note
MQTTServerCertificatePrivateKeyPath	Il percorso della chiave privata del server MQTT locale.	Utilizza questo valore per specificare la tua chiave privata del server MQTT locale. Per lo storage di file system, deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code>. Per lo storage HSM, deve essere un percorso PKCS#11 RFC 7512 che specifica l'etichetta dell'oggetto. Se questa proprietà viene omessa, AWS IoT Greengrass ruota la chiave in base alle impostazioni di rotazione. Se viene specificata, il cliente sarà responsabile della rotazione della chiave.
SecretsManager	La chiave privata che protegge la chiave di dati utilizzata per la crittografia. Per ulteriori informazioni, consulta Distribuzione dei segreti nel core .	

Campo	Descrizione	Note
SecretsManager.privateKeyPath	Il percorso della chiave privata del Secrets Manager locale.	Solo una chiave RSA è supportata. Per lo storage di file system, deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code>. Per lo storage HSM, deve essere un percorso PKCS#11 RFC 7512 che specifica l'etichetta dell'oggetto. La chiave privata deve essere generata utilizzando il meccanismo di padding PKCS#1 v1.5.

Sono supportate anche le seguenti proprietà di configurazione:

Campo	Descrizione	Note
mqttMaxConnectionRetryInterval	Facoltativo. L'intervallo massimo (in secondi) tra tentativi di connessione MQTT se la connessione viene interrotta.	Specifica questo valore come numero intero senza segno. Il valore predefinito è 60.
managedRespawn	Facoltativo. Indica che l'agente OTA deve eseguire il codice personalizzato prima di un aggiornamento.	I valori validi sono <code>true</code> e <code>false</code> . Per ulteriori informazioni, consulta Aggiornamenti OTA del software AWS IoT Greengrass Core .

Campo	Descrizione	Note
writeDirectory	Facoltativo. La directory di scrittura in cui vengono AWS IoT Greengrass create tutte le risorse di lettura/scrittura.	Per ulteriori informazioni, consulta Configura una directory di scrittura per AWS IoT Greengrass .

GGC v1.7

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

Il file `config.json` supporta le seguenti proprietà:

coreThing

Campo	Descrizione	Note
caPath	Il percorso della CA AWS IoT principale relativo alla <i>/greengrass-root</i> / certs directory.	Per la compatibilità con le versioni precedenti alla 1.7.0. Questa proprietà viene ignorata quando l'oggetto è presente. crypto  Note Assicurati che gli endpoint corrispondano al tipo di certificato.
certPath	Il percorso del certificato del dispositivo core relativo alla directory <i>/greengrass-root</i> /certs.	Per la retrocompatibilità con le versioni precedenti alla 1.7.0. Questa proprietà viene ignorata quando l'oggetto è presente. crypto
keyPath	Il percorso della chiave privata core relativa alla directory <i>/greengrass-root</i> /certs.	Per la compatibilità con le versioni precedenti alla 1.7.0. Questa proprietà viene ignorata quando l'oggetto è presente. crypto
thingArn	L'Amazon Resource Name (ARN) dell' AWS IoT elemento che rappresenta il dispositivo AWS IoT Greengrass principale.	Trova l'ARN per il tuo core nella AWS IoT Greengrass console sotto Cores o eseguendo il comando aws greengrass get-core-

Campo	Descrizione	Note
		definition-version _CLI.
iotHost	Il tuo endpoint. AWS IoT	Trova l'endpoint nella AWS IoT console in Impostazioni o eseguendo il comando aws iot describe-endpoint --endpoint-type iot:Data-ATS CLI. Questo comando restituisce l'endpoint Amazon Trust Services (ATS). Per ulteriori informazioni, consulta la documentazione Autenticazione del server.  Note Assicurati che gli endpoint corrispondano al tipo di certificato. Assicurati che i tuoi endpoint corrispondano ai tuoi. Regione AWS

Campo	Descrizione	Note
ggHost	Il tuo AWS IoT Greengrass endpoint.	Questo è l'endpoint <code>iotHost</code> con il prefisso <code>host</code> sostituito da <code>greengrass</code> (ad esempio, <code>greengrass-ats.iot.region.amazonaws.com</code>). Usa lo stesso Regione AWS di <code>iotHost</code>.  Note Assicurati che gli endpoint corrispondano al tipo di certificato. Assicurati che i tuoi endpoint corrispondano ai tuoi Regione AWS
iotMqttPort	Facoltativo. Il numero di porta da utilizzare per la comunicazione MQTT con AWS IoT	I valori validi sono 8883 e 443. Il valore predefinito è 8883. Per ulteriori informazioni, consulta Connessione alla porta 443 o tramite un proxy di rete .
keepAlive	Facoltativo. Il periodo MQTT KeepAlive in secondi.	L'intervallo valido è 30-1200 secondi. Il valore predefinito è 600.
networkProxy	Facoltativo. Un oggetto che definisce un server proxy a cui connettersi.	Il server proxy può essere HTTP o HTTPS. Per ulteriori informazioni, consulta Connessione alla porta 443 o tramite un proxy di rete .

runtime

Campo	Descrizione	Note
cgroup		
useSystemd	Indica se il dispositivo utilizza systemd .	I valori validi sono yes e no. Esegui lo script <code>check_ggc_dependencies</code> nel Modulo 1 per visualizzare se il dispositivo usa systemd.

crypto

L'oggetto `crypto` aggiunto nella versione v1.7.0, introduce proprietà che supportano lo storage di chiavi private in un modulo di sicurezza hardware (HSM) tramite PKCS#11 e storage di segreti locali. Per ulteriori informazioni, consulta [the section called “Integrazione della sicurezza hardware”](#) e [Distribuzione dei segreti nel core](#). Sono supportate le configurazioni per l'archiviazione delle chiavi private su HSMs o nel file system.

Campo	Descrizione	Note
caPath	Il percorso assoluto verso la CA AWS IoT principale.	Deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code> .

 **Note**

Assicurati che gli [endpoint corrispondano al tipo di certificato](#).

PKCS11

Campo	Descrizione	Note
OpenSSLEngine	Facoltativo. Il percorso assoluto del file .so del motore OpenSSL per abilitare il supporto PKCS#11 su OpenSSL.	Deve essere un percorso di un file nel file system. Questa proprietà è obbligatoria se si utilizza l'agente di aggiornamento OTA Greengrass con sicurezza hardware. Per ulteriori informazioni, consulta the section called “Configurazione degli aggiornamenti OTA” .
P11Provider	Il percorso assoluto della libreria libdl-loadable dell'implementazione PKCS#11.	Deve essere un percorso di un file nel file system.
slotLabel	L'etichetta dello slot utilizzata per identificare il modulo hardware.	Deve essere conforme alle specifiche dell'etichetta PKCS#11.
slotUserPin	Il PIN utente utilizzato per autenticare il core Greengrass nel modulo.	Deve disporre delle autorizzazioni sufficienti a eseguire C_Sign con le chiavi private configurate.
principals		
IoTCertificate	Il certificato e la chiave privata che il core utilizza per effettuare le richieste a AWS IoT.	

Campo	Descrizione	Note
IoTCertificate .privateKeyPath	Il percorso della chiave privata del core.	Per lo storage di file system, deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code>. Per lo storage HSM, deve essere un percorso PKCS#11 RFC 7512 che specifica l'etichetta dell'oggetto.
IoTCertificate .certificatePath	Il percorso assoluto al certificato del dispositivo core.	Deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code> .
MQTTServerCertificate	Facoltativo. La chiave privata che il core utilizza insieme al certificato per fungere da gateway o server MQTT.	

Campo	Descrizione	Note
MQTTServerCertificatePrivateKeyPath	Il percorso della chiave privata del server MQTT locale.	Utilizza questo valore per specificare la tua chiave privata del server MQTT locale. Per lo storage di file system, deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code>. Per lo storage HSM, deve essere un percorso PKCS#11 RFC 7512 che specifica l'etichetta dell'oggetto. Se questa proprietà viene omessa, AWS IoT Greengrass ruota la chiave in base alle impostazioni di rotazione. Se viene specificata, il cliente sarà responsabile della rotazione della chiave.
SecretsManager	La chiave privata che protegge la chiave di dati utilizzata per la crittografia. Per ulteriori informazioni, consulta Distribuzione dei segreti nel core .	

Campo	Descrizione	Note
SecretsManager.privateKeyPath	Il percorso della chiave privata del Secrets Manager locale.	Solo una chiave RSA è supportata. Per lo storage di file system, deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code>. Per lo storage HSM, deve essere un percorso PKCS#11 RFC 7512 che specifica l'etichetta dell'oggetto. La chiave privata deve essere generata utilizzando il meccanismo di padding PKCS#1 v1.5.

Sono supportate anche le seguenti proprietà di configurazione:

Campo	Descrizione	Note
mqttMaxConnectionRetryInterval	Facoltativo. L'intervallo massimo (in secondi) tra tentativi di connessione MQTT se la connessione viene interrotta.	Specifica questo valore come numero intero senza segno. Il valore predefinito è 60.
managedRespawn	Facoltativo. Indica che l'agente OTA deve eseguire il codice personalizzato prima di un aggiornamento.	I valori validi sono <code>true</code> e <code>false</code> . Per ulteriori informazioni, consulta Aggiornamenti OTA del software AWS IoT Greengrass Core .

Campo	Descrizione	Note
writeDirectory	Facoltativo. La directory di scrittura in cui vengono AWS IoT Greengrass create tutte le risorse di lettura/scrittura.	Per ulteriori informazioni, consulta Configura una directory di scrittura per AWS IoT Greengrass .

GGC v1.6

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

 Note

Se si utilizza l'opzione di creazione del gruppo predefinito dalla AWS IoT Greengrass console, il `config.json` file viene distribuito sul dispositivo principale in uno stato funzionante che specifica la configurazione predefinita.

Il file `config.json` supporta le seguenti proprietà:

Campo	Descrizione	Note
caPath	Il percorso della CA root AWS IoT relativa alla directory <i>/greengrass-root/certs</i> .	Salva il file in <i>/greengrass-root/certs</i> .
certPath	Il percorso del certificato di AWS IoT Greengrass base relativo alla <i>/greengrass-root/certs</i> directory.	Salva il file in <i>/greengrass-root/certs</i> .
keyPath	Il percorso della AWS IoT Greengrass chiave privata principale relativa alla <i>/greengrass-root/certs</i> directory.	Salva il file in <i>/greengrass-root/certs</i> .
thingArn	L'Amazon Resource Name (ARN) dell' AWS IoT elemento che rappresenta il dispositivo AWS IoT Greengrass principale.	Trova l'ARN per il tuo core nella AWS IoT Greengrass console sotto Cores o eseguendo il comando aws greengrass get-core-definition-version CLI.
iotHost	Il tuo endpoint. AWS IoT	Trovalo nella AWS IoT console in Impostazioni o eseguendo il comando aws iot describe-endpoint CLI.
ggHost	Il tuo AWS IoT Greengrass endpoint.	Questo valore usa il formato greengrass.s.iot. <i>region</i> .amazonaws.com . Usa la stessa regione di iotHost.

Campo	Descrizione	Note
<code>keepAlive</code>	Il periodo MQTT KeepAlive in secondi.	Questo valore è opzionale. Il valore predefinito è 600.
<code>mqttMaxConnectionRetryInterval</code>	L'intervallo massimo (in secondi) tra tentativi di connessione MQTT se la connessione viene interrotta.	Specifica questo valore come numero intero senza segno. Questo valore è opzionale. Il valore predefinito è 60.
<code>useSystemd</code>	Indica se il dispositivo utilizza systemd .	I valori validi sono yes e no. Esegui lo script <code>check_ggc_dependencies</code> nel Modulo 1 per visualizzare se il dispositivo usa systemd.
<code>managedRespawn</code>	Una funzionalità di aggiornamento opzionale over-the-air (OTA), indica che l'agente OTA deve eseguire codice personalizzato prima di un aggiornamento.	I valori validi sono true e false. Per ulteriori informazioni, consulta Aggiornamenti OTA del software AWS IoT Greengrass Core .
<code>writeDirectory</code>	La directory di scrittura in cui vengono AWS IoT Greengrass create tutte le risorse di lettura/scrittura.	Questo valore è opzionale. Per ulteriori informazioni, consulta Configura una directory di scrittura per AWS IoT Greengrass .

GGC v1.5

```
{
  "coreThing": {
    "caPath": "root-ca-pem",
    "certPath": "cloud-pem-crt",
    "keyPath": "cloud-pem-key",
```

```

    "thingArn": "arn:aws:iot:region:account-id:thing/core-thing-name",
    "iotHost": "host-prefix.iot.region.amazonaws.com",
    "ggHost": "greengrass.iot.region.amazonaws.com",
    "keepAlive": 600
  },
  "runtime": {
    "cgroup": {
      "useSystemd": "yes/no"
    }
  },
  "managedRespawn": true
}

```

Il file `config.json` esiste in `/greengrass-root/config` e contiene i parametri seguenti:

Campo	Descrizione	Note
caPath	Il percorso della CA root AWS IoT relativa alla cartella <code>/greengrass-root/certs</code> .	Salvare il file nella cartella <code>/greengrass-root/certs</code> .
certPath	Il percorso del certificato di AWS IoT Greengrass base relativo alla <code>/greengrass-root/certs</code> cartella.	Salvare il file nella cartella <code>/greengrass-root/certs</code> .
keyPath	Il percorso della AWS IoT Greengrass chiave privata principale relativa alla <code>/greengrass-root/certs</code> cartella.	Salvare il file nella cartella <code>/greengrass-root/certs</code> .
thingArn	L'Amazon Resource Name (ARN) dell' AWS IoT elemento che rappresenta il dispositivo AWS IoT Greengrass principale.	Trova l'ARN per il tuo core nella AWS IoT Greengrass console sotto Cores o eseguendo il comando aws greengrass get-core-

Campo	Descrizione	Note
		definition-version _CLI.
iotHost	Il tuo endpoint. AWS IoT	Trovalo nella AWS IoT console sotto Impostazioni o eseguendo il aws iot describe-endpoint comando.
ggHost	Il tuo AWS IoT Greengrass endpoint.	Questo valore usa il formato greengrass.iot. <i>region</i> .amazonaws.com. Usa la stessa regione di iotHost.
keepAlive	Il periodo MQTT KeepAlive in secondi.	Questo valore è opzionale. Il valore predefinito è 600 secondi.
useSystemd	Indica se il dispositivo utilizza systemd .	I valori validi sono yes e no. Esegui lo script check_ggc_dependencies nel Modulo 1 per visualizzare se il dispositivo usa systemd.
managedRespawn	Una funzionalità di aggiornamento opzionale over-the-air (OTA), indica che l'agente OTA deve eseguire codice personalizzato prima di un aggiornamento.	Per ulteriori informazioni, consulta Aggiornamenti OTA del software AWS IoT Greengrass Core .

GGC v1.3

{

```

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

```

Il file `config.json` esiste in `/greengrass-root/config` e contiene i parametri seguenti:

Campo	Descrizione	Note
caPath	Il percorso della CA root AWS IoT relativa alla cartella <code>/greengrass-root/certs</code> .	Salvare il file nella cartella <code>/greengrass-root/certs</code> .
certPath	Il percorso del certificato AWS IoT Greengrass principale relativo alla <code>/greengrass-root/certs</code> cartella.	Salvare il file nella cartella <code>/greengrass-root/certs</code> .
keyPath	Il percorso della AWS IoT Greengrass chiave privata principale relativa alla <code>/greengrass-root/certs</code> cartella.	Salvare il file nella cartella <code>/greengrass-root/certs</code> .
thingArn	L'Amazon Resource Name (ARN) dell' AWS IoT	Puoi trovare questo valore nella AWS IoT Greengrass

Campo	Descrizione	Note
	elemento che rappresenta il AWS IoT Greengrass core.	s console sotto la definizione del tuo programma AWS IoT .
iotHost	Il tuo AWS IoT endpoint.	Puoi trovare questo valore nella AWS IoT console in Impostazioni.
ggHost	Il tuo AWS IoT Greengrass endpoint.	Puoi trovare questo valore nella AWS IoT console in Impostazioni con greengrass . prefisso.
keepAlive	Il periodo MQTT KeepAlive in secondi.	Questo valore è opzionale . Il valore predefinito è 600 secondi.
useSystemd	Un flag binario, se il dispositivo utilizza systemd .	I valori sono yes o no. Usa lo script di dipendenza nel Modulo 1 per visualizzare se il dispositivo usa systemd.
managedRespawn	Una funzionalità di aggiornamento opzionale over-the-air (OTA), indica che l'agente OTA deve eseguire codice personalizzato prima di un aggiornamento.	Per ulteriori informazioni, consulta Aggiornamenti OTA del software AWS IoT Greengrass Core .

GGC v1.1

```
{
  "coreThing": {
    "caPath": "root-ca-pem",
    "certPath": "cloud-pem-crt",
    "keyPath": "cloud-pem-key",
```

```
"thingArn": "arn:aws:iot:region:account-id:thing/core-thing-name",
"iotHost": "host-prefix.iot.region.amazonaws.com",
"ggHost": "greengrass.iot.region.amazonaws.com",
"keepAlive": 600
},
"runtime": {
  "cgroup": {
    "useSystemd": "yes/no"
  }
}
}
```

Il file `config.json` esiste in `/greengrass-root/config` e contiene i parametri seguenti:

Campo	Descrizione	Note
caPath	Il percorso della CA root AWS IoT relativa alla cartella <code>/greengrass-root/certs</code> .	Salvare il file nella cartella <code>/greengrass-root / certs</code> .
certPath	Il percorso del certificato AWS IoT Greengrass principale relativo alla <code>/greengrass-root / certs</code> cartella.	Salvare il file nella cartella <code>/greengrass-root / certs</code> .
keyPath	Il percorso della AWS IoT Greengrass chiave privata principale relativa alla <code>/greengrass-root / certs</code> cartella.	Salvare il file nella cartella <code>/greengrass-root / certs</code> .
thingArn	L'Amazon Resource Name (ARN) dell' AWS IoT elemento che rappresenta il AWS IoT Greengrass core.	Puoi trovare questo valore nella AWS IoT Greengrass console sotto la definizione del tuo programma AWS IoT .

Campo	Descrizione	Note
iotHost	Il tuo AWS IoT endpoint.	Puoi trovare questo valore nella AWS IoT console in Impostazioni.
ggHost	Il tuo AWS IoT Greengrass endpoint.	Puoi trovare questo valore nella AWS IoT console in Impostazioni con greengrass. prefisso.
keepAlive	Il periodo MQTT KeepAlive in secondi.	Questo valore è opzionale. Il valore predefinito è 600 secondi.
useSystemd	Un flag binario, se il dispositivo utilizza systemd .	I valori sono yes o no. Usa lo script di dipendenza nel Modulo 1 per visualizzare se il dispositivo usa systemd.

GGC 1.0

In AWS IoT Greengrass Core v1.0, `config.json` viene distribuito su. *greengrass-root/*configuration

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

}

Il file `config.json` esiste in `/greengrass-root/configuration` e contiene i parametri seguenti:

Campo	Descrizione	Note
<code>caPath</code>	Il percorso della CA root AWS IoT relativa alla cartella <code>/greengrass-root/configuration/certs</code> .	Salvare il file nella cartella <code>/greengrass-root / configuration/certs</code> .
<code>certPath</code>	Il percorso del certificato di AWS IoT Greengrass base relativo alla <code>/greengrass-root / configuration/certs</code> cartella.	Salvare il file nella cartella <code>/greengrass-root / configuration/certs</code> .
<code>keyPath</code>	Il percorso della AWS IoT Greengrass chiave privata principale relativa alla <code>/greengrass-root / configuration/certs</code> cartella.	Salvare il file nella cartella <code>/greengrass-root / configuration/certs</code> .
<code>thingArn</code>	L'Amazon Resource Name (ARN) dell' AWS IoT elemento che rappresenta il AWS IoT Greengrass core.	Puoi trovare questo valore nella AWS IoT Greengrass console sotto la definizione del tuo AWS IoT thing.
<code>iotHost</code>	Il tuo AWS IoT endpoint.	Puoi trovare questo valore nella AWS IoT console in Impostazioni.
<code>ggHost</code>	Il tuo AWS IoT Greengrass endpoint.	Puoi trovare questo valore nella AWS IoT console

Campo	Descrizione	Note
		in Impostazioni con greengrass. prefisso.
keepAlive	Il periodo MQTT KeepAlive in secondi.	Questo valore è opzionale. Il valore predefinito è 600 secondi.
useSystemd	Un flag binario, se il dispositivo utilizza systemd .	I valori sono yes o no. Usa lo script di dipendenza nel Modulo 1 per visualizzare se il dispositivo usa systemd.

Gli endpoint del servizio devono corrispondere al tipo di certificato CA principale

AWS IoT Greengrass Gli endpoint AWS IoT Core e gli endpoint devono corrispondere al tipo di certificato del certificato CA principale sul dispositivo. Se gli endpoint e il tipo di certificato non corrispondono, i tentativi di autenticazione tra il dispositivo e AWS IoT Core o hanno esito negativo. AWS IoT Greengrass Per ulteriori informazioni, consulta [Autenticazione del server](#) nella Guida per gli sviluppatori di AWS IoT.

Se il tuo dispositivo utilizza un certificato CA root di Amazon Trust Services (ATS), che è il metodo preferito, deve utilizzare anche gli endpoint ATS per la gestione dei dispositivi e le operazioni del piano dati di rilevamento. Gli endpoint ATS includono il `ats` segmento, come illustrato nella seguente sintassi per l'endpoint. AWS IoT Core

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

Note

Per motivi di compatibilità con le versioni precedenti, AWS IoT Greengrass attualmente supporta i certificati e gli endpoint CA VeriSign root legacy in alcuni casi. Regione AWS Se utilizzi un certificato CA VeriSign radice legacy, ti consigliamo di creare un endpoint ATS e utilizzare invece un certificato CA radice ATS. Altrimenti utilizza i corrispondenti endpoint

legacy. Per ulteriori informazioni, consulta [Endpoint legacy supportati](#) nella Riferimenti generali di Amazon Web Services.

Endpoint in config.json

Su un dispositivo principale di Greengrass, gli endpoint sono specificati nell'oggetto `coreThing` nel [config.json](#) file. La `iotHost` proprietà rappresenta l' AWS IoT Core endpoint. La `ggHost` proprietà rappresenta l' AWS IoT Greengrass endpoint. In questo frammento di esempio, le proprietà specificano gli endpoint ATS.

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

AWS IoT Core endpoint

Puoi ottenere il tuo AWS IoT Core endpoint eseguendo il comando [aws iot describe-endpoint](#) CLI con il `--endpoint-type` parametro appropriato.

- Per restituire un endpoint firmato con ATS, esegui:

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

- Per restituire un endpoint VeriSign firmato legacy, esegui:

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

AWS IoT Greengrass endpoint

Il tuo AWS IoT Greengrass endpoint è il tuo `iotHost` endpoint con il prefisso `host` sostituito da `greengrass`. Ad esempio, l'endpoint firmato con ATS è `greengrass-ats.iot.region.amazonaws.com`. Questo utilizza la stessa regione dell'endpoint. AWS IoT Core

Connessione alla porta 443 o tramite un proxy di rete

Questa funzionalità è disponibile per AWS IoT Greengrass Core v1.7 e versioni successive.

I core Greengrass comunicano AWS IoT Core utilizzando il protocollo di messaggistica MQTT con l'autenticazione client TLS. Per convenzione, il protocollo MQTT su TLS utilizza la porta 8883. Tuttavia, come misura di sicurezza, gli ambienti restrittivi potrebbero limitare il traffico in entrata e in uscita a un numero ridotto di porte TCP. Ad esempio, un firewall aziendale potrebbe aprire la porta 443 per il traffico HTTPS, ma chiudere le altre porte utilizzate per i protocolli meno comuni, come la porta 8883 per il traffico MQTT. Altri ambienti restrittivi potrebbero richiedere l'indirizzamento di tutto il traffico a un proxy HTTP prima della connessione a Internet.

Per abilitare la comunicazione in questi scenari, AWS IoT Greengrass consente le seguenti configurazioni:

- MQTT con autenticazione client TLS sulla porta 443. Se la rete in uso consente connessioni alla porta 443, è possibile configurare il core in modo da utilizzare la porta 443 anziché la porta predefinita 8883 per il traffico MQTT. Si può trattare di una connessione diretta alla porta 443 o di una connessione tramite un server proxy di rete.

AWS IoT Greengrass utilizza l'estensione TLS [Application Layer Protocol Network](#) (ALPN) per abilitare questa connessione. Come per la configurazione predefinita, sulla porta 443 il protocollo MQTT su TLS utilizza l'autenticazione client basata sul certificato.

Se configurato per utilizzare una connessione diretta alla porta 443, il core supporta [gli aggiornamenti over-the-air \(OTA\)](#) per il software. AWS IoT Greengrass Questo supporto richiede AWS IoT Greengrass Core v1.9.3 o versione successiva.

- Comunicazione HTTPS tramite la porta 443. AWS IoT Greengrass invia il traffico HTTPS sulla porta 8443 per impostazione predefinita, ma è possibile configurarlo per utilizzare la porta 443.
- Connessione tramite un proxy di rete. Puoi configurare un server proxy di rete in modo che funga da intermediario per la connessione al core Greengrass. Sono supportati solo l'autenticazione di base e i proxy HTTP e HTTPS.

La configurazione del proxy viene passata alle funzioni Lambda definite dall'utente tramite `http_proxy` le variabili di `https_proxy` ambiente, `no_proxy` e. Le funzioni Lambda definite dall'utente devono utilizzare queste impostazioni passate per connettersi tramite il proxy. Le librerie comuni utilizzate dalle funzioni Lambda per effettuare connessioni (come boto3 o cURL e pacchetti `requests python`) in genere utilizzano queste variabili di ambiente per impostazione predefinita.

Se una funzione Lambda specifica anche queste stesse variabili di ambiente, AWS IoT Greengrass non le sovrascrive.

⚠ Important

I dispositivi Greengrass Core che sono configurati per utilizzare un proxy di rete non supportano [aggiornamenti OTA](#).

Per configurare MQTT sulla porta 443

Questa funzionalità richiede AWS IoT Greengrass Core v1.7 o versione successiva.

Questa procedura consente al core Greengrass di utilizzare la porta 443 per la messaggistica MQTT con AWS IoT Core.

1. Eseguite il seguente comando per arrestare il demone Greengrass:

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

2. Apri `greengrass-root/config/config.json` per la modifica come utente su.
3. Nell'oggetto `coreThing`, aggiungere la proprietà `iotMqttPort` e impostare il valore su **443**, come mostrato nel seguente esempio.

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

4. Avvia il daemon.

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

Per configurare HTTPS sulla porta 443

Questa funzionalità richiede AWS IoT Greengrass Core v1.8 o versione successiva.

Questa procedura consente di configurare il core per utilizzare la porta 443 per la comunicazione HTTPS.

1. Eseguite il seguente comando per arrestare il demone Greengrass:

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

2. Apri **greengrass-root**/config/config.json per la modifica come utente su.
3. Nell'oggetto coreThing, aggiungere le proprietà iotHttpPort e ggHttpPort, come mostrato nel seguente esempio.

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

4. Avvia il daemon.

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

Per configurare un proxy di rete

Questa funzionalità richiede AWS IoT Greengrass Core v1.7 o versione successiva.

Questa procedura consente di AWS IoT Greengrass connettersi a Internet tramite un proxy di rete HTTP o HTTPS.

1. Eseguite il seguente comando per arrestare il demone Greengrass:

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

2. Apri *greengrass-root*/config/config.json per la modifica come utente su.
3. Nell'oggetto coreThing, aggiungere l'oggetto [networkProxy](#) come mostrato nel seguente esempio.

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

4. Avvia il daemon.

```
cd /greengrass-root/ggc/core/
```

```
sudo ./greengrassd start
```

oggetto networkProxy

Utilizza l'oggetto `networkProxy` per specificare le informazioni sul proxy di rete. Questo oggetto ha le proprietà seguenti.

Campo	Descrizione
<code>noProxyAddresses</code>	Facoltativo. Un elenco separato da virgole di indirizzi IP o nomi host esenti dal proxy.
<code>proxy</code>	Il proxy a cui connettersi. Un proxy ha le seguenti proprietà. • <code>url</code>. L'URL del server proxy, nel formato <code>scheme://userinfo@host:port</code>. • <code>scheme</code>. Lo schema. Deve essere <code>http</code> o <code>https</code>. • <code>userinfo</code>. Opzionale. Le informazioni sul nome utente e la password. Se specificate, i campi <code>password</code> e <code>username</code> vengono ignorati. • <code>host</code>. Il nome host o l'indirizzo IP del server proxy. • <code>port</code>. Opzionale. Il numero di porta. Se non è specificato, verranno utilizzati i seguenti valori predefiniti: • <code>http</code>: 80 • <code>https</code>: 443 • <code>username</code>. Opzionale. Il nome utente da utilizzare per eseguire l'autenticazione nel server proxy.

Campo	Descrizione
	password. Opzionale. La password da utilizzare per eseguire l'autenticazione nel server proxy.

Consentire gli endpoint

La comunicazione tra i dispositivi Greengrass AWS IoT Core e/o AWS IoT Greengrass deve essere autenticata. Questa autenticazione si basa su certificati di dispositivo X.509 registrati e chiavi crittografiche. Per consentire alle richieste autenticate di passare attraverso i proxy senza crittografia aggiuntiva, consentire i seguenti endpoint.

Endpoint	Porta	Descrizione
greengrass. <i>region</i> .amazonaws.com	443	Utilizzato per controllare le operazioni del piano per la gestione dei gruppi.
<i>prefix</i> -ats.iot. <i>region</i> .amazonaws.com oppure <i>prefix</i> .iot. <i>region</i> .amazonaws.com	MQTT: 8883 o 443 HTTPS: 8443 o 443	Utilizzato per operazioni del piano dei dati per la gestione dei dispositivi, ad esempio la sincronizzazione shadow. Consenti l'uso di uno o entrambi gli endpoint,

Endpoint	Porta	Descrizione
		a seconda che i dispositivi core e client utilizzino certificati CA root di Amazon Trust Services (preferiti), certificati CA root legacy o entrambi. Per ulteriori informazioni, consulta the section called “Gli endpoint del servizio devono corrispondere al tipo di certificato” .

Endpoint	Porta	Descrizione
greengrass-ats.iot . <i>region</i> .amazonaws.com oppure greengrass.iot. <i>region</i> .amazonaws.com	8443 o 443	Utilizzato per operazioni di rilevamento dispositivo. Consenti l'uso di uno o entrambi gli endpoint, a seconda che i dispositivi core e client utilizzino o certifichino CA root di Amazon Trust Services (preferiti), certificati CA root legacy o entrambi. Per ulteriori informazioni, consulta the section called “Gli endpoint del servizio devono corrispondere al tipo di certificato” .

Endpoint	Porta	Descrizione
		Note I client che si connettono alla porta 443 devono implementare l'estensione TLS Application Layer Protocol Negotiation (ALPN) e passare x-amzn-http-ca come ProtocolName o ProtocolNameList. Per

Endpoint	Porta	Descrizione
		ulteriori informazioni, consulta Protocols nella Developer Guide.AWS IoT
*.s3.amazonaws.com	443	Utilizzato per operazioni di distribuzione e over-the-air aggiornamenti. Questo formato include il carattere * perché i prefissi endpoint vengono controllati internamente e potrebbero o cambiare in qualsiasi momento.

Endpoint	Porta	Descrizione
logs. <i>region</i> .amazonaws.com	443	Richiesto se il gruppo Greengrass viene configurato per scrivere i log in CloudWatch.

Configura una directory di scrittura per AWS IoT Greengrass

Questa funzionalità è disponibile per AWS IoT Greengrass Core v1.6 e versioni successive.

Per impostazione predefinita, il software AWS IoT Greengrass Core viene distribuito in un'unica directory principale in cui AWS IoT Greengrass esegue tutte le operazioni di lettura e scrittura. Tuttavia, è possibile configurare l'utilizzo AWS IoT Greengrass di una directory separata per tutte le operazioni di scrittura, inclusa la creazione di directory e file. In questo caso, AWS IoT Greengrass utilizza due directory di primo livello:

- La directory *greengrass-root*, che può essere lasciata in lettura/scrittura o essere resa, se necessario, di sola lettura. Contiene il software AWS IoT Greengrass Core e altri componenti critici che dovrebbero rimanere immutabili durante il runtime, come certificati e `config.json`
- La directory di scrittura specificata. Contiene contenuti scrivibili, come registri, informazioni sullo stato e funzioni Lambda definite dall'utente distribuite.

Questa configurazione determina la seguente struttura di directory.

Directory principale Greengrass

```
greengrass-root/
|-- certs/
|   |-- root.ca.pem
|   |-- hash.cert.pem
|   |-- hash.private.key
|   |-- hash.public.key
|-- config/
```

```
| |-- config.json
|-- ggc/
| |-- packages/
|     |-- package-version/
|         |-- bin/
|             |-- daemon
|             |-- greengrassd
|             |-- lambda/
|             |-- LICENSE/
|             |-- release_notes_package-version.html
|             |-- runtime/
|                 |-- java8/
|                 |-- nodejs8.10/
|                 |-- python3.8/
|-- core/
```

Directory di scrittura

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

Per configurare una directory di scrittura

1. Esegui il comando seguente per fermare il demone: AWS IoT Greengrass

```
cd /greengrass-root/ggc/core/
```

```
sudo ./greengrassd stop
```

2. Apri **greengrass-root**/config/config.json per la modifica come utente su.
3. Aggiungi writeDirectory come parametro e specifica il percorso della directory di destinazione, come mostrato nel seguente esempio.

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

Note

Puoi aggiornare le impostazioni di writeDirectory in qualsiasi momento. Dopo l'aggiornamento dell'impostazione, AWS IoT Greengrass utilizza la directory di scrittura appena specificata all'avvio successivo, ma non migra il contenuto dalla directory di scrittura precedente.

4. Ora che la directory di scrittura è configurata, puoi rendere la directory **greengrass-root** di sola lettura. Per maggiori istruzioni, consulta [Come rendere la directory principale Greengrass di sola lettura](#).

Altrimenti, avvia il AWS IoT Greengrass demone:

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

Per rendere la directory principale di Greengrass di sola lettura

Seguire queste fasi solo se si desidera rendere la directory principale Greengrass di sola lettura. La directory di scrittura deve essere configurata prima di iniziare.

1. Concedere le autorizzazioni di accesso alle directory richieste:

- a. Fornisci le autorizzazioni di lettura e scrittura al proprietario di `config.json`.

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

- b. Rendi `ggc_user` il proprietario dei certificati e delle directory Lambda del sistema.

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

Note

Gli account `ggc_user` e `ggc_group` vengono utilizzati di default per eseguire le funzioni Lambda di sistema. Se hai configurato l'[identità di accesso predefinita](#) a livello di gruppo per utilizzare account differenti, è consigliabile concedere le autorizzazioni a tali utente (UID) e gruppo (GID).

2. Rendi la directory `greengrass-root` di sola lettura utilizzando il meccanismo preferito.

Note

Un modo per rendere la directory `greengrass-root` di sola lettura è montare la directory come di sola lettura. Tuttavia, per applicare gli aggiornamenti over-the-air (OTA) al software AWS IoT Greengrass Core in una directory montata, la directory deve prima essere smontata e poi rimontata dopo l'aggiornamento. Puoi aggiungere queste operazioni `umount` e `mount` agli script `ota_pre_update` e `ota_post_update`. Per ulteriori informazioni sugli aggiornamenti OTA, consulta [the section called “Agente di aggiornamento OTA di Greengrass”](#) e [the section called “Rigenerazione gestita con aggiornamenti OTA”](#).

3. Avvia il daemon.

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

Se le autorizzazioni della fase 1 non sono state impostate correttamente, il daemon non verrà avviato.

Configurazione delle impostazioni MQTT

Nell' AWS IoT Greengrass ambiente, i dispositivi client locali, le funzioni Lambda, i connettori e i componenti di sistema possono comunicare tra loro e con. AWS IoT Core. Tutta la comunicazione passa attraverso il core, che gestisce le [sottoscrizioni](#) che autorizzano la comunicazione MQTT tra entità.

Per informazioni sulle impostazioni MQTT che è possibile configurare AWS IoT Greengrass, consulta le seguenti sezioni:

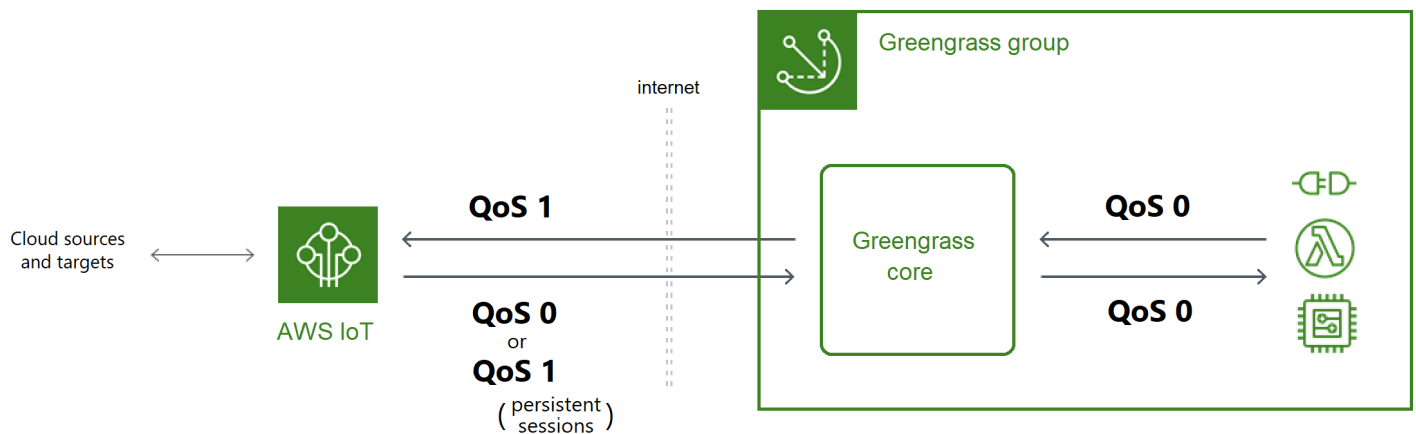
- [the section called “Messaggio di qualità del servizio”](#)
- [the section called “Coda di messaggi MQTT”](#)
- [the section called “Sessioni persistenti MQTT con AWS IoT Core”](#)
- [the section called “Client IDs per connessioni MQTT con AWS IoT”](#)
- [Porta MQTT per la messaggistica locale](#)
- [the section called “Timeout per le operazioni di pubblicazione, sottoscrizione e annullamento dell'iscrizione nelle connessioni MQTT con Cloud AWS”](#)

Note

OPC-UA è uno standard di scambio di informazioni per la comunicazione industriale. [Per implementare il supporto per OPC-UA sul core Greengrass, puoi utilizzare il connettore IoT SiteWise](#). Il connettore invia i dati dei dispositivi industriali dai server OPC-UA alle proprietà degli asset in. AWS IoT SiteWise

Messaggio di qualità del servizio

AWS IoT Greengrass supporta i livelli di qualità del servizio (QoS) 0 o 1, a seconda della configurazione, dell'obiettivo e della direzione della comunicazione. Il core Greengrass funge da client per la comunicazione AWS IoT Core e broker di messaggi per la comunicazione sulla rete locale.



Per ulteriori informazioni su MQTT e QoS, [vedere Guida introduttiva](#) sul sito Web MQTT.

Comunicazione con Cloud AWS

- I messaggi in uscita utilizzano QoS 1

Il core invia messaggi destinati Cloud AWS agli obiettivi utilizzando QoS 1. AWS IoT Greengrass utilizza una coda di messaggi MQTT per elaborare questi messaggi. Se il recapito del messaggio non è confermato da AWS IoT, il messaggio viene spoolato per essere riprovato in un secondo momento. Il messaggio non può essere riprovato se la coda è piena. La conferma del recapito del messaggio può aiutare a ridurre al minimo la perdita di dati dovuta alla connettività intermittente.

Poiché i messaggi in uscita AWS IoT utilizzano QoS 1, la velocità massima con cui il core Greengrass può inviare messaggi dipende dalla latenza tra il core e AWS IoT. Ogni volta che il core invia un messaggio, attende che lo AWS IoT confermi prima di inviare il messaggio successivo. Ad esempio, se il tempo di andata e ritorno tra il core e la sua Regione AWS è di 50 millisecondi, il core può inviare fino a 20 messaggi al secondo. Considerate questo comportamento quando scegliete il punto di connessione del core Regione AWS. Per importare dati IoT ad alto volume su Cloud AWS, puoi utilizzare [stream manager](#).

Per ulteriori informazioni sulla coda di messaggi MQTT, incluso come configurare una cache di archiviazione locale in grado di rendere persistenti i messaggi destinati alle destinazioni, vedere Cloud AWS [the section called “Coda di messaggi MQTT”](#)

- I messaggi in entrata utilizzano QoS 0 (predefinito) o QoS 1

Per impostazione predefinita, il core si iscrive con QoS 0 ai messaggi Cloud AWS provenienti dalle sorgenti. Se si abilitano sessioni persistenti, il core effettua la sottoscrizione con QoS 1.

Ciò consente di ridurre al minimo la perdita di dati causata dall'instabilità della connessione. Per gestire il QoS per queste sottoscrizioni, configura le impostazioni di persistenza sul componente di sistema di spooler locale.

Per ulteriori informazioni, incluso come consentire al core di stabilire una sessione persistente con Cloud AWS obiettivi, vedere. [the section called “Sessioni persistenti MQTT con AWS IoT Core”](#)

Comunicazione con target locali

Tutte le comunicazioni locali utilizzano QoS 0. [Il core tenta di inviare un messaggio a una destinazione locale, che può essere una funzione Greengrass Lambda, un connettore o un dispositivo client.](#) Il core non archivia i messaggi né conferma la consegna. I messaggi possono essere rimossi ovunque tra i componenti.

Note

Sebbene la comunicazione diretta tra le funzioni Lambda non utilizzi la messaggistica MQTT, il comportamento è lo stesso.

Coda di messaggi MQTT per destinazioni sul cloud

I messaggi MQTT destinati alle destinazioni vengono messi in coda in Cloud AWS attesa dell'elaborazione. I messaggi in coda vengono elaborati in ordine FIFO (first-in-first-out). Dopo l'elaborazione e la pubblicazione di un messaggio su AWS IoT Core, il messaggio viene rimosso dalla coda.

Per impostazione predefinita, il core Greengrass archivia in memoria i messaggi non elaborati destinati alle destinazioni. Cloud AWS Puoi configurare il core per archiviare invece i messaggi non elaborati in una cache di storage locale. A differenza dell'archiviazione in memoria, la cache di archiviazione locale ha la capacità di persistere anche dopo i riavvii principali (ad esempio, dopo una distribuzione di gruppo o il riavvio di un dispositivo), quindi AWS IoT Greengrass può continuare a elaborare i messaggi. È anche possibile configurare le dimensioni dello storage.

Warning

Il core Greengrass potrebbe mettere in coda messaggi MQTT duplicati quando perde la connessione, perché tenta nuovamente un'operazione di pubblicazione prima che il client MQTT rilevi che è offline. Per evitare la duplicazione dei messaggi MQTT per gli

obiettivi cloud, configura il valore del core a meno della metà del suo keepAlive valore. `mqttoperationTimeout` Per ulteriori informazioni, consulta [AWS IoT Greengrass file di configurazione principale](#).

AWS IoT Greengrass utilizza il componente del sistema spooler (la funzione `GGCloudSpooler` Lambda) per gestire la coda dei messaggi. Puoi utilizzare le seguenti variabili di ambiente `GGCloudSpooler` per configurare le impostazioni di storage.

- `GG_CONFIG_STORAGE_TYPE`. La posizione della coda di messaggi. I seguenti valori sono validi:
 - `FileSystem`. Archivia i messaggi non elaborati nella cache di archiviazione locale sul disco del dispositivo principale fisico. In caso di riavvio core, i messaggi in coda vengono conservati per l'elaborazione. I messaggi vengono rimossi dopo essere stati elaborati.
 - `Memory` (impostazione predefinita). Archivia i messaggi non elaborati in memoria. In caso di riavvio del core, i messaggi in coda vengono persi.

Questa opzione è ottimizzata per dispositivi con funzionalità hardware limitate. Quando si utilizza questa configurazione, consigliamo di distribuire gruppi o riavviare il dispositivo nel momento in cui l'interruzione di servizio non crea particolari problemi.

- `GG_CONFIG_MAX_SIZE_BYTES`. Le dimensioni dello storage, in byte. Questo valore può essere qualsiasi numero intero non negativo maggiore o uguale a 262144 (256 KB); una dimensione inferiore impedisce l'AWS IoT Greengrass avvio del software Core. Le dimensioni predefinite sono di 2,5 MB. Quando viene raggiunto il limite di dimensioni, i messaggi in coda più vecchi vengono sostituiti da nuovi messaggi.

Note

Questa funzionalità è disponibile per AWS IoT Greengrass Core v1.6 e versioni successive. Le versioni precedenti utilizzavano l'archiviazione in memoria con una dimensione della coda di 2,5 MB. Non è possibile configurare le impostazioni di storage per le versioni precedenti.

Come aggiungere i messaggi alla cache nello storage locale

È possibile AWS IoT Greengrass configurare la memorizzazione nella cache dei messaggi nel file system in modo che persistano anche dopo i riavvii principali. A tale scopo, distribuisce una

versione di definizione della funzione in cui la funzione GGCloudSpooler imposta il tipo di storage su FileSystem. È necessario utilizzare l' AWS IoT Greengrass API per configurare la cache di archiviazione locale. Questa operazione può essere eseguita nella console.

La procedura seguente utilizza il comando [create-function-definition-version](#) CLI per configurare lo spooler per salvare i messaggi in coda nel file system. Configura anche una coda di 2,6 MB.

1. Ottieni il gruppo Greengrass IDs di destinazione e la versione del gruppo. Questa procedura presuppone che questa sia la versione più recente del gruppo e del gruppo. La seguente query restituisce il gruppo creato più di recente.

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

In alternativa, puoi eseguire query in base al nome. I nomi dei gruppi non devono essere univoci, pertanto potrebbero essere restituiti più gruppi.

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

Note

Puoi trovare questi valori anche nella AWS IoT console. L'ID gruppo viene visualizzato nella pagina Settings (Impostazioni) del gruppo. IDs Le versioni del gruppo vengono visualizzate nella scheda Distribuzioni del gruppo.

2. Copiare i valori Id e LatestVersion dal gruppo target nell'output.
3. Ottieni la versione gruppo più recente.
 - Sostituisci l'*group-id* con l'Id copiato.
 - Sostituisci l'*latest-group-version-id* con l'LatestVersion copiato.

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

4. Dall'Definitionoggetto nell'output, copia il CoreDefinitionVersionArn e il ARNs di tutti gli altri componenti del gruppo tranne. FunctionDefinitionVersionArn Si usano questi valori quando si crea una nuova versione gruppo.
5. Da FunctionDefinitionVersionArn nell'output, copia l'ID della definizione della funzione. L'ID è il GUID che segue il segmento functions nell'ARN, come mostrato nell'esempio seguente.

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

Note

In alternativa, è possibile creare una definizione di funzione eseguendo il [create-function-definition](#) comando e quindi copiando l'ID dall'output.

6. Aggiungi una versione di definizione della funzione alla definizione della funzione.
 - *function-definition-id* Sostituiscilo con Id quello che hai copiato per la definizione della funzione.
 - Sostituire *arbitrary-function-id* con un nome per la funzione, ad esempio **spooler-function**.
 - Aggiungi all'array tutte le funzioni Lambda che desideri includere in questa versione. functions È possibile utilizzare il [get-function-definition-version](#) comando per ottenere le funzioni Greengrass Lambda da una versione di definizione di funzione esistente.

Warning

Assicurati di specificare un valore per GG_CONFIG_MAX_SIZE_BYTES che sia maggiore o uguale a 262144. Una dimensione più piccola impedisce l'avvio del software AWS IoT Greengrass Core.

```
aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
```

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

Note

Se in precedenza hai impostato la variabile di GG_CONFIG_SUBSCRIPTION_QUALITY ambiente per [supportare sessioni persistenti con AWS IoT Core](#), includila in questa istanza di funzione.

7. Copia l'Arn della definizione di funzione dall'output.
8. Crea una versione di gruppo che contenga la funzione Lambda del sistema.
 - Sostituisci *group-id* con Id for the group.
 - Sostituiscilo *core-definition-version-arn* con CoreDefinitionVersionArn quello che hai copiato dalla versione più recente del gruppo.
 - Sostituiscilo *function-definition-version-arn* con Arn quello che hai copiato per la nuova versione della definizione di funzione.
 - Sostituisci ARNs i componenti degli altri gruppi (ad esempio, SubscriptionDefinitionVersionArn o DeviceDefinitionVersionArn) che hai copiato dalla versione più recente del gruppo.
 - Rimuovi eventuali parametri inutilizzati. Ad esempio, rimuovi `--resource-definition-version-arn` se la versione gruppo non contiene risorse.

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

9. Copia il valore Version dall'output. Questo è l'ID della nuova versione gruppo.

10. Distribuisci il gruppo con la nuova versione del gruppo.

- Sostituiscilo *group-id* con Id quello che hai copiato per il gruppo.
- Sostituisci *group-version-id* con Version quello che hai copiato per la nuova versione del gruppo.

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

Per aggiornare le impostazioni di archiviazione, si utilizza l' AWS IoT Greengrass API per creare una nuova versione della definizione di funzione che contenga la GGCloudSpooler funzione con la configurazione aggiornata. Quindi aggiungi la versione della definizione della funzione a una nuova versione del gruppo (insieme agli altri componenti del gruppo) e distribuisci la versione del gruppo. Se desideri ripristinare la configurazione predefinita, puoi distribuire una versione di definizione della funzione che non include la funzione GGCloudSpooler.

Questa funzione Lambda di sistema non è visibile nella console. Tuttavia, dopo che la funzione è stata aggiunta all'ultima versione del gruppo, è inclusa nelle distribuzioni eseguite dalla console, a meno che non usi l'API per sostituirla o rimuoverla.

Sessioni persistenti MQTT con AWS IoT Core

Questa funzionalità è disponibile per AWS IoT Greengrass Core v1.10 e versioni successive.

Un core Greengrass può stabilire una sessione persistente con il broker di AWS IoT messaggi. Una sessione persistente è una connessione continua che consente al core di ricevere messaggi inviati mentre è offline. Il core è il client nella connessione.

In una sessione persistente, il broker di AWS IoT messaggi salva tutte le sottoscrizioni effettuate dal core durante la connessione. [Se il core si disconnette, il broker di AWS IoT messaggi archivia i messaggi nuovi e non riconosciuti pubblicati come QoS 1 e destinati a destinazioni locali, come le funzioni Lambda e i dispositivi client.](#) Quando il core si riconnette, la sessione persistente viene ripresa e il broker di messaggi invia i AWS IoT messaggi archiviati al core a una velocità massima di 10 messaggi al secondo. Le sessioni persistenti hanno un periodo di scadenza predefinito di 1 ora, che inizia quando il broker messaggi rileva che il core si disconnette. Per ulteriori informazioni, consulta [MQTT persistent sessions](#) nella Developer Guide.AWS IoT

AWS IoT Greengrass utilizza il componente del sistema spooler (la funzione `GGCloudSpooler` Lambda) per creare sottoscrizioni che hanno come origine. AWS IoT Puoi utilizzare la seguente variabile di ambiente `GGCloudSpooler` per configurare sessioni persistenti.

- `GG_CONFIG_SUBSCRIPTION_QUALITY`. La qualità degli abbonamenti che hanno come fonte. AWS IoT I seguenti valori sono validi:
 - `AtMostOnce` (impostazione predefinita). Disabilita sessioni persistenti. Le sottoscrizioni utilizzano QoS 0.
 - `AtLeastOncePersistent`. Abilita sessioni persistenti. Imposta il flag `cleanSession` su `0` nei messaggi `CONNECT` ed effettua la sottoscrizione con QoS 1.

I messaggi pubblicati con QoS 1 ricevuti dal core raggiungono sicuramente la coda di lavoro in memoria del daemon Greengrass. Il core riconosce il messaggio dopo che è stato aggiunto alla coda. La comunicazione successiva dalla coda alla destinazione locale (ad esempio, funzione, connettore o dispositivo Greengrass Lambda) viene inviata come QoS 0. AWS IoT Greengrass non garantisce la consegna a destinazioni locali.

Note

È possibile utilizzare la proprietà di configurazione [maxWorkItemCount](#) per controllare la dimensione della coda degli elementi di lavoro. Ad esempio, puoi aumentare le dimensioni della coda se il carico di lavoro richiede un intenso traffico MQTT.

Quando le sessioni persistenti sono abilitate, il core apre almeno una connessione aggiuntiva con cui scambiare messaggi MQTT. AWS IoT Per ulteriori informazioni, consulta [the section called “Client IDs per connessioni MQTT con AWS IoT”](#).

Per configurare sessioni persistenti MQTT

È possibile configurare l'utilizzo AWS IoT Greengrass di sessioni persistenti con AWS IoT Core. A tale scopo, distribuisce una versione di definizione della funzione in cui la funzione `GGCloudSpooler` imposta la qualità della sottoscrizione su `AtLeastOncePersistent`. Questa impostazione si applica a tutte le sottoscrizioni che hanno AWS IoT Core (`cloud`) come origine. È necessario utilizzare l'AWS IoT Greengrass API per configurare le sessioni persistenti. Questa operazione può essere eseguita nella console.

La procedura seguente utilizza il comando [create-function-definition-version](#) CLI per configurare lo spooler per l'utilizzo di sessioni persistenti. In questa procedura si presuppone che si stia aggiornando la configurazione dell'ultima versione di un gruppo esistente.

1. Ottieni il gruppo Greengrass IDs di destinazione e la versione del gruppo. Questa procedura presuppone che questa sia la versione più recente del gruppo e del gruppo. La seguente query restituisce il gruppo creato più di recente.

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

In alternativa, puoi eseguire query in base al nome. I nomi dei gruppi non devono essere univoci, pertanto potrebbero essere restituiti più gruppi.

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

Note

Puoi trovare questi valori anche nella AWS IoT console. L'ID gruppo viene visualizzato nella pagina Settings (Impostazioni) del gruppo. IDs Le versioni del gruppo vengono visualizzate nella scheda Distribuzioni del gruppo.

2. Copiare i valori Id e LatestVersion dal gruppo target nell'output.
3. Ottieni la versione gruppo più recente.
 - Sostituisci l'*group-id* con l'Id copiato.
 - Sostituisci l'*latest-group-version-id* con l'LatestVersion copiato.

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

4. Dall'Definitionoggetto nell'output, copia il CoreDefinitionVersionArn e il ARNs di tutti gli altri componenti del gruppo tranne. FunctionDefinitionVersionArn Si usano questi valori quando si crea una nuova versione gruppo.

5. Da `FunctionDefinitionVersionArn` nell'output, copia l'ID della definizione della funzione. L'ID è il GUID che segue il segmento `functions` nell'ARN, come mostrato nell'esempio seguente.

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

Note

In alternativa, è possibile creare una definizione di funzione eseguendo il [create-function-definition](#) comando e quindi copiando l'ID dall'output.

6. Aggiungi una versione di definizione della funzione alla definizione della funzione.
 - *function-definition-id* Sostituiscilo con Id quello che hai copiato per la definizione della funzione.
 - Sostituire *arbitrary-function-id* con un nome per la funzione, ad esempio **spooler-function**.
 - Aggiungi all'array tutte le funzioni Lambda che desideri includere in questa versione. `functions` È possibile utilizzare il [get-function-definition-version](#) comando per ottenere le funzioni Greengrass Lambda da una versione di definizione di funzione esistente.

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

Note

Se in precedenza hai impostato le variabili di ambiente `GG_CONFIG_STORAGE_TYPE` o `GG_CONFIG_MAX_SIZE_BYTES` per [definire le impostazioni di storage](#), includerle in questa istanza di funzione.

7. Copia l'Arn della definizione di funzione dall'output.
8. Crea una versione di gruppo che contenga la funzione Lambda del sistema.
 - Sostituisci *group-id* con Id for the group.
 - Sostituiscilo *core-definition-version-arn* con CoreDefinitionVersionArn quello che hai copiato dalla versione più recente del gruppo.
 - Sostituiscilo *function-definition-version-arn* con Arn quello che hai copiato per la nuova versione della definizione di funzione.
 - Sostituisci ARNs i componenti degli altri gruppi (ad esempio, SubscriptionDefinitionVersionArn o DeviceDefinitionVersionArn) che hai copiato dalla versione più recente del gruppo.
 - Rimuovi eventuali parametri inutilizzati. Ad esempio, rimuovi `--resource-definition-version-arn` se la versione gruppo non contiene risorse.

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

9. Copia il valore Version dall'output. Questo è l'ID della nuova versione gruppo.
10. Distribuisci il gruppo con la nuova versione del gruppo.
 - Sostituiscilo *group-id* con Id quello che hai copiato per il gruppo.
 - Sostituisci *group-version-id* con Version quello che hai copiato per la nuova versione del gruppo.

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

11. (Facoltativo) Aumentate la proprietà [maxWorkItemCount](#) nel file di configurazione principale. Questo consente al core di gestire l'aumento del traffico MQTT e la comunicazione con i target locali.

Per aggiornare il core con queste modifiche alla configurazione, utilizzate l' AWS IoT Greengrass API per creare una nuova versione della definizione della funzione che contenga la `GGCloudSpooler` funzione con la configurazione aggiornata. Quindi aggiungi la versione della definizione della funzione a una nuova versione del gruppo (insieme agli altri componenti del gruppo) e distribuisce la versione del gruppo. Se desideri ripristinare la configurazione predefinita, puoi creare una versione di definizione della funzione che non include la funzione `GGCloudSpooler`.

Questa funzione Lambda di sistema non è visibile nella console. Tuttavia, dopo che la funzione è stata aggiunta all'ultima versione del gruppo, è inclusa nelle distribuzioni eseguite dalla console, a meno che non usi l'API per sostituirla o rimuoverla.

Client IDs per connessioni MQTT con AWS IoT

Questa funzionalità è disponibile per AWS IoT Greengrass Core v1.8 e versioni successive.

Il core Greengrass apre le connessioni MQTT AWS IoT Core per operazioni come la sincronizzazione degli shadow e la gestione dei certificati. Per queste connessioni, il core genera un client prevedibile in IDs base al nome dell'oggetto principale. [Il client prevedibile IDs può essere utilizzato con funzionalità di monitoraggio, controllo e determinazione dei prezzi, inclusi AWS IoT Device Defender gli eventi del ciclo di vita.AWS IoT](#) È inoltre possibile creare una logica attorno a un client prevedibile IDs (ad esempio, modelli di [policy di sottoscrizione](#) basati sugli attributi del certificato).

GGC v1.9 and later

Due componenti del sistema Greengrass aprono connessioni MQTT con. AWS IoT Core Questi componenti utilizzano i seguenti modelli per generare il client IDs per le connessioni.

Operazione	Modello ID client
Distribuzioni	<i>core-thing-name</i> Esempio: MyCoreThing Utilizza questo ID client per connetterti, scollegarti, sottoscrivere e annullare la

Operazione	Modello ID client
	sottoscrizione delle notifiche di eventi del ciclo di vita.
Sottoscrizioni	<i>core-thing-name</i> -<i>cn</i> Esempio: MyCoreThing-c01 <i>n</i> è un numero intero che inizia da 00 e aumenta con ogni nuova connessione fino a un numero massimo di 250. Il numero di connessioni è determinato dal numero di dispositivi con cui sincronizzano il proprio stato ombra AWS IoT Core (massimo 2.500 per gruppo) e dal numero di abbonamenti che hanno c1oud come origine nel gruppo (massimo 10.000 per gruppo). Il componente del sistema spooler si connette AWS IoT Core per scambiare messaggi per abbonamenti con una sorgente o una destinazione cloud. Lo spooler funge anche da proxy per lo scambio di messaggi tra il servizio shadow locale AWS IoT Core e il gestore dei certificati del dispositivo.

Per calcolare il numero di connessioni MQTT per gruppo, utilizzate la seguente formula:

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

Dove,

- numero di connessioni per Deployment Agent = 1.
- numero di connessioni per gli abbonamenti =(2 subscriptions for supporting certificate generation + number of MQTT topics in AWS IoT Core + number of device shadows synced) / 50.

- Dove, 50 = il numero massimo di abbonamenti per connessione che AWS IoT Core può supportare.

Note

Se abiliti [le sessioni persistenti](#) per l'abbonamento con AWS IoT Core, il core apre almeno una connessione aggiuntiva da utilizzare in una sessione persistente. I componenti di sistema non supportano sessioni persistenti, quindi non possono condividere tale connessione.

Per ridurre il numero di connessioni MQTT e contribuire a ridurre i costi, puoi utilizzare le funzioni Lambda locali per aggregare i dati sull'edge. Quindi invii i dati aggregati a Cloud AWS. Di conseguenza, si utilizzano meno argomenti MQTT in AWS IoT Core. Per ulteriori informazioni, consulta la sezione [Prezzi di AWS IoT Greengrass](#).

GGC v1.8

Alcuni componenti di sistema Greengrass aprono le connessioni MQTT con AWS IoT Core. Questi componenti utilizzano i seguenti modelli per generare il client ID per le connessioni.

Operazione	Modello ID client
Distribuzioni	<i>core-thing-name</i> Esempio: MyCoreThing Utilizza questo ID client per connetterti, scollegarti, sottoscrivere e annullare la sottoscrizione delle notifiche di eventi del ciclo di vita.
Scambio di messaggi MQTT con AWS IoT Core	<i>core-thing-name</i> -spr Esempio: MyCoreThing-spr
Sincronizzazione shadow	<i>core-thing-name</i> -snn Esempio: MyCoreThing-s01

Operazione	Modello ID client
	<i>nn</i> è un intero che inizia da 00 e incrementa a ogni nuova connessione fino a un massimo di 03. Il numero di connessioni è determinato dal numero di dispositivi (massimo 200 dispositivi per gruppo) con cui sincronizzano il proprio stato ombra AWS IoT Core (massimo 50 abbonamenti per connessione).
Gestione dei certificati del dispositivo	<i>core-thing-name</i> -dcm Esempio: MyCoreThing-dcm

Note

Il client duplicato IDs utilizzato nelle connessioni simultanee può causare un ciclo infinito di connessione-disconnessione. Ciò può accadere se un altro dispositivo è hardcoded per utilizzare il nome del dispositivo core come ID client nelle connessioni. Per ulteriori informazioni, consulta questa [fase della risoluzione dei problemi](#).

I dispositivi Greengrass sono anche completamente integrati con il servizio di indicizzazione del parco istanze di AWS IoT Device Management. In questo modo è possibile indicizzare e cercare dispositivi basati su attributi dei dispositivi, stato ombra e stato della connessione nel cloud. Ad esempio, i dispositivi Greengrass stabiliscono almeno una connessione che utilizza il nome dell'oggetto come ID del client, perciò è possibile utilizzare l'indicizzazione della connettività del dispositivo per rilevare quali dispositivi Greengrass sono attualmente collegati a AWS IoT Core o scollegati da esso. Per ulteriori informazioni, consulta il servizio di [indicizzazione del parco veicoli nella Guida](#) per gli sviluppatori.AWS IoT

Configurazione della porta MQTT per la messaggistica locale

Questa funzionalità richiede AWS IoT Greengrass Core v1.10 o versione successiva.

[Il core Greengrass funge da broker di messaggi locale per la messaggistica MQTT tra funzioni Lambda locali, connettori e dispositivi client.](#) Per impostazione predefinita, il core utilizza la porta

8883 per il traffico MQTT sulla rete locale. Potrebbe essere opportuno modificare la porta per evitare un conflitto con altro software in esecuzione sulla porta 8883.

Per configurare il numero di porta utilizzato dal core per il traffico MQTT locale

1. Eseguite il seguente comando per arrestare il demone Greengrass:

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

2. Apri **greengrass-root**/config/config.json per la modifica come utente su.
3. Nell'oggetto coreThing, aggiungere la proprietà ggMqttPort e impostare il valore sul numero di porta che si desidera utilizzare. I valori validi sono compresi tra 1024 e 65535. Nell'esempio seguente il numero di porta viene impostato su 9000.

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

4. Avvia il daemon.

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

5. Se il [rilevamento IP automatico](#) è abilitato per il core, la configurazione è completata.

Se il rilevamento IP automatico non è abilitato, è necessario aggiornare le informazioni di connettività per il core. Ciò consente ai dispositivi client di ricevere il numero di porta corretto durante le operazioni di rilevamento per acquisire le informazioni di connettività di base. È possibile utilizzare la AWS IoT console o l' AWS IoT Greengrass API per aggiornare le

informazioni di connettività di base. Per questa procedura, aggiornare solo il numero di porta. L'indirizzo IP locale per il core non cambia.

Per aggiornare le informazioni di connettività per il core (console)

1. Nella pagina di configurazione del gruppo, scegli il core Greengrass.
2. Nella pagina dei dettagli principali, scegli la scheda Endpoints del broker MQTT.
3. Scegli Gestisci endpoint, quindi scegli Aggiungi endpoint
4. Inserisci il tuo attuale indirizzo IP locale e il nuovo numero di porta. Nell'esempio seguente il numero di porta viene impostato su 9000 per l'indirizzo IP 192.168.1.8.
5. Rimuovere l'endpoint obsoleto e quindi scegliere Update (Aggiorna)

Per aggiornare le informazioni di connettività per il core (API)

- Usa l'azione [UpdateConnectivityInfo](#). Nell'esempio seguente viene utilizzato `update-connectivity-info` nella AWS CLI per impostare il numero di porta 9000 per l'indirizzo IP 192.168.1.8.

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

Note

Puoi anche configurare la porta utilizzata dal core per la messaggistica MQTT. AWS IoT Core Per ulteriori informazioni, consulta [the section called “Connessione alla porta 443 o tramite un proxy di rete”](#).

Timeout per le operazioni di pubblicazione, sottoscrizione e annullamento dell'iscrizione nelle connessioni MQTT con Cloud AWS

Questa funzionalità è disponibile nella AWS IoT Greengrass versione 1.10.2 o successiva.

È possibile configurare la quantità di tempo (in secondi) per consentire al core Greengrass di completare un'operazione di pubblicazione, sottoscrizione o annullamento della sottoscrizione nelle connessioni MQTT a AWS IoT Core. È possibile modificare questa impostazione se le operazioni vanno in timeout a causa di vincoli di larghezza di banda o latenza elevata. Per configurare questa impostazione nel file [config.json](#) aggiungere o modificare la proprietà `mqttOperationTimeout` nell'oggetto `coreThing`. Per esempio:

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

Il timeout predefinito è 5 secondi. Il timeout minimo è di 5 secondi.

Attivazione del rilevamento automatico dell'IP

È possibile configurare AWS IoT Greengrass per consentire ai dispositivi client di un gruppo Greengrass di scoprire automaticamente il core Greengrass. Se abilitato, il core controlla le modifiche ai suoi indirizzi IP. Se un indirizzo cambia, il core pubblica un elenco aggiornato di indirizzi. Questi indirizzi sono resi disponibili ai dispositivi client che fanno parte dello stesso gruppo Greengrass del core.

Note

La AWS IoT politica per i dispositivi client deve concedere `greengrass:Discover` autorizzazione per consentire ai dispositivi di recuperare le informazioni di connettività per il core. Per ulteriori informazioni sull'istruzione di policy, consulta [the section called “Autorizzazione alla scoperta”](#).

Per abilitare questa funzionalità dalla AWS IoT Greengrass console, scegli Rilevamento automatico quando distribuisce il tuo gruppo Greengrass per la prima volta. Puoi anche abilitare o disabilitare questa funzionalità nella pagina di configurazione del gruppo scegliendo la scheda Funzioni Lambda

e selezionando il rilevatore IP. Il rilevamento automatico degli IP è abilitato se è selezionata l'opzione Rileva automaticamente e sostituisci gli endpoint del broker MQTT.

Per gestire il rilevamento automatico con l' AWS IoT Greengrass API, è necessario configurare la funzione Lambda IPDetector del sistema. La procedura seguente mostra come utilizzare il comando [create-function-definition-version](#) CLI per configurare il rilevamento automatico del core Greengrass.

1. Ottieni il gruppo Greengrass IDs di destinazione e la versione del gruppo. Questa procedura presuppone che questa sia la versione più recente del gruppo e del gruppo. La seguente query restituisce il gruppo creato più di recente.

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

In alternativa, puoi eseguire query in base al nome. I nomi dei gruppi non devono essere univoci, pertanto potrebbero essere restituiti più gruppi.

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

Note

Puoi trovare questi valori anche nella AWS IoT console. L'ID gruppo viene visualizzato nella pagina Settings (Impostazioni) del gruppo. IDs Le versioni del gruppo vengono visualizzate nella scheda Distribuzioni del gruppo.

2. Copiare i valori Id e LatestVersion dal gruppo target nell'output.
3. Ottieni la versione gruppo più recente.
 - Sostituisci l'*group-id* con l'Id copiato.
 - Sostituisci l'*latest-group-version-id* con l'LatestVersion copiato.

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

4. Dall'Definitionoggetto nell'output, copia il CoreDefinitionVersionArn e il ARNs di tutti gli altri componenti del gruppo tranne. FunctionDefinitionVersionArn Si usano questi valori quando si crea una nuova versione gruppo.
5. Da FunctionDefinitionVersionArn nell'output, copia l'ID della definizione della funzione e la versione della definizione della funzione:

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

Note

È anche possibile creare una definizione di funzione eseguendo il comando [create-function-definition](#) e quindi copiare l'ID dall'output.

6. Usa il comando [get-function-definition-version](#) per ottenere lo stato della definizione corrente. Utilizzate quelli *function-definition-id* che avete copiato per la definizione della funzione. Ad esempio *4d941bc7-92a1-4f45-8d64-EXAMPLEf76c3*.

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

Prendi nota delle configurazioni della funzione elencate. Dovrai includerle al momento della creazione di una nuova versione della definizione della funzione per evitare di perdere le impostazioni della definizione corrente.

7. Aggiungi una versione di definizione della funzione alla definizione della funzione.
 - Sostituisci *function-definition-id* con quello Id che hai copiato per la definizione della funzione. Ad esempio *4d941bc7-92a1-4f45-8d64-EXAMPLEf76c3*.
 - Sostituire *arbitrary-function-id* con un nome per la funzione, ad esempio **auto-detection-function**.
 - Aggiungi all'functionsarray tutte le funzioni Lambda che desideri includere in questa versione, come quelle elencate nel passaggio precedente.

```
aws greengrass create-function-definition-version \
```

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

8. Copia l'Arn della definizione di funzione dall'output.
9. Crea una versione di gruppo che contenga la funzione Lambda del sistema.
 - Sostituisci *group-id* con Id for the group.
 - Sostituiscilo *core-definition-version-arn* con CoreDefinitionVersionArn quello che hai copiato dalla versione più recente del gruppo.
 - Sostituiscilo *function-definition-version-arn* con Arn quello che hai copiato per la nuova versione della definizione di funzione.
 - Sostituisci ARNs i componenti degli altri gruppi (ad esempio, SubscriptionDefinitionVersionArn o DeviceDefinitionVersionArn) che hai copiato dalla versione più recente del gruppo.
 - Rimuovi eventuali parametri inutilizzati. Ad esempio, rimuovi `--resource-definition-version-arn` se la versione gruppo non contiene risorse.

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

10. Copia il valore Version dall'output. Questo è l'ID della nuova versione gruppo.
11. Distribuisci il gruppo con la nuova versione del gruppo.
 - Sostituiscilo *group-id* con Id quello che hai copiato per il gruppo.
 - Sostituisci *group-version-id* con Version quello che hai copiato per la nuova versione del gruppo.

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

Se vuoi immettere manualmente l'indirizzo IP del core Greengrass, puoi completare questo tutorial con un'altra definizione della funzione che non includa la funzione `IPDetector`. Ciò impedirà alla funzione di rilevamento di localizzare e inserire automaticamente l'indirizzo IP principale di Greengrass.

Questa funzione Lambda di sistema non è visibile nella console Lambda. Una volta aggiunta all'ultima versione del gruppo, la funzione viene inclusa nelle distribuzioni eseguite dalla console, a meno che non usi l'API per sostituirla o rimuoverla.

Configurazione del sistema di inizializzazione per avviare il daemon Greengrass

È buona norma impostare il sistema di inizializzazione in modo tale che il lancio del daemon Greengrass avvenga all'avvio, soprattutto in caso di gestione di grandi gruppi di dispositivi.

Note

Se in passato apt installavi il software AWS IoT Greengrass Core, puoi usare gli script `systemd` per abilitare l'avvio all'avvio. Per ulteriori informazioni, consulta [the section called “Utilizzare gli script systemd per gestire il ciclo di vita del daemon Greengrass”](#).

Esistono diversi tipi di sistema di inizializzazione, ad esempio `initd`, `systemd` e `SystemV`, i quali utilizzano parametri di configurazione simili. L'esempio seguente è un file di servizio per `systemd`. Il parametro `Type` è impostato su `forking` perché `greengrassd` (che viene utilizzato per avviare Greengrass) biforca il processo del daemon Greengrass, mentre il parametro `Restart` è impostato su `on-failure` per indirizzare `systemd` in modo tale da riavviare Greengrass se Greengrass presenta uno stato di errore.

 Note

Per vedere se il dispositivo utilizza systemd, eseguire lo script `check_ggc_dependencies` come descritto nel [Modulo 1](#). Quindi, per utilizzare systemd fai in modo che il parametro `useSystemd` in [config.json](#) sia impostato su `yes`.

```
[Unit]
Description=Greengrass Daemon

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

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

Consulta anche

- [Che cos'è AWS IoT Greengrass?](#)
- [the section called “Piattaforme supportate e requisiti”](#)
- [Iniziare con AWS IoT Greengrass](#)
- [the section called “Panoramica del modello di oggetti del gruppo”](#)
- [the section called “Integrazione della sicurezza hardware”](#)

AWS IoT Greengrass Version 1 politica di manutenzione

Utilizza questa politica AWS IoT Greengrass V1 di manutenzione per comprendere i diversi livelli di manutenzione e aggiornamenti per il AWS IoT Greengrass V1 servizio e il software Core v1.x. AWS IoT Greengrass

Argomenti

- [AWS IoT Greengrass schema di controllo delle versioni](#)
- [Fasi del ciclo di vita delle versioni principali del software Core AWS IoT Greengrass](#)
- [Politica di manutenzione per AWS IoT Greengrass il software Core](#)
- [Pianificazione della rinuncia](#)
- [Politica di supporto per AWS Lambda le funzioni sui dispositivi core Greengrass](#)
- [Politica di supporto per AWS IoT Device Tester per AWS IoT Greengrass V1](#)
- [Fine del programma di manutenzione](#)

AWS IoT Greengrass schema di controllo delle versioni

AWS IoT Greengrass utilizza il controllo delle [versioni semantiche](#) per il AWS IoT Greengrass software Core. Le versioni semantiche seguono una delle principali. minore. sistema di numerazione delle patch. La versione principale aumenta per le modifiche funzionali e alle API che non sono retrocompatibili con le versioni principali precedenti. La versione secondaria aumenta per le versioni che aggiungono nuove funzionalità compatibili con le versioni precedenti. La versione della patch aumenta per le patch di sicurezza o le correzioni di bug. Dalla sua prima versione principale, la v1.0.0, AWS IoT Greengrass ha rilasciato 11 versioni minori del software AWS IoT Greengrass Core v1.x, di cui la v1.11.6 è l'ultima versione. Ti consigliamo di aggiornare il software AWS IoT Greengrass Core all'ultima versione disponibile per sfruttare nuove funzionalità, miglioramenti e correzioni di bug.

A dicembre 2020, AWS IoT Greengrass ha rilasciato il suo primo aggiornamento importante della versione. Questo aggiornamento includeva il AWS IoT Greengrass V2 servizio e la versione 2.0.3 del software AWS IoT Greengrass Core. Per le nuove applicazioni, si consiglia vivamente di utilizzare AWS IoT Greengrass Version 2 e il software AWS IoT Greengrass Core v2.x. La versione 2 riceve nuove funzionalità, include tutte le funzionalità chiave della V1 e supporta piattaforme aggiuntive e implementazioni continue su grandi flotte di dispositivi. [Per ulteriori informazioni, consulta Cos'è? AWS IoT Greengrass V2](#) .

Fasi del ciclo di vita delle versioni principali del software Core AWS IoT Greengrass

Ogni versione principale del software AWS IoT Greengrass Core prevede le seguenti tre fasi sequenziali del ciclo di vita. Ogni fase del ciclo di vita offre diversi livelli di manutenzione per un periodo di tempo successivo alla data di rilascio iniziale.

- Fase di rilascio: AWS IoT Greengrass potrebbe rilasciare i seguenti aggiornamenti:
 - Aggiornamenti di versione minori che forniscono nuove funzionalità o miglioramenti alle funzionalità esistenti
 - Aggiornamenti delle versioni delle patch che forniscono patch di sicurezza e correzioni di bug
- Fase di manutenzione: AWS IoT Greengrass può rilasciare aggiornamenti della versione delle patch che forniscono patch di sicurezza e correzioni di bug. AWS IoT Greengrass non rilascerà nuove funzionalità o miglioramenti alle funzionalità esistenti durante la fase di manutenzione.
- Fase di vita estesa: AWS IoT Greengrass non rilascerà aggiornamenti che forniscano funzionalità, miglioramenti alle funzionalità esistenti, patch di sicurezza o correzioni di bug. [Tuttavia, gli Cloud AWS endpoint e le operazioni API rimarranno disponibili e funzioneranno in base al Service Level Agreement.](#) [AWS IoT Greengrass](#) I dispositivi che eseguono il software AWS IoT Greengrass Core v1.x possono continuare a connettersi e funzionare. Cloud AWS

Al termine della fase di vita prolungata di una versione principale di AWS IoT Greengrass, gli Cloud AWS endpoint e le operazioni delle API saranno obsoleti e non più disponibili. I dispositivi che eseguono il software AWS IoT Greengrass Core v1.x non saranno in grado di connettersi ai servizi per funzionare. Cloud AWS

Politica di manutenzione per AWS IoT Greengrass il software Core

Il software AWS IoT Greengrass Core v1.x è entrato nella fase di durata prolungata il 30 giugno 2023. Dopo questa data, il software AWS IoT Greengrass Core v1.x rimarrà nella fase di vita prolungata fino a nuovo avviso.

Il software AWS IoT Greengrass Core v2.x è attualmente in fase di rilascio e rimarrà in fase di rilascio fino a nuovo avviso. AWS IoT Greengrass continua ad aggiungere nuove funzionalità e miglioramenti al software Core v2.x. AWS IoT Greengrass Ad esempio, è AWS IoT Greengrass stato rilasciato il supporto per Windows nella versione 2.5.0 del software Core. AWS IoT Greengrass AWS IoT

Greengrass rilascia patch di sicurezza e correzioni di bug per tutte le versioni minori di AWS IoT Greengrass Core v2.x per almeno 1 anno dopo la data di rilascio. Per ulteriori informazioni, consulta [What's new in. AWS IoT Greengrass V2](#)

Pianificazione della fase di manutenzione

Il 30 giugno 2023, la fase di manutenzione del software AWS IoT Greengrass Core v1.11.x si è conclusa. Il 31 marzo 2022, la fase di manutenzione è terminata per il software AWS IoT Greengrass Core v1.10.x. La fase di manutenzione per alcuni artefatti e AWS IoT Greengrass funzionalità del software Core v1.x termina prima di queste date. Per ulteriori informazioni, consulta [Fine del programma di manutenzione](#).

Se hai un Supporto AWS piano, la fase di manutenzione del software AWS IoT Greengrass Core v1.x non influisce sul tuo piano. Supporto AWS Puoi continuare ad aprire Supporto AWS i ticket anche dopo la fine della fase di manutenzione. Se hai domande o dubbi, contatta il tuo Supporto AWS contatto o fai una domanda su [AWS Re:post](#) utilizzando il AWS IoT Greengrassstag.

Pianificazione della rinuncia

Al momento, non è previsto l'interruzione del supporto del software AWS IoT Greengrass Core v1.x. Gli AWS IoT Greengrass V1 endpoint e le operazioni API rimarranno disponibili fino a nuovo avviso. Il software AWS IoT Greengrass Core v1.11.6 è entrato nella fase di durata prolungata il 30 giugno 2023. Durante questa fase, i dispositivi che eseguono il software AWS IoT Greengrass Core v1.x possono continuare a connettersi al AWS IoT Greengrass V1 servizio per funzionare fino a nuovo avviso.

Se AWS IoT Greengrass V1 smetterà di essere supportato in futuro, AWS IoT Greengrass fornirà un preavviso di 12 mesi prima che ciò accada. Questo vi aiuterà a pianificare l'aggiornamento delle applicazioni da utilizzare AWS IoT Greengrass V2 e del software AWS IoT Greengrass Core v2.x. Per ulteriori informazioni su come aggiornare le applicazioni alla V2, consulta [Passa dalla AWS IoT Greengrass V1](#) versione 2 alla versione 2.

Politica di supporto per AWS Lambda le funzioni sui dispositivi core Greengrass

AWS IoT Greengrass consente di eseguire AWS Lambda funzioni su dispositivi IoT. AWS Lambda fornisce una politica di supporto e delle tempistiche che determinano il supporto per i runtime Lambda

in. AWS IoT Greengrass Quando un runtime Lambda raggiunge la fine della fase di supporto, termina AWS IoT Greengrass anche il supporto per quel runtime. Per ulteriori informazioni, consulta la [politica di supporto di Runtime](#) nella AWS Lambda Developer Guide.

Quando un runtime Lambda raggiunge la fine del supporto, non puoi creare o aggiornare funzioni Lambda che utilizzano quel runtime. Tuttavia, puoi continuare a distribuire queste funzioni Lambda sui dispositivi principali Greengrass e richiamare le funzioni Lambda distribuite. Questa politica si applica anche a. AWS IoT Greengrass V2

Politica di supporto per AWS IoT Device Tester per AWS IoT Greengrass V1

AWS IoT [Device Tester \(IDT\) AWS IoT Greengrass V1](#) consente di convalidare e qualificare i [AWS IoT Greengrass dispositivi per l'inclusione nel Device Catalog.AWS Partner](#) A partire dal 4 aprile 2022, AWS IoT Device Tester (IDT) AWS IoT Greengrass V1 non genera più rapporti di qualificazione firmati. [Non è più possibile qualificare nuovi AWS IoT Greengrass V1 dispositivi per inserirli nel catalogo dei dispositivi tramite il AWS Partner Device Qualification Program.AWS](#) Anche se non puoi qualificare i dispositivi Greengrass V1, puoi continuare a utilizzare IDT per AWS IoT Greengrass V1 testare i tuoi dispositivi Greengrass V1. [Ti consigliamo di utilizzare IDT per qualificare ed AWS IoT Greengrass V2 elencare i dispositivi Greengrass nel Device Catalog.AWS Partner](#) Per ulteriori informazioni, consulta [Politica di supporto per AWS IoT Device Tester per AWS IoT Greengrass V1](#).

Fine del programma di manutenzione

La tabella seguente elenca le date di fine della manutenzione per gli artefatti e le funzionalità di AWS IoT Greengrass Core v1.x. In caso di domande sul programma o sulla politica di manutenzione, contatta l'[AWS assistenza](#).

Artefatto o caratteristica	Data di fine della manutenzione
Installazione del repository APT Greengrass	11 febbraio 2022
Connettore ML Image Classification	31 marzo 2022
Connettore ML Object Detection	31 marzo 2022

Artefatto o caratteristica	Data di fine della manutenzione
Connettore ML Feedback	31 marzo 2022
AWS IoT Analytics connettore	31 marzo 2022
Connettore di notifica Twilio	31 marzo 2022
Connettore di integrazione Splunk	31 marzo 2022
Connettore Serial Stream	31 marzo 2022
ServiceNow MetricBase Connettore di integrazione	31 marzo 2022
Connettore GPIO Raspberry Pi	31 marzo 2022
AWS IoT Greengrass Software di base v1.10.x	31 marzo 2022
AWS IoT Greengrass Immagini Docker del software principale v1.x	30 giugno 2022
AWS IoT Greengrass Software di base v1.11.x	30 giugno 2023
AWS IoT Greengrass Software di base v1.11.x Snap	31 dicembre 2023

Fine della manutenzione per le immagini AWS IoT Greengrass Docker del software Core v1.x

Il 30 giugno 2022, AWS IoT Greengrass è terminata la manutenzione delle immagini Docker del software AWS IoT Greengrass Core v1.x pubblicate su Amazon Elastic Container Registry (Amazon ECR) e Docker Hub. Puoi continuare a scaricare queste immagini Docker da Amazon ECR e Docker Hub fino al 30 giugno 2023, ovvero 1 anno dopo la fine della manutenzione. Tuttavia, le immagini Docker del software AWS IoT Greengrass Core v1.x non ricevono più patch di sicurezza o correzioni di bug dopo la fine della manutenzione il 30 giugno 2022. Se esegui un carico di lavoro di produzione che dipende da queste immagini Docker, ti consigliamo di creare le tue immagini Docker utilizzando i Dockerfile forniti. AWS IoT Greengrass Per ulteriori informazioni, consulta [AWS IoT Greengrass Software Docker](#).

Fine della manutenzione del repository APT del software Core v1.x AWS IoT Greengrass

L'11 febbraio 2022, AWS IoT Greengrass è terminata la manutenzione dell'opzione di [installazione del software AWS IoT Greengrass Core v1.x](#) da un repository APT. Il repository APT è stato rimosso in questa data, quindi non è più possibile utilizzare il repository APT per aggiornare il software Core o installare il software AWS IoT Greengrass Core su nuovi dispositivi. AWS IoT Greengrass Sui dispositivi in cui è stato aggiunto il AWS IoT Greengrass repository, è necessario [rimuovere il repository dall'elenco delle fonti](#). [Ti consigliamo di aggiornare il software AWS IoT Greengrass Core v1.x utilizzando i file tar.](#)

Fine della manutenzione per il software AWS IoT Greengrass Core v1.11.x Snap

[Il 31 dicembre 2023, AWS IoT Greengrass terminerà la manutenzione per la versione del software AWS IoT Greengrass principale 1.11.x Snap pubblicata su \[snapcraft.io\]\(#\)](#). I dispositivi che attualmente eseguono Snap continueranno a funzionare fino a nuovo avviso. Tuttavia, lo Snap AWS IoT Greengrass principale non riceverà più patch di sicurezza o correzioni di bug al termine della manutenzione.

Guida introduttiva con AWS IoT Greengrass

Questo tutorial introduttivo include diversi moduli progettati per mostrarti le AWS IoT Greengrass nozioni di base e aiutarti a iniziare a utilizzare AWS IoT Greengrass. Questo tutorial copre concetti fondamentali, come ad esempio:

- Configurazione di AWS IoT Greengrass core e gruppi.
- Il processo di implementazione per l'esecuzione di AWS Lambda funzioni all'edge.
- Connessione di AWS IoT dispositivi, denominati dispositivi client, al AWS IoT Greengrass core.
- Creazione di abbonamenti per consentire la comunicazione MQTT tra funzioni Lambda locali, dispositivi client e. AWS IoT

Scegli come iniziare con AWS IoT Greengrass

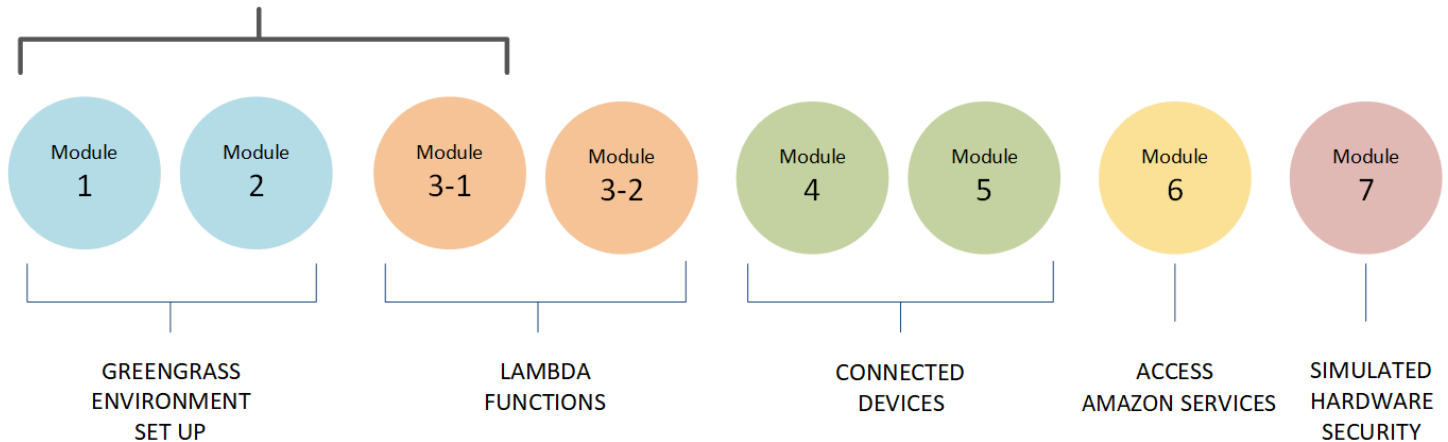
Puoi scegliere di usare questo tutorial per configurare il tuo dispositivo principale:

- Esegui la [configurazione del dispositivo Greengrass](#) sul tuo dispositivo principale, che ti porta dall'installazione AWS IoT Greengrass delle dipendenze al test di una funzione Hello World Lambda in pochi minuti. Questo script riproduce i passaggi nel modulo 1 attraverso il modulo 3-1.

oppure

- Passa attraverso i passaggi del modulo 1 attraverso il modulo 3-1 per esaminare più da vicino i requisiti e i processi di Greengrass. Questi passaggi consentono di configurare il dispositivo principale, creare e configurare un gruppo Greengrass contenente una funzione Hello World Lambda e distribuire il gruppo Greengrass. In genere, il completamento richiede un'ora o due.

Quick Start: Greengrass Device Setup



Quick Start

[La configurazione del dispositivo Greengrass](#) configura il dispositivo principale e le risorse Greengrass. Lo script:

- Installa le AWS IoT Greengrass dipendenze.
- Scarica il certificato CA principale, il certificato e le chiavi del dispositivo principale.
- Scarica, installa e configura il software AWS IoT Greengrass Core sul tuo dispositivo.
- Avvia il processo daemon di Greengrass sul dispositivo principale.
- Crea o aggiorna il [ruolo del servizio Greengrass](#), se necessario.
- Crea un gruppo Greengrass e un core Greengrass.
- (Facoltativo) Crea una funzione Hello World Lambda, un abbonamento e una configurazione di registrazione locale.
- (Facoltativo) Implementa il gruppo Greengrass.

Moduli 1 e 2

Il [modulo 1](#) e il [modulo 2](#) descrivono come configurare l'ambiente. (Oppure, utilizza la [configurazione del dispositivo Greengrass](#) per eseguire questi moduli.)

- Configura il tuo dispositivo principale per Greengrass.
- Esegui lo script di controllo delle dipendenze.
- Crea un gruppo Greengrass e un core Greengrass.
- Scaricate e installate il software AWS IoT Greengrass Core più recente da un file tar.gz.

- Avvia il processo daemon di Greengrass sul core.

 Note

AWS IoT Greengrass fornisce anche altre opzioni per l'installazione del software AWS IoT Greengrass Core, incluse le apt installazioni su piattaforme Debian supportate. Per ulteriori informazioni, consulta [the section called “Installare il software AWS IoT Greengrass Core”](#).

Moduli 3-1 e 3-2

[Il Modulo 3-1](#) e il [Modulo 3-2](#) descrivono come utilizzare le funzioni Lambda locali. (Oppure, utilizza la [configurazione del dispositivo Greengrass](#) per eseguire il modulo 3-1.)

- Crea funzioni Hello World Lambda in. AWS Lambda
- Aggiungi le funzioni Lambda al tuo gruppo Greengrass.
- Crea abbonamenti che consentano la comunicazione MQTT tra le funzioni Lambda e. AWS IoT
- Configura la registrazione locale per i componenti del sistema Greengrass e le funzioni Lambda.
- Implementa un gruppo Greengrass che contenga le funzioni e gli abbonamenti Lambda.
- Invia messaggi dalle funzioni Lambda locali a. AWS IoT
- Richiama funzioni Lambda locali da. AWS IoT
- Provare le funzioni on-demand e a lungo termine.

Moduli 4 e 5

[Il Modulo 4](#) mostra come i dispositivi client si connettono al core e comunicano tra loro.

[Il Modulo 5](#) mostra come i dispositivi client possono utilizzare le ombre per controllare lo stato.

- Registra ed esegui il provisioning AWS IoT dei dispositivi (rappresentati da terminali a riga di comando).
- Installa il SDK per dispositivi AWS IoT for Python. Viene utilizzato dai dispositivi client per scoprire il core Greengrass.
- Aggiungi i dispositivi client al tuo gruppo Greengrass.
- Creare sottoscrizioni che consentano la comunicazione MQTT.

- Implementa un gruppo Greengrass che contiene i tuoi dispositivi client.
- Verifica la comunicazione device-to-device.
- Test degli aggiornamenti dello stato shadow.

Modulo 6

Il [Modulo 6](#) mostra come le funzioni Lambda possono accedere a. Cloud AWS

- Crea un ruolo di gruppo Greengrass che consenta l'accesso alle risorse Amazon DynamoDB.
- Aggiungi una funzione Lambda al tuo gruppo Greengrass. Questa funzione utilizza l' AWS SDK per Python per interagire con DynamoDB.
- Creare sottoscrizioni che consentano la comunicazione MQTT.
- Verifica l'interazione con DynamoDB.

Modulo 7

Nel [Modulo 7](#) viene illustrato come configurare un modulo di sicurezza hardware (HSM) simulato per l'uso con un core Greengrass.

Important

Questo modulo avanzato è fornito solo per la sperimentazione e il test iniziale. Non è per uso di produzione di alcun tipo.

- Installare e configurare un HSM basato su software e una chiave privata.
- Configurare il core di Greengrass per utilizzare la sicurezza hardware.
- Test della sicurezza hardware

Requisiti

Per completare questo tutorial, è necessario quanto segue:

- Un MAC, un PC con Windows o un sistema di tipo UNIX.
- Un Account AWS. Se non lo hai, consultare [the section called “Crea un Account AWS”](#).
- L'uso di una AWS [regione](#) che supporta AWS IoT Greengrass. Per l'elenco delle regioni supportate per AWS IoT Greengrass, consulta gli [AWS endpoint e le quote](#) in. Riferimenti generali di AWS

 Note

Prendi nota del tuo Regione AWS e assicurati che venga utilizzato in modo coerente in questo tutorial. Se cambi il tuo Regione AWS durante il tutorial, potresti riscontrare problemi nel completare i passaggi.

- Un Raspberry Pi 4 Model B o Raspberry Pi 3 Model B/B+, con una scheda microSD da 8 GB o un'istanza Amazon. EC2 Dato che AWS IoT Greengrass è concepito idealmente per essere utilizzato con hardware fisico, consigliamo di utilizzare un Raspberry Pi.

 Note

Per determinare il modello del Raspberry Pi, esegui il comando seguente:

```
cat /proc/cpuinfo
```

Vicino alla fine dell'elenco, annota il valore dell'attributo `Revision`, quindi consulta la tabella [Il mio modello di Pi](#). Ad esempio, se il valore di `Revision` è `a02082`, la tabella indica che il Pi è un 3 modello B.

Per determinare l'architettura del Raspberry Pi, esegui il comando seguente:

```
uname -m
```

Per questo tutorial, il risultato deve essere maggiore o uguale a `armv7l`.

- Conoscenza di base di Python.

Sebbene questo tutorial sia destinato all'esecuzione AWS IoT Greengrass su un Raspberry Pi, supporta anche altre piattaforme. AWS IoT Greengrass Per ulteriori informazioni, consulta [the section called "Piattaforme supportate e requisiti"](#).

Crea un Account AWS

Se non ne hai uno Account AWS, segui questi passaggi per creare e attivare un Account AWS:

Registrati per un Account AWS

Se non ne hai uno Account AWS, completa i seguenti passaggi per crearne uno.

Per iscriverti a un Account AWS

1. Apri la <https://portal.aws.amazon.com/billing/registrazione>.
2. Segui le istruzioni online.

Parte della procedura di registrazione prevede la ricezione di una telefonata o di un messaggio di testo e l'immissione di un codice di verifica sulla tastiera del telefono.

Quando ti iscrivi a un Account AWS, Utente root dell'account AWS viene creato un. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWS nell'account. Come best practice di sicurezza, assegna l'accesso amministrativo a un utente e utilizza solo l'utente root per eseguire [attività che richiedono l'accesso di un utente root](#).

AWS ti invia un'email di conferma dopo il completamento della procedura di registrazione. In qualsiasi momento, puoi visualizzare l'attività corrente del tuo account e gestirlo accedendo a <https://aws.amazon.com/> e scegliendo Il mio account.

Crea un utente con accesso amministrativo

Dopo esserti registrato Account AWS, proteggi Utente root dell'account AWS AWS IAM Identity Center, abilita e crea un utente amministrativo in modo da non utilizzare l'utente root per le attività quotidiane.

Proteggi i tuoi Utente root dell'account AWS

1. Accedi [AWS Management Console](#) come proprietario dell'account scegliendo Utente root e inserendo il tuo indirizzo Account AWS email. Nella pagina successiva, inserisci la password.

Per informazioni sull'accesso utilizzando un utente root, consulta la pagina [Signing in as the root user](#) della Guida per l'utente di Accedi ad AWS .

2. Abilita l'autenticazione a più fattori (MFA) per l'utente root.

Per istruzioni, consulta [Abilitare un dispositivo MFA virtuale per l'utente Account AWS root \(console\)](#) nella Guida per l'utente IAM.

Crea un utente con accesso amministrativo

1. Abilita Centro identità IAM.

Per istruzioni, consulta [Abilitazione di AWS IAM Identity Center](#) nella Guida per l'utente di AWS IAM Identity Center .

2. In IAM Identity Center, assegna l'accesso amministrativo a un utente.

Per un tutorial sull'utilizzo di IAM Identity Center directory come fonte di identità, consulta [Configurare l'accesso utente con l'impostazione predefinita IAM Identity Center directory](#) nella Guida per l'AWS IAM Identity Center utente.

Accesso come utente amministratore

- Per accedere con l'utente IAM Identity Center, utilizza l'URL di accesso che è stato inviato al tuo indirizzo e-mail quando hai creato l'utente IAM Identity Center.

Per informazioni sull'accesso utilizzando un utente IAM Identity Center, consulta [AWS Accedere al portale di accesso](#) nella Guida per l'Accedi ad AWS utente.

Assegna l'accesso a ulteriori utenti

1. In IAM Identity Center, crea un set di autorizzazioni conforme alla best practice dell'applicazione di autorizzazioni con il privilegio minimo.

Segui le istruzioni riportate nella pagina [Creazione di un set di autorizzazioni](#) nella Guida per l'utente di AWS IAM Identity Center .

2. Assegna al gruppo prima gli utenti e poi l'accesso con autenticazione unica (Single Sign-On).

Per istruzioni, consulta [Aggiungere gruppi](#) nella Guida per l'utente di AWS IAM Identity Center .

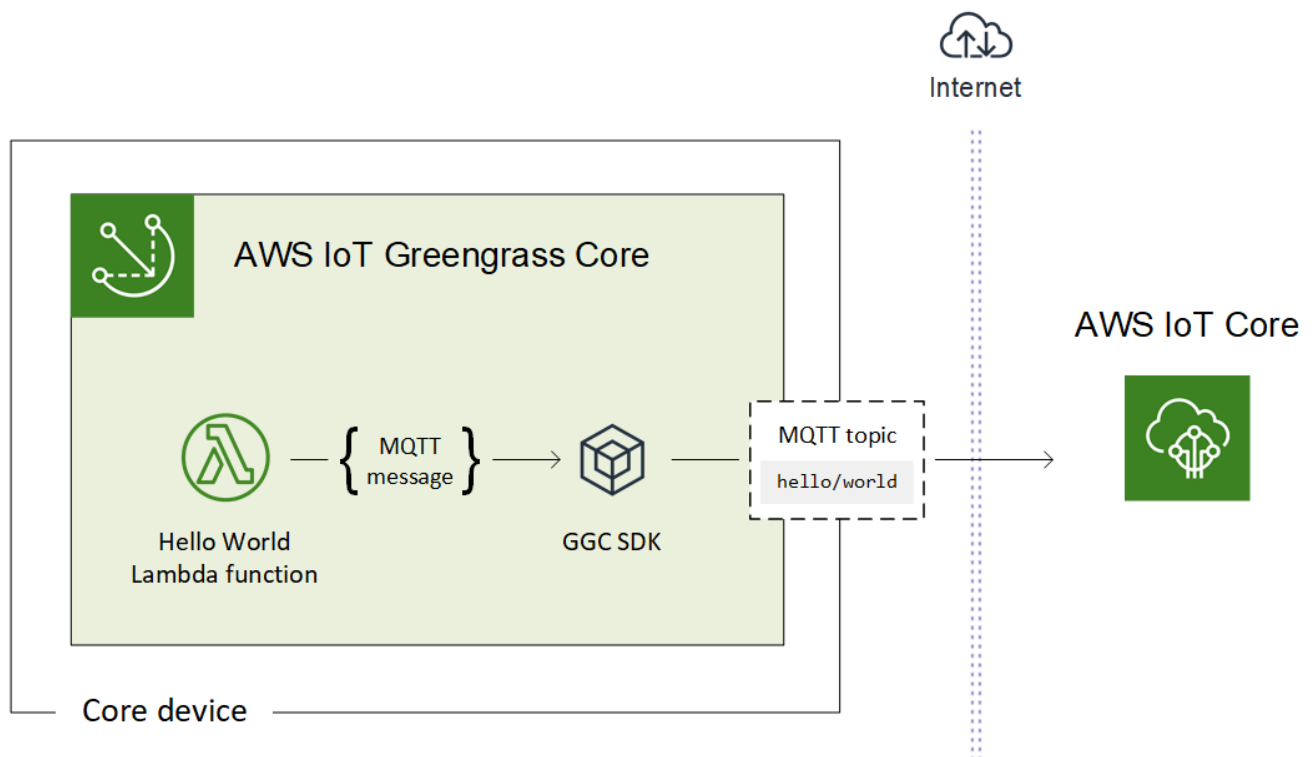
Important

Per questo tutorial, supponiamo che il tuo account utente IAM disponga dei permessi di accesso di amministratore.

Avvio rapido: configurazione dispositivo Greengrass

La configurazione del dispositivo Greengrass è uno script che configura il dispositivo principale in pochi minuti, in modo da poterlo iniziare a utilizzare. AWS IoT Greengrass Usa questo script per:

1. Configura il dispositivo e installa il software AWS IoT Greengrass Core.
2. Configura le tue risorse basate sul cloud.
3. Implementa facoltativamente un gruppo Greengrass con una funzione Hello World Lambda che invia messaggi MQTT dal core. AWS IoT Greengrass Questo imposta l'ambiente Greengrass mostrato nel diagramma seguente.



Requisiti

La configurazione del dispositivo Greengrass ha i seguenti requisiti:

- Il dispositivo principale deve utilizzare una [piattaforma supportata](#). Il dispositivo deve avere installato un gestore di pacchetti appropriato: apt, yum, o opkg.
- L'utente Linux che esegue lo script deve disporre delle autorizzazioni per eseguire come sudo.

- È necessario fornire le proprie credenziali. Account AWS Per ulteriori informazioni, consulta [the section called “Fornisci le credenziali Account AWS”](#).

Note

La configurazione del dispositivo Greengrass installa l'[ultima versione](#) del software AWS IoT Greengrass Core sul dispositivo. Installando il software AWS IoT Greengrass Core, l'utente accetta il contratto di [licenza del software Greengrass Core](#).

Esecuzione della configurazione del dispositivo Greengrass

Puoi eseguire la configurazione del dispositivo Greengrass in pochi passaggi. Dopo aver fornito le Account AWS credenziali, lo script esegue il provisioning del dispositivo principale Greengrass e distribuisce un gruppo Greengrass in pochi minuti. Esegui i seguenti comandi in una finestra terminale sul dispositivo di destinazione.

Note

Questi passaggi ti illustrano come eseguire lo script in modalità interattiva, che richiede di immettere o accettare ogni valore di input. Per informazioni su come eseguire lo script in modo invisibile, consulta [the section called “Esecuzione della configurazione del dispositivo Greengrass in modalità silenziosa”](#).

1. [Fornisci le tue credenziali](#). In questa procedura, si suppone che vengano fornite credenziali di sicurezza temporanee come variabili di ambiente.

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

 Note

Se stai eseguendo la configurazione del dispositivo Greengrass su una piattaforma Raspbian o su una OpenWrt piattaforma, crea una copia di questi comandi. Devi fornirli nuovamente dopo il riavvio del dispositivo.

2. Scarica e avvia lo script. Puoi utilizzare `wget` o `curl` scaricare lo script.

`wget`:

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

`curl`:

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

3. Procedi attraverso i prompt dei comandi per [i valori di input](#). Puoi premere il tasto Enter (Invio) per utilizzare il valore predefinito o digitare un valore personalizzato e quindi premere Enter (Invio).

Lo script scrive messaggi di stato sul terminale che sono simili ai seguenti.


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

[GreengrassDeviceSetup] Validating the device environment...
[GreengrassDeviceSetup] Validation of the device environment is complete.

[GreengrassDeviceSetup] Running the Greengrass environment setup...
[GreengrassDeviceSetup] The Greengrass environment setup is complete.

[GreengrassDeviceSetup] Configuring cloud-based Greengrass group management...
[GreengrassDeviceSetup] The Greengrass group configuration is complete.

[GreengrassDeviceSetup] Preparing the Greengrass core software...
[GreengrassDeviceSetup] The Greengrass core software is running.

[GreengrassDeviceSetup] Configuring the group deployment...
[GreengrassDeviceSetup] The group deployment is complete.
```

4. Se il tuo dispositivo principale esegue Raspbian oppure OpenWrt, riavvia il dispositivo quando richiesto, fornisci le tue credenziali e quindi riavvia lo script.

- a. Quando viene richiesto di riavviare il dispositivo, esegui uno dei seguenti comandi.

Per le piattaforme Raspbian:

```
sudo reboot
```

Per le piattaforme: OpenWrt

```
reboot
```

- b. Dopo il riavvio avvio del dispositivo, apri il terminale e fornisci le credenziali come variabili di ambiente.

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

- c. Riavvia lo script.

```
sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass-interactive
```

- d. Quando viene chiesto se utilizzare i valori di input della sessione precedente o avviare una nuova installazione, immetti yes per riutilizzare i valori di input.

Note

Sulle piattaforme che richiedono un riavvio, tutti i valori di input (ad eccezione delle credenziali) della sessione precedente vengono temporaneamente memorizzati nel file `GreengrassDeviceSetup.config.info`.

Al termine dell'installazione, il terminale visualizza un messaggio di stato di successo simile al seguente.

```
=====
Your device is running the Greengrass core software.
Your Greengrass group and Hello World Lambda function were deployed to the core device.

Setup information:

Device info: Linux-4.14.152-127.182.amzn2.x86_64-x86_64-with-glibc2.2.5
Greengrass core software location: /
Installed Greengrass core software version: 1.10.0
Greengrass core: arn:aws:iot:us-west-2:012345678910:thing/GreengrassDeviceSetup_Core_d46a0ea4-18ae-4376-8f44-4a504cdea608
Greengrass core IoT certificate: arn:aws:iot:us-west-2:012345678910:cert/23fbf0f4b6a5ea369f2b97f1a1b558180a240faa8e059ce19dc58f4a4c0d3b77
Greengrass core IoT certificate location: /greengrass/certs/23fbf0f4b6.cert.pem
Greengrass core IoT key location: /greengrass/certs/23fbf0f4b6.private.key
Deployed Greengrass group name: GreengrassDeviceSetup_Group_ee70f777-9af0-43b6-8612-a18b418e8b4a
Deployed Greengrass group ID: 6f5c8410-f3a6-43a2-acf3-33158e10fb8e
Deployed Greengrass group version: arn:aws:greengrass:us-west-2:012345678910:/greengrass/groups/6f5c8410-f3a6-43a2-acf3-33158e10fb8e/vers
Greengrass service role: arn:aws:iam::012345678910:role/GreengrassServiceRole_mui1v
GreengrassDeviceSetup log location: GreengrassDeviceSetup-20191209-232356.log
Deployed hello-world Lambda function: arn:aws:lambda:us-west-2:012345678910:function:Greengrass_HelloWorld_uNTf2:1
Hello-world subscriber topic: hello/world

You can now use the AWS IoT Console to subscribe
to the 'hello/world' topic to receive messages published from your
Greengrass core.
=====
```

5. Esamina il nuovo gruppo Greengrass che lo script configura utilizzando i valori di input forniti.
 - a. Accedi a [AWS Management Consoles](#) sul tuo computer e apri la AWS IoT console.

Note

Assicurati che quello Regione AWS selezionato nella console sia lo stesso che hai usato per configurare l'ambiente Greengrass. Per impostazione predefinita, la regione è Stati Uniti occidentali (Oregon).

- b. Nel pannello di navigazione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1) per individuare il gruppo appena creato.
6. Se hai incluso la funzione Hello World Lambda, la configurazione del dispositivo Greengrass distribuisce il gruppo Greengrass sul tuo dispositivo principale. Per testare la funzione Lambda o

per informazioni su come rimuovere la funzione Lambda dal gruppo, continua con il Modulo 3-1 del tutorial [the section called “Verifica che la funzione Lambda sia in esecuzione sul dispositivo principale”](#) Getting Started.

Note

Assicurati che quella Regione AWS selezionato nella console sia lo stesso che hai usato per configurare l'ambiente Greengrass. Per impostazione predefinita, la regione è Stati Uniti occidentali (Oregon).

Se non hai incluso la funzione Hello World Lambda, puoi [creare la tua funzione Lambda o provare altre funzionalità](#) di Greengrass. Ad esempio, puoi aggiungere il connettore di [distribuzione dell'applicazione Docker](#) al gruppo e utilizzarlo per distribuire contenitori Docker nel dispositivo principale.

Risoluzione dei problemi

È possibile utilizzare le seguenti informazioni per risolvere i problemi relativi alla configurazione del dispositivo. AWS IoT Greengrass

Errore: Python (python3.7) non trovato. Tentativo di installazione in corso...

Soluzione: potresti visualizzare questo errore quando lavori con un' EC2 istanza Amazon. Questo errore si verifica quando Python non è installato nella `/usr/bin/python3.7` cartella. Per risolvere questo errore, sposta Python nella directory corretta dopo averlo installato:

```
sudo ln -s /usr/local/bin/python3.7 /usr/bin/python3.7
```

Risoluzione di problemi aggiuntivi

Per risolvere altri problemi relativi alla configurazione del AWS IoT Greengrass dispositivo, puoi cercare le informazioni di debug nei file di registro:

- Per problemi con la configurazione del dispositivo Greengrass, controlla il file `/tmp/greengrass-device-setup-bootstrap-epoch-timestamp.log`.

- Per problemi con il gruppo Greengrass o l'impostazione dell'ambiente principale, controlla il file `GreengrassDeviceSetup-date-time.log` nella stessa directory `gg-device-setup-latest.sh` o nella posizione specificata.

Per ulteriori informazioni sulla risoluzione dei problemi, consulta [Risoluzione dei problemi](#) o controlla il [AWS IoT Greengrass tag](#) su re:POST. AWS

Opzioni di configurazione del dispositivo Greengrass

È possibile configurare la configurazione del dispositivo Greengrass per accedere alle AWS risorse e configurare l'ambiente Greengrass.

Fornisci le credenziali Account AWS

La configurazione del dispositivo Greengrass utilizza Account AWS le tue credenziali per accedere alle tue risorse. AWS Supporta credenziali a lungo termine per un utente IAM o credenziali di sicurezza temporanee da un ruolo IAM.

Per prima cosa, devi ottenere le credenziali

- Per utilizzare credenziali a lungo termine, fornisci l'ID della chiave di accesso e la chiave di accesso segreta per il tuo utente IAM. Per informazioni sulla creazione di chiavi di accesso per credenziali a lungo termine, consulta [Managing access keys for IAM users](#) nella IAM User Guide.
- Per utilizzare credenziali di sicurezza temporanee (consigliato), fornisci l'ID della chiave di accesso, la chiave di accesso segreta e il token di sessione relativi a un ruolo IAM presunto. Per informazioni sull'estrazione di credenziali di sicurezza temporanee dal AWS STS `assume-role` comando, consulta [Using temporary security credenziali with the AWS CLI](#) nella IAM User Guide.

Note

Ai fini di questo tutorial, si presuppone che l'utente o il ruolo IAM disponga dei permessi di accesso di amministratore.

Quindi, dovrai fornire tali credenziali per la configurazione del dispositivo Greengrass in uno dei due modi seguenti:

- Come variabile d'ambiente. Imposta le variabili d'ambiente `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, e `AWS_SESSION_TOKEN` (se necessario) prima di avviare lo script, come illustrato al passaggio 1 di [the section called “Esecuzione della configurazione del dispositivo Greengrass”](#).
- Come valori di input. Immetti l'ID della chiave di accesso, la chiave di accesso segreta e i valori del token di sessione (se necessario) direttamente nel terminale dopo l'avvio dello script.

La configurazione del dispositivo Greengrass non salva o memorizza le credenziali.

Fornire i valori di input

In modalità interattiva, la configurazione del dispositivo Greengrass richiede i valori di input. Puoi premere il tasto Enter (Invio) per utilizzare il valore predefinito o digitare un valore personalizzato e quindi premere Enter (Invio). In modalità silenziosa, fornisci i valori di input dopo l'avvio dello script.

Valori di input

AWS ID chiave di accesso

L'ID della chiave di accesso dalle credenziali di sicurezza a lungo termine o temporanee.

Specifica questa opzione come valore di input solo se non fornisci le credenziali come variabili di ambiente. Per ulteriori informazioni, consulta [the section called “Fornisci le credenziali Account AWS”](#).

Nome di opzione per la modalità silenziosa: `--aws-access-key-id`

AWS chiave di accesso segreta

Chiave di accesso segreta dalle credenziali di sicurezza a lungo termine o temporanee. Specifica questa opzione come valore di input solo se non fornisci le credenziali come variabili di ambiente. Per ulteriori informazioni, consulta [the section called “Fornisci le credenziali Account AWS”](#).

Nome di opzione per la modalità silenziosa: `--aws-secret-access-key`

AWS token di sessione

Il token di sessione dalle credenziali di sicurezza temporanee. Specifica questa opzione come valore di input solo se non fornisci le credenziali come variabili di ambiente. Per ulteriori informazioni, consulta [the section called “Fornisci le credenziali Account AWS”](#).

Nome di opzione per la modalità silenziosa: `--aws-session-token`

Regione AWS

Regione AWS Dove vuoi creare il gruppo Greengrass. Per l'elenco dei servizi Regione AWS supportati, vedere [AWS IoT Greengrass](#). Riferimenti generali di Amazon Web Services

Valore predefinito: `us-west-2`

Nome di opzione per la modalità silenziosa: `--region`

Group name (Nome gruppo)

Il nome del gruppo Greengrass.

Valore predefinito: `GreengrassDeviceSetup_Group_`*guid*

Nome di opzione per la modalità silenziosa: `--group-name`

Nome principale

Il nome per il core di Greengrass. Il core è un AWS IoT dispositivo (cosa) che esegue il software AWS IoT Greengrass Core. Il core viene aggiunto al AWS IoT registro e al gruppo Greengrass. Se fornite un nome, questo deve essere univoco alla lettera Account AWS e Regione AWS.

Valore predefinito: `GreengrassDeviceSetup_Core_`*guid*

Nome di opzione per la modalità silenziosa: `--core-name`

AWS IoT Greengrass Percorso di installazione del software principale

La posizione nel file system del dispositivo in cui si desidera installare il software AWS IoT Greengrass Core.

Valore predefinito: `/`

Nome di opzione per la modalità silenziosa: `--ggc-root-path`

Funzione Hello World Lambda

Indica se includere una funzione Hello World Lambda nel gruppo Greengrass. La funzione pubblica un messaggio MQTT sull'argomento `hello/world` ogni cinque secondi.

Lo script crea e pubblica questa funzione Lambda definita dall'utente AWS Lambda e la aggiunge al gruppo Greengrass. Lo script crea anche una sottoscrizione nel gruppo che consente alla funzione di inviare messaggi MQTT a AWS IoT.

Note

Questa è una funzione Lambda di Python 3.7. Se Python 3.7 non è installato sul dispositivo e lo script non è in grado di installarlo, lo script stampa un messaggio di errore nel terminale. Per includere la funzione Lambda nel gruppo, è necessario installare Python 3.7 manualmente e riavviare lo script. Per creare il gruppo Greengrass senza la funzione Lambda, riavviate lo script e immettete `no` quando richiesto di includere la funzione.

Valore predefinito: `no`

Nome di opzione per la modalità silenziosa: `--hello-world-lambda` - Questa opzione non accetta alcun valore. Includere il valore nel comando se si desidera creare la funzione.

Tempo di distribuzione

Il numero di secondi prima che l'installazione del dispositivo Greengrass interrompa il controllo dello stato della [distribuzione del gruppo Greengrass](#). Questo valore viene utilizzato solo quando il gruppo include la funzione Lambda Hello World. In caso contrario, il gruppo non viene distribuito.

Il tempo di distribuzione dipende dalla velocità della rete. Per velocità di rete lente, è possibile aumentare questo valore.

Valore predefinito: `180`

Nome di opzione per la modalità silenziosa: `--deployment-timeout`

Log Path (Percorso log)

Il percorso del file di registro che contiene informazioni sulle operazioni di installazione di base e gruppo Greengrass. Utilizza questo registro per risolvere i problemi relativi alla distribuzione e ad altri problemi relativi al gruppo Greengrass e all'installazione principale.

Valore predefinito: ./

Nome di opzione per la modalità silenziosa: --log-path

Livello di dettaglio

Indica se stampare informazioni di registro dettagliate nel terminale durante l'esecuzione dello script. Puoi utilizzare queste informazioni per risolvere i problemi di configurazione del dispositivo.

Valore predefinito: no

Nome di opzione per la modalità silenziosa: --verbose - Questa opzione non accetta alcun valore. Includi tale valore nel comando se si desidera stampare informazioni dettagliate di registro.

Esecuzione della configurazione del dispositivo Greengrass in modalità silenziosa

Puoi eseguire l'impostazione del dispositivo Greengrass in modalità silenziosa in modo che lo script non richieda alcun valore. Per eseguire l'operazione in modalità silenziosa, specifica la modalità `bootstrap-greengrass` e i [valori di input](#) dopo l'avvio dello script. Puoi omettere i valori di input se desideri utilizzare i valori predefiniti.

La procedura dipende dal fatto che vengano fornite le Account AWS credenziali come variabili di ambiente prima di avviare lo script o come valori di input dopo l'avvio dello script.

Fornire le credenziali come variabili di ambiente

1. [Fornisci le credenziali](#) come variabili di ambiente. Nell'esempio seguente vengono esportate credenziali temporanee, che includono il token di sessione.

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

Note

Se stai eseguendo la configurazione del dispositivo Greengrass su una piattaforma Raspbian o su una OpenWrt piattaforma, crea una copia di questi comandi. Devi fornirli nuovamente dopo il riavvio del dispositivo.

2. Scarica e avvia lo script. Fornisci i valori di input in base alle tue esigenze. Per esempio:

- Per utilizzare tutti i valori predefiniti:

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

- Per specificare valori personalizzati:

```
wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass
--region us-east-1
--group-name Custom_Group_Name
--core-name Custom_Core_Name
--ggc-root-path /custom/ggc/root/path
--deployment-timeout 300
--log-path /customized/log/path
--hello-world-lambda
--verbose
```

Note

Per utilizzare `curl` per scaricare lo script, sostituisci `wget -q -O` con `curl` nel comando.

3. Se il tuo dispositivo principale esegue Raspbian oppure OpenWrt, riavvia il dispositivo quando richiesto, fornisci le tue credenziali e quindi riavvia lo script.

- a. Quando viene richiesto di riavviare il dispositivo, esegui uno dei seguenti comandi.

Per le piattaforme Raspbian:

```
sudo reboot
```

Per le piattaforme: OpenWrt

```
reboot
```

- b. Dopo il riavvio avvio del dispositivo, apri il terminale e fornisci le credenziali come variabili di ambiente.

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

- c. Riavvia lo script.

```
sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass
```

- d. Quando viene chiesto se utilizzare i valori di input della sessione precedente o avviare una nuova installazione, immetti yes per riutilizzare i valori di input.

Note

Sulle piattaforme che richiedono un riavvio, tutti i valori di input (ad eccezione delle credenziali) della sessione precedente vengono temporaneamente memorizzati nel file `GreengrassDeviceSetup.config.info`.

Al termine dell'installazione, il terminale visualizza un messaggio di stato di successo simile al seguente.

```
=====
Your device is running the Greengrass core software.
Your Greengrass group and Hello World Lambda function were deployed to the core device.

Setup information:

Device info: Linux-4.14.152-127.182.amzn2.x86_64-x86_64-with-glibc2.2.5
Greengrass core software location: /
Installed Greengrass core software version: 1.10.0
Greengrass core: arn:aws:iot:us-west-2:012345678910:thing/GreengrassDeviceSetup_Core_d46a0ea4-18ae-4376-8f44-4a504cdea608
Greengrass core IoT certificate: arn:aws:iot:us-west-2:012345678910:cert/23fbf0f4b6a5ea369f2b97f1a1b558180a240faa8e059ce19dc58f4a4c0d3b77
Greengrass core IoT certificate location: /greengrass/certs/23fbf0f4b6.cert.pem
Greengrass core IoT key location: /greengrass/certs/23fbf0f4b6.private.key
Deployed Greengrass group name: GreengrassDeviceSetup_Group_ee70f777-9af0-43b6-8612-a18b418e8b4a
Deployed Greengrass group ID: 6f5c8410-f3a6-43a2-acf3-33158e10fb8e
Deployed Greengrass group version: arn:aws:greengrass:us-west-2:012345678910:/greengrass/groups/6f5c8410-f3a6-43a2-acf3-33158e10fb8e/versions/1
Greengrass service role: arn:aws:iam::012345678910:role/GreengrassServiceRole_mu1lv
GreengrassDeviceSetup log location: GreengrassDeviceSetup-20191209-232356.log
Deployed hello-world Lambda function: arn:aws:lambda:us-west-2:012345678910:function:Greengrass_HelloWorld_uNTf2:1
Hello-world subscriber topic: hello/world

You can now use the AWS IoT Console to subscribe
to the 'hello/world' topic to receive messages published from your
Greengrass core.
=====
```

4. Se hai incluso la funzione Hello World Lambda, la configurazione del dispositivo Greengrass distribuisce il gruppo Greengrass sul tuo dispositivo principale. Per testare la funzione Lambda o per informazioni su come rimuovere la funzione Lambda dal gruppo, continua con il Modulo 3-1 del tutorial [the section called “Verifica che la funzione Lambda sia in esecuzione sul dispositivo principale”](#) Getting Started.

 Note

Assicurati che quello Regione AWS selezionato nella console sia lo stesso che hai usato per configurare l'ambiente Greengrass. Per impostazione predefinita, la regione è Stati Uniti occidentali (Oregon).

Se non hai incluso la funzione Hello World Lambda, puoi [creare la tua funzione Lambda o provare altre funzionalità](#) di Greengrass. Ad esempio, puoi aggiungere il connettore di [distribuzione dell'applicazione Docker](#) al gruppo e utilizzarlo per distribuire contenitori Docker nel dispositivo principale.

Fornire le credenziali come valori di input

1. Scarica e avvia lo script. [Fornisci le credenziali](#) e gli altri valori di input che desideri specificare. Negli esempi seguenti viene illustrato come fornire credenziali temporanee, che includono il token di sessione.

- Per utilizzare tutti i valori predefiniti:

```
wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass  
--aws-access-key-id AKIAIOSFODNN7EXAMPLE  
--aws-secret-access-key wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY  
--aws-session-token AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

- Per specificare valori personalizzati:

```
wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass  
--aws-access-key-id AKIAIOSFODNN7EXAMPLE  
--aws-secret-access-key wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY  
--aws-session-token AQoDYXdzEJr1K...o50ytwEXAMPLE=  
--region us-east-1  
--group-name Custom_Group_Name  
--core-name Custom_Core_Name  
--ggc-root-path /custom/ggc/root/path  
--deployment-timeout 300  
--log-path /customized/log/path  
--hello-world-lambda  
--verbose
```

Note

Se stai eseguendo la configurazione del dispositivo Greengrass su una piattaforma Raspbian o su una OpenWrt piattaforma, crea una copia delle tue credenziali. Devi fornirli nuovamente dopo il riavvio del dispositivo.

Per utilizzare `curl` per scaricare lo script, sostituisci `wget -q -O` con `curl` nel comando.

2. Se il tuo dispositivo principale esegue Raspbian oppure OpenWrt, riavvia il dispositivo quando richiesto, fornisci le tue credenziali e quindi riavvia lo script.
 - a. Quando viene richiesto di riavviare il dispositivo, esegui uno dei seguenti comandi.

Per le piattaforme Raspbian:

```
sudo reboot
```

Per le piattaforme: OpenWrt

```
reboot
```

- b. Riavvia lo script. Devi includere le credenziali nel comando, ma non gli altri valori di input. Per esempio:

```
sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass
--aws-access-key-id AKIAIOSFODNN7EXAMPLE
--aws-secret-access-key wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
--aws-session-token AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

- c. Quando viene chiesto se utilizzare i valori di input della sessione precedente o avviare una nuova installazione, immetti yes per riutilizzare i valori di input.

Note

Sulle piattaforme che richiedono un riavvio, tutti i valori di input (ad eccezione delle credenziali) della sessione precedente vengono temporaneamente memorizzati nel file `GreengrassDeviceSetup.config.info`.

Al termine dell'installazione, il terminale visualizza un messaggio di stato di successo simile al seguente.

```
=====
Your device is running the Greengrass core software.
Your Greengrass group and Hello World Lambda function were deployed to the core device.

Setup information:

Device info: Linux-4.14.152-127.182.amzn2.x86_64-x86_64-with-glibc2.2.5
Greengrass core software location: /
Installed Greengrass core software version: 1.10.0
Greengrass core: arn:aws:iot:us-west-2:012345678910:thing/GreengrassDeviceSetup_Core_d46a0ea4-18ae-4376-8f44-4a504cdea608
Greengrass core IoT certificate: arn:aws:iot:us-west-2:012345678910:cert/23fbf0f4b6a5ea369f2b97f1a1b558180a240faa8e059ce19dc58f4a4c0d3b77
Greengrass core IoT certificate location: /greengrass/certs/23fbf0f4b6.cert.pem
Greengrass core IoT key location: /greengrass/certs/23fbf0f4b6.private.key
Deployed Greengrass group name: GreengrassDeviceSetup_Group_ee70f777-9af0-43b6-8612-a18b418e8b4a
Deployed Greengrass group ID: 6f5c8410-f3a6-43a2-acf3-33158e10fb8e
Deployed Greengrass group version: arn:aws:greengrass:us-west-2:012345678910:/greengrass/groups/6f5c8410-f3a6-43a2-acf3-33158e10fb8e/versions/1
Greengrass service role: arn:aws:iam::012345678910:role/GreengrassServiceRole_mui1v
GreengrassDeviceSetup log location: GreengrassDeviceSetup-20191209-232356.log
Deployed hello-world Lambda function: arn:aws:lambda:us-west-2:012345678910:function:Greengrass_HelloWorld_uNTf2:1
Hello-world subscriber topic: hello/world

You can now use the AWS IoT Console to subscribe
to the 'hello/world' topic to receive messages published from your
Greengrass core.
=====
```

3. Se hai incluso la funzione Hello World Lambda, la configurazione del dispositivo Greengrass distribuisce il gruppo Greengrass sul tuo dispositivo principale. Per testare la funzione Lambda o per informazioni su come rimuovere la funzione Lambda dal gruppo, continua con il Modulo 3-1

del tutorial [the section called “Verifica che la funzione Lambda sia in esecuzione sul dispositivo principale”](#) Getting Started.

 Note

Assicurati che quello Regione AWS selezionato nella console sia lo stesso che hai usato per configurare l'ambiente Greengrass. Per impostazione predefinita, la regione è Stati Uniti occidentali (Oregon).

Se non hai incluso la funzione Hello World Lambda, puoi [creare la tua funzione Lambda o provare altre funzionalità](#) di Greengrass. Ad esempio, puoi aggiungere il connettore di [distribuzione dell'applicazione Docker](#) al gruppo e utilizzarlo per distribuire contenitori Docker nel dispositivo principale.

Modulo 1: impostazione dell'ambiente per Greengrass

Questo modulo mostra come preparare un out-of-the-box Raspberry Pi, un' EC2 istanza Amazon o un altro dispositivo per essere utilizzato AWS IoT Greengrass come dispositivo AWS IoT Greengrass principale.

 Tip

Oppure, per utilizzare uno script che configuri automaticamente il dispositivo core, consulta [the section called “Avvio rapido: configurazione dispositivo Greengrass”](#).

Il completamento di questo modulo dovrebbe richiedere meno di 30 minuti.

Prima di iniziare, leggi i [requisiti](#) per questo tutorial. Segui quindi le istruzioni di installazione in uno dei seguenti argomenti. Scegli solo l'argomento che si applica al tipo di dispositivo principale.

Argomenti

- [Impostazione di un Raspberry Pi](#)
- [Configurazione di un' EC2 istanza Amazon](#)
- [Configurazione di altri dispositivi](#)

Note

Per informazioni su come utilizzare l' AWS IoT Greengrass esecuzione in un contenitore Docker predefinito, consulta [the section called “Esegui AWS IoT Greengrass in un contenitore Docker”](#)

Impostazione di un Raspberry Pi

Segui i passaggi descritti in questo argomento per configurare un Raspberry Pi da utilizzare come AWS IoT Greengrass core.

Tip

AWS IoT Greengrass fornisce anche altre opzioni per l'installazione del software AWS IoT Greengrass Core. Ad esempio, è possibile utilizzare la [configurazione del dispositivo Greengrass](#) per configurare l'ambiente e installare l'ultima versione del software AWS IoT Greengrass Core. Oppure, sulle piattaforme Debian supportate, è possibile utilizzare il [gestore di pacchetti APT](#) per installare o aggiornare il AWS IoT Greengrass software Core. Per ulteriori informazioni, consulta [the section called “Installare il software AWS IoT Greengrass Core”](#).

Se stai impostando un Raspberry Pi per la prima volta, è necessario seguire tutti questi passaggi. In caso contrario, passa alla [fase 9](#). Tuttavia, ti consigliamo di reimpostare il tuo Raspberry Pi con il sistema operativo come raccomandato nel passaggio 2.

1. Scaricate e installate un formattatore di schede SD come [SD Memory Card Formatter](#). Inserisci la scheda SD nel computer. Avvia il programma e scegli l'unità in cui è inserita la scheda SD. È possibile formattare rapidamente la scheda SD.
2. Scarica il sistema operativo [Raspbian Buster](#) come file zip.
3. Utilizzando uno strumento di scrittura su scheda SD (ad esempio [Etcher](#)), segui le istruzioni per trasferire il file zip scaricato nella scheda SD. Poiché l'immagine del sistema operativo è di grandi dimensioni, l'operazione potrebbe richiedere alcuni minuti. Espelli la scheda SD dal computer e inserisci la scheda MicroSD nel Raspberry Pi.

4. Per il primo avvio, ti consigliamo di connettere il Raspberry Pi a un monitor (tramite HDMI), una tastiera e un mouse. Quindi, collega il Pi a una sorgente di alimentazione micro USB e il sistema operativo Raspbian dovrebbe avviarsi.
5. È possibile configurare il layout della tastiera del Pi prima di continuare. Per farlo, scegli l'icona Raspberry in alto a destra, scegli Preferences (Preferenze), quindi Mouse and Keyboard Settings (Impostazioni mouse e tastiera). Quindi, sulla scheda Keyboard (Tastiera) scegli, Keyboard Layout (Layout tastiera) e scegli una variante adatta di tastiera.
6. Quindi, [collega il Raspberry Pi a Internet tramite una rete Wi-Fi](#) o un cavo Ethernet.

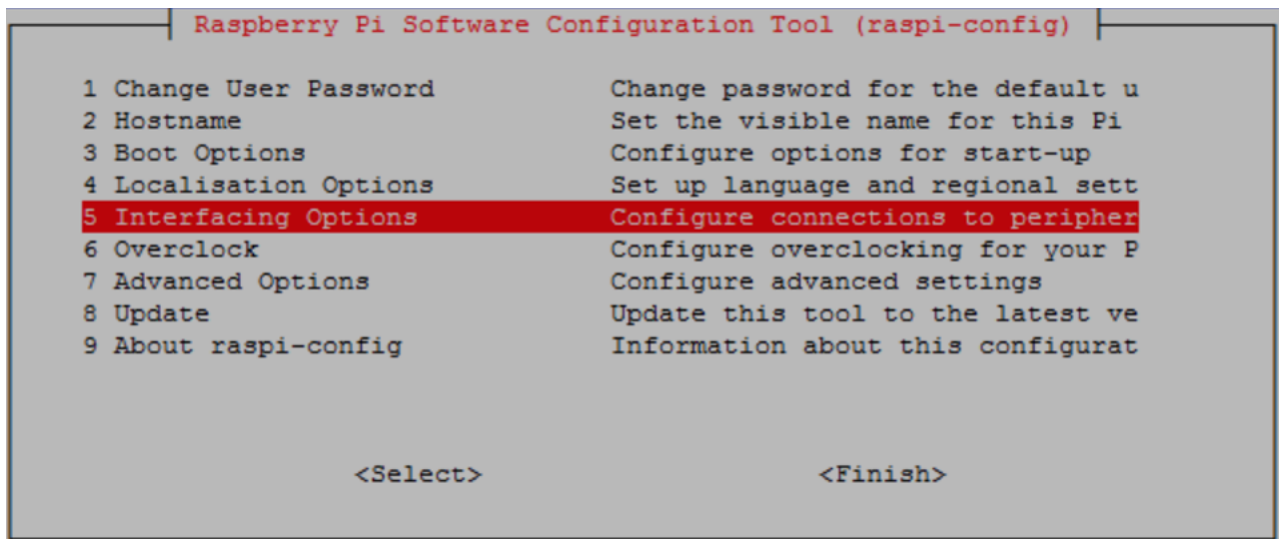
Note

Collega il Raspberry Pi alla stessa rete a cui è connesso il computer e accertati che sia il computer che il Raspberry Pi abbiano accesso a Internet prima di continuare. Se ti trovi in un ambiente di lavoro o dietro un firewall, potrebbe essere necessario collegare il Pi e il computer alla rete guest per avere entrambi i dispositivi sulla stessa rete. Questo approccio, tuttavia, potrebbe scollegare il computer dalle risorse di rete locali, ad esempio l'Intranet. Una soluzione potrebbe essere quella di collegare il Pi alla rete Wi-Fi guest e il computer alla rete Wi-Fi guest e alla rete locale tramite un cavo Ethernet. Con questa configurazione dovresti essere in grado di connetterti al Raspberry Pi tramite la rete Wi-Fi guest e alle risorse di rete locali tramite il cavo Ethernet.

7. È necessario configurare [SSH](#) sul Pi per connetterti in remoto. Apri una [finestra del terminale](#) sul Raspberry Pi ed esegui il comando seguente:

```
sudo raspi-config
```

Verrà visualizzato un codice analogo al seguente:



Scorri in basso e scegli Interfacing Options (Opzioni di interfaccia), quindi scegli P2 SSH. Quando viene richiesto, scegliere Yes (Sì). (Utilizza la chiave Tab seguita da Enter). L'SSH deve essere abilitata. Scegli OK. Usa il tasto Tab per scegliere Finish (Fine) e quindi premi Enter. Se il dispositivo Raspberry Pi non si riavvia automaticamente, esegui il comando seguente:

```
sudo reboot
```

8. Esegui il comando seguente nel terminale del Raspberry Pi:

```
hostname -I
```

Questo restituisce l'indirizzo IP del Raspberry Pi.

Note

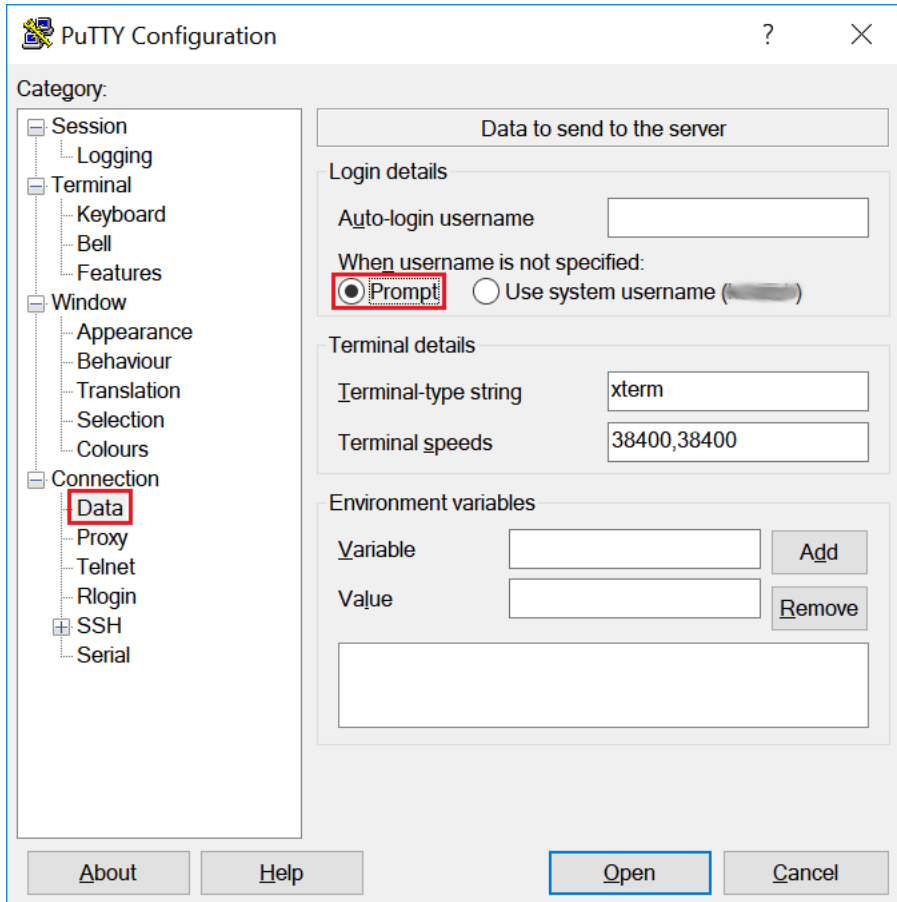
Successivamente, se ricevi un messaggio relativo all'impronta della chiave ECDSA (Are you sure you want to continue connecting (yes/no)?), immetti yes. La password predefinita per il Raspberry Pi è **raspberrypi**.

Se utilizzi macOS, apri una finestra del terminale e digita:

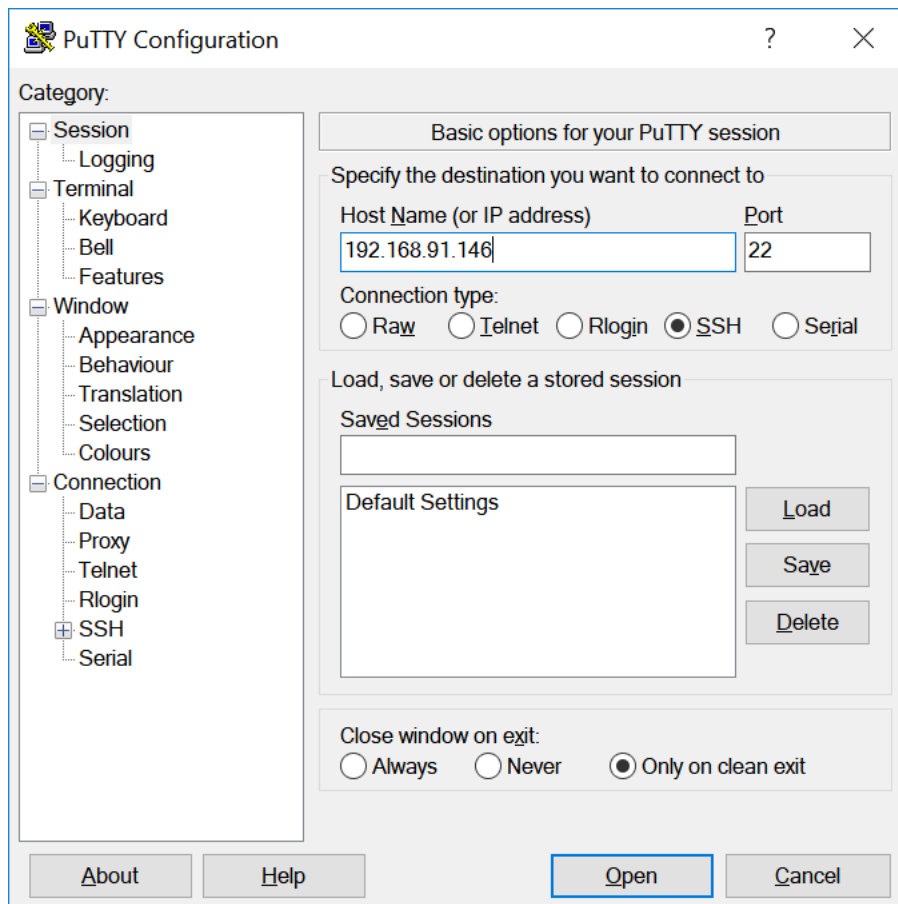
```
ssh pi@IP-address
```

IP-address è l'indirizzo IP del tuo Raspberry Pi che hai ottenuto utilizzando il comando.
`hostname -I`

Se stai usando Windows, è necessario installare e configurare [PuTTY](#). Apri Connection (Connessione), Data (Dati) e accertati che sia selezionato Prompt:

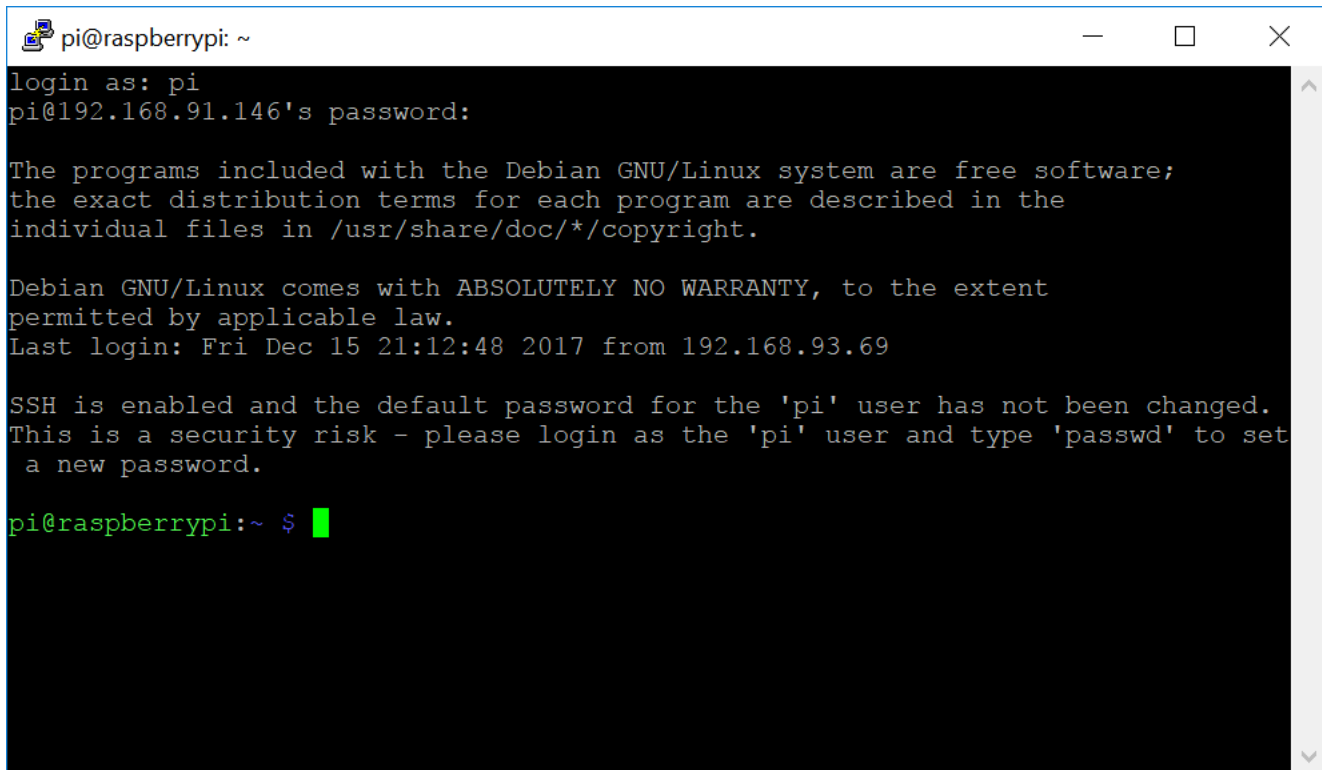


Quindi, scegli Session (Sessione), immetti l'indirizzo IP del Raspberry Pi e scegli Open (Apri) utilizzando le impostazioni predefinite.



Se viene visualizzato il messaggio " PuTTY security alert", scegli Yes (Sì).

I valori predefiniti di login e password di Raspberry Pi sono rispettivamente **pi** e **raspberrypi**.



```
pi@raspberrypi: ~
login as: pi
pi@192.168.91.146's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Dec 15 21:12:48 2017 from 192.168.93.69

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
a new password.

pi@raspberrypi:~ $
```

Note

Se il computer è connesso a una rete remota tramite VPN, questo può causare difficoltà di connessione dal computer al Raspberry Pi tramite SSH.

9. Ora sei pronto per configurare il Raspberry Pi per AWS IoT Greengrass. In primo luogo, esegui i seguenti comandi da una finestra del terminale locale del Raspberry Pi o da una finestra del terminale SSH:

Tip

AWS IoT Greengrass fornisce anche altre opzioni per l'installazione del software AWS IoT Greengrass Core. Ad esempio, è possibile utilizzare la [configurazione del dispositivo Greengrass](#) per configurare l'ambiente e installare l'ultima versione del software AWS IoT Greengrass Core. Oppure, sulle piattaforme Debian supportate, è possibile utilizzare il [gestore di pacchetti APT](#) per installare o aggiornare il AWS IoT Greengrass software Core. Per ulteriori informazioni, consulta [the section called "Installare il software AWS IoT Greengrass Core"](#).

```
sudo adduser --system ggc_user
sudo addgroup --system ggc_group
```

10. Per migliorare la sicurezza sul dispositivo Pi, abilitare la protezione hardlink e softlink (symlink) sul sistema operativo all'avvio.

- a. Andare al file `98-rpi.conf`.

```
cd /etc/sysctl.d
ls
```

 Note

Se non visualizzi il file `98-rpi.conf`, segui le istruzioni contenute nel file `README.sysctl`.

- b. Utilizza un editor di testo (ad esempio Leafpad, GNU nano o vi) per aggiungere le seguenti due righe alla fine del file. Potresti dover utilizzare il comando `sudo` per modificare come root (ad esempio `sudo nano 98-rpi.conf`).

```
fs.protected_hardlinks = 1
fs.protected_symlinks = 1
```

- c. Riavvia il dispositivo Pi.

```
sudo reboot
```

Dopo un minuto circa, connettiti al dispositivo Pi tramite SSH, quindi esegui il comando seguente per confermare la modifica:

```
sudo sysctl -a 2> /dev/null | grep fs.protected
```

Dovresti visualizzare `fs.protected_hardlinks = 1` e `fs.protected_symlinks = 1`.

11. Modifica il file di avvio a riga di comando per abilitare e montare cgroups di memoria. Ciò consente di AWS IoT Greengrass impostare il limite di memoria per le funzioni Lambda. I

Cgroup devono inoltre essere eseguiti AWS IoT Greengrass nella modalità di [containerizzazione](#) predefinita.

- a. Passa alla directory boot.

```
cd /boot/
```

- b. Aprire il file `cmdline.txt` con un editor di testo. Aggiungi quanto segue alla fine della linea esistente, non come una nuova riga. Potresti dover utilizzare il comando `sudo` per modificare come root (ad esempio `sudo nano cmdline.txt`).

```
cgroup_enable=memory cgroup_memory=1
```

- c. Ora riavvia il dispositivo Pi.

```
sudo reboot
```

Il Raspberry Pi dovrebbe essere pronto per AWS IoT Greengrass.

12. Facoltativo. Installare il runtime Java 8, richiesto dal [Gestore di flussi](#). Questo tutorial non utilizza Gestore di flussi, ma utilizza il flusso di lavoro Creazione gruppo predefinito che abilita Gestore di flussi per impostazione predefinita. Utilizzare i seguenti comandi per installare il runtime Java 8 sul dispositivo principale o disabilitare Gestore di flussi prima di distribuire il gruppo. Le istruzioni per disabilitare Gestore di flussi sono fornite nel Modulo 3.

```
sudo apt install openjdk-8-jdk
```

13. [Per assicurarti di avere tutte le dipendenze necessarie, scarica ed esegui il controllo delle dipendenze Greengrass dal AWS IoT Greengrass repository Samples in poi.](#) GitHub Questi comandi decomprimono ed eseguono lo script di controllo delle dipendenze nella directory. Downloads

Note

Il controllo delle dipendenze potrebbe fallire se si utilizza la versione 5.4.51 del kernel Raspbian. Questa versione non monta correttamente i cgroup di memoria. Ciò potrebbe causare il fallimento delle funzioni Lambda eseguite in modalità contenitore.

Per ulteriori informazioni sull'aggiornamento del kernel, consulta i [Cgroups non caricati dopo l'aggiornamento del kernel](#) nei forum di Raspberry Pi.

```
cd /home/pi/Downloads
mkdir greengrass-dependency-checker-GGCv1.11.x
cd greengrass-dependency-checker-GGCv1.11.x
wget https://github.com/aws-samples/aws-greengrass-samples/raw/master/greengrass-dependency-checker-GGCv1.11.x.zip
unzip greengrass-dependency-checker-GGCv1.11.x.zip
cd greengrass-dependency-checker-GGCv1.11.x
sudo modprobe configs
sudo ./check_ggc_dependencies | more
```

Dove viene visualizzato `more`, premi il tasto Spacebar per visualizzare un'altra schermata di testo.

Important

Questo tutorial richiede il runtime di Python 3.7 per eseguire le funzioni Lambda locali. Quando il gestore di flusso è abilitato, richiede anche il runtime Java 8. Se lo script `check_ggc_dependencies` produce avvisi su questi prerequisiti di runtime mancanti, assicurati di installarli prima di continuare. Puoi ignorare gli avvisi relativi ad altri prerequisiti di runtime facoltativi mancanti.

Per informazioni sul comando `modprobe`, esegui `man modprobe` nel terminale.

La configurazione del Raspberry Pi è completa. Continua su [the section called “Modulo 2: Installazione del software AWS IoT Greengrass Core”](#).

Configurazione di un' EC2 istanza Amazon

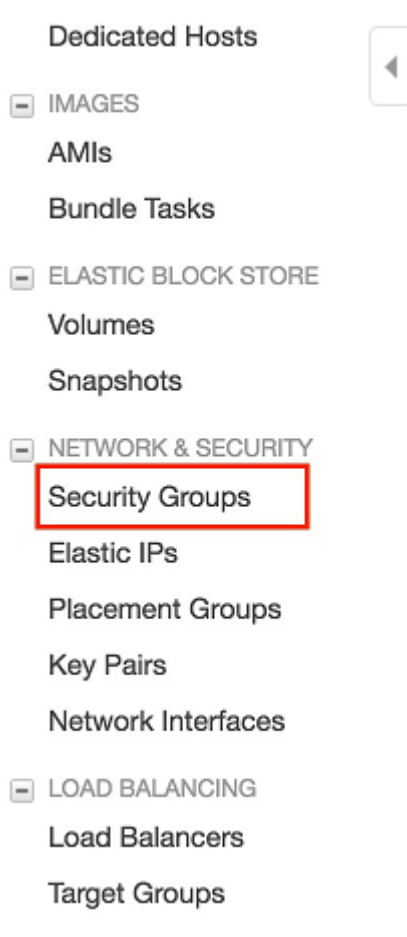
Segui i passaggi descritti in questo argomento per configurare un' EC2 istanza Amazon da utilizzare come AWS IoT Greengrass core.

 Tip

Oppure, per utilizzare uno script che configura il tuo ambiente e installa il software AWS IoT Greengrass Core per te, consulta [the section called “Avvio rapido: configurazione dispositivo Greengrass”](#).

Sebbene sia possibile completare questo tutorial utilizzando un' EC2 istanza Amazon, idealmente AWS IoT Greengrass dovrebbe essere utilizzato con hardware fisico. Ti consigliamo di [configurare un Raspberry Pi](#) invece di utilizzare un' EC2 istanza Amazon quando possibile. Se si utilizza un Raspberry Pi, non è necessario seguire la procedura descritta in questo argomento.

1. Accedi [AWS Management Console](#) e avvia un' EC2 istanza Amazon utilizzando un'AMI Amazon Linux. Per informazioni sulle EC2 istanze Amazon, consulta la [Amazon EC2 Getting Started Guide](#).
2. Dopo l'esecuzione dell' EC2 istanza Amazon, abilita la porta 8883 per consentire le comunicazioni MQTT in entrata in modo che altri dispositivi possano connettersi al core. AWS IoT Greengrass
 - a. Nel pannello di navigazione della EC2 console Amazon, scegli Gruppi di sicurezza.



- b. Seleziona il gruppo di sicurezza per l'istanza appena avviata, quindi scegli la scheda Regole in entrata.
- c. Scegliere Edit inbound rules (Modifica regole in entrata).

Per abilitare la porta 8883, aggiungi una regola TCP personalizzata al gruppo di sicurezza. Per ulteriori informazioni, consulta [Aggiungere regole a un gruppo di sicurezza](#) nella Amazon EC2 User Guide.

- d. Nella pagina Modifica regole in entrata, scegli Aggiungi regola, inserisci le seguenti impostazioni, quindi scegli Salva.
 - Per Type (Tipo) seleziona Custom TCP Rule (Regola TCP personalizzata).
 - Per Port range, inserisci **8883**.
 - Per Source (Origine), selezionare Anywhere (Ovunque).
 - Per Descrizione, inserisci **MQTT Communications**.

3. Connect alla tua EC2 istanza Amazon.

- a. Nel riquadro di navigazione, scegli Instances (Istanze), seleziona la tua istanza, quindi scegli Connect (Connetti).
- b. Segui le istruzioni nella pagina Connect To Your Instance (Connettiti alla tua istanza) per connetterti all'istanza [tramite SSH](#) e il tuo file della chiave privata.

È possibile utilizzare [PuTTY](#) per Windows o Terminal per macOS. Per ulteriori informazioni, consulta [Connect to your Linux instance](#) nella Amazon EC2 User Guide.

Ora sei pronto per configurare la tua EC2 istanza Amazon per AWS IoT Greengrass.

4. Dopo esserti connesso alla tua EC2 istanza Amazon, crea gli ggc_group account ggc_user e:

```
sudo adduser --system ggc_user
sudo groupadd --system ggc_group
```

Note

Se il comando `adduser` non è disponibile nel sistema, utilizza il comando seguente.

```
sudo useradd --system ggc_user
```

5. Per migliorare la sicurezza, assicurati che le protezioni hardlink e softlink (symlink) siano abilitate sul sistema operativo dell'istanza Amazon EC2 all'avvio.

Note

Le procedure di abilitazione hardlink e softlink possono variare a seconda del sistema operativo. Consultare la documentazione per la distribuzione.

- a. Eseguire il comando seguente per controllare se le protezioni hardlink e softlink sono abilitate:

```
sudo sysctl -a | grep fs.protected
```

Se hardlink e softlink vengono impostati su 1, le protezioni sono abilitate correttamente. Continuare con la fase 6.

 Note

I softlink sono rappresentati da `fs.protected_symlinks`.

- b. Se hardlink e softlink non sono impostati su 1, abilitare queste protezioni. Navigare al file di configurazione del sistema.

```
cd /etc/sysctl.d
ls
```

- c. Utilizza il tuo editor di testo preferito (ad esempio Leafpad, GNU nano o vi) per aggiungere le seguenti due righe alla fine del file di configurazione del sistema. Su Amazon Linux 1, questo è il file `00-defaults.conf`. Su Amazon Linux 2, questo è il file `99-amazon.conf`. Potrebbe essere necessario modificare le autorizzazioni (utilizzando il comando `chmod`) per scrivere sul file oppure utilizzare il comando `sudo` per modificare come root (ad esempio `sudo nano 00-defaults.conf`).

```
fs.protected_hardlinks = 1
fs.protected_symlinks = 1
```

- d. Riavvia l' EC2 istanza Amazon.

```
sudo reboot
```

Dopo pochi minuti, connettiti all'istanza tramite SSH, quindi esegui il comando seguente per confermare la modifica.

```
sudo sysctl -a | grep fs.protected
```

Dovresti vedere che hardlink e softlink sono impostati a 1.

6. Estrai ed esegui il seguente script per montare i [gruppi di controllo Linux](#) (cgroups). Ciò consente di AWS IoT Greengrass impostare il limite di memoria per le funzioni Lambda. I Cgroup devono inoltre essere eseguiti AWS IoT Greengrass nella modalità di [containerizzazione](#) predefinita.

```
curl https://raw.githubusercontent.com/tianon/cgroupfs-mount/951c38ee8d802330454bdede20d85ec1c0f8d312/cgroupfs-mount > cgroupfs-mount.sh
chmod +x cgroupfs-mount.sh
sudo bash ./cgroupfs-mount.sh
```

La tua EC2 istanza Amazon dovrebbe ora essere pronta per AWS IoT Greengrass.

7. Facoltativo. Installare il runtime Java 8, richiesto dal [Gestore di flussi](#). Questo tutorial non utilizza Gestore di flussi, ma utilizza il flusso di lavoro Creazione gruppo predefinito che abilita Gestore di flussi per impostazione predefinita. Utilizzare i seguenti comandi per installare il runtime Java 8 sul dispositivo principale o disabilitare Gestore di flussi prima di distribuire il gruppo. Le istruzioni per disabilitare Gestore di flussi sono fornite nel Modulo 3.

- Per le distribuzioni basate su Debian:

```
sudo apt install openjdk-8-jdk
```

- Per le distribuzioni basate su Red Hat:

```
sudo yum install java-1.8.0-openjdk
```

8. [Per assicurarti di avere tutte le dipendenze necessarie, scarica ed esegui il controllo delle dipendenze Greengrass dal AWS IoT Greengrass repository Samples in poi.](#) GitHub Questi comandi scaricano, decomprimono ed eseguono lo script di controllo delle dipendenze nella tua istanza Amazon. EC2

```
mkdir greengrass-dependency-checker-GGCv1.11.x
cd greengrass-dependency-checker-GGCv1.11.x
wget https://github.com/aws-samples/aws-greengrass-samples/raw/master/greengrass-dependency-checker-GGCv1.11.x.zip
unzip greengrass-dependency-checker-GGCv1.11.x.zip
cd greengrass-dependency-checker-GGCv1.11.x
sudo ./check_ggc_dependencies | more
```

Important

Questo tutorial richiede il runtime di Python 3.7 per eseguire le funzioni Lambda locali. Quando il gestore di flusso è abilitato, richiede anche il runtime Java 8. Se lo script `check_ggc_dependencies` produce avvisi su questi prerequisiti di runtime mancanti,

assicurati di installarli prima di continuare. Puoi ignorare gli avvisi relativi ad altri prerequisiti di runtime facoltativi mancanti.

La configurazione dell' EC2 istanza Amazon è completa. Continua su [the section called “Modulo 2: Installazione del software AWS IoT Greengrass Core”](#).

Configurazione di altri dispositivi

Segui i passaggi descritti in questo argomento per configurare un dispositivo (diverso da un Raspberry Pi) da utilizzare come AWS IoT Greengrass core.

Tip

Oppure, per utilizzare uno script che configura il tuo ambiente e installa il software AWS IoT Greengrass Core per te, consulta. [the section called “Avvio rapido: configurazione dispositivo Greengrass”](#)

Se non lo usi AWS IoT Greengrass, ti consigliamo di utilizzare un' EC2istanza Raspberry Pi o Amazon come dispositivo principale e di seguire i [passaggi di configurazione](#) appropriati per il tuo dispositivo.

Se hai intenzione di creare un sistema personalizzato basato su Linux utilizzando il progetto Yocto, puoi utilizzare la Bitbake Recipe del AWS IoT Greengrass progetto. `meta-aws` Questa ricetta aiuta anche a sviluppare una piattaforma software che supporti AWS il software edge per applicazioni integrate. La build Bitbake installa, configura ed esegue automaticamente il software AWS IoT Greengrass Core sul dispositivo.

Progetto Yocto

Un progetto di collaborazione open source che consente di creare sistemi personalizzati basati su Linux per applicazioni integrate indipendentemente dall'architettura hardware. [Per ulteriori informazioni, consulta il progetto Yocto.](#)

`meta-aws`

Un progetto AWS gestito che fornisce ricette Yocto. Puoi usare le ricette per sviluppare software AWS edge in sistemi basati su Linux creati con Yocto Project. [OpenEmbedded](#) Per ulteriori

informazioni su questa funzionalità supportata dalla comunità, consulta il progetto su [meta-aws](#) GitHub

meta-aws-demos

Un progetto AWS gestito che contiene dimostrazioni del meta-aws progetto. Per altri esempi sul processo di integrazione, consulta il [meta-aws-demos](#) progetto su GitHub.

Per utilizzare un dispositivo diverso o un'altra [piattaforma supportata](#), seguire i passaggi descritti in questo argomento.

1. Se il tuo dispositivo principale è un dispositivo NVIDIA Jetson, devi prima eseguire il flashing del firmware con il JetPack programma di installazione 4.3. Se stai configurando un altro dispositivo, passa alla fase 2.

 Note

La versione di JetPack installazione che utilizzi si basa sulla versione di CUDA Toolkit di destinazione. Le seguenti istruzioni utilizzano JetPack 4.3 e CUDA Toolkit 10.0. Per informazioni sull'utilizzo delle versioni appropriate per il dispositivo, consulta [How to Install Jetpack](#) nella documentazione di NVIDIA.

- a. Su un desktop fisico che esegue Ubuntu 16.04 o versione successiva, esegui il flashing del firmware con il programma di installazione JetPack 4.3, come descritto in [Download e installazione JetPack](#) (4.3) nella documentazione NVIDIA.

Segui le istruzioni del programma di installazione per installare tutti i pacchetti e le dipendenze sulla scheda Jetson, che deve essere connessa al desktop con un cavo Micro-B.

- b. Riavvia la scheda in modalità normale e connetti un display alla scheda.

 Note

Quando utilizzi SSH per connetterti alla scheda Jetson, utilizza il nome utente predefinito (**nvidia**) e la password predefinita (**nvidia**).

2. Esegui i seguenti comandi per creare l'utente ggc_user e il gruppo ggc_group. I comandi che esegui variano a seconda della distribuzione installata sul dispositivo core.

- Se il dispositivo principale è in esecuzione OpenWrt, esegui i seguenti comandi:

```
opkg install shadow-useradd
opkg install shadow-groupadd
useradd --system ggc_user
groupadd --system ggc_group
```

- In caso contrario, esegui i comandi seguenti:

```
sudo adduser --system ggc_user
sudo addgroup --system ggc_group
```

Note

Se il comando `addgroup` non è disponibile nel sistema, utilizza il comando seguente.

```
sudo groupadd --system ggc_group
```

3. Facoltativo. Installare il runtime Java 8, richiesto dal [Gestore di flussi](#). Questo tutorial non utilizza Gestore di flussi, ma utilizza il flusso di lavoro Creazione gruppo predefinito che abilita Gestore di flussi per impostazione predefinita. Utilizzare i seguenti comandi per installare il runtime Java 8 sul dispositivo principale o disabilitare Gestore di flussi prima di distribuire il gruppo. Le istruzioni per disabilitare Gestore di flussi sono fornite nel Modulo 3.

- Per le distribuzioni basate su Debian o basate su Ubuntu:

```
sudo apt install openjdk-8-jdk
```

- Per le distribuzioni basate su Red Hat:

```
sudo yum install java-1.8.0-openjdk
```

4. [Per assicurarti di avere tutte le dipendenze necessarie, scarica ed esegui il controllo delle dipendenze Greengrass dal AWS IoT Greengrass repository Samples in poi.](#) GitHub Questi comandi decomprimono ed eseguono lo script di controllo delle dipendenze.

```
mkdir greengrass-dependency-checker-GGCv1.11.x
cd greengrass-dependency-checker-GGCv1.11.x
```

```
wget https://github.com/aws-samples/aws-greengrass-samples/raw/master/greengrass-  
dependency-checker-GGCv1.11.x.zip  
unzip greengrass-dependency-checker-GGCv1.11.x.zip  
cd greengrass-dependency-checker-GGCv1.11.x  
sudo ./check_ggc_dependencies | more
```

Note

Lo `check_ggc_dependencies` script viene eseguito su piattaforme AWS IoT Greengrass supportate e richiede comandi di sistema Linux specifici. Per ulteriori informazioni, consulta il file [Readme](#) dello strumento di controllo delle dipendenze.

5. Installa tutte le dipendenze necessarie sul dispositivo, come indicato dall'output dello strumento di controllo delle dipendenze. Per dipendenze mancanti a livello di kernel, potrebbe essere necessario ricompilare il kernel. Per montare i gruppi di controllo Linux (cgroups), è possibile eseguire lo script [cgroupfs-mount](#). Ciò consente di AWS IoT Greengrass impostare il limite di memoria per le funzioni Lambda. I Cgroup devono inoltre essere eseguiti AWS IoT Greengrass nella modalità di [containerizzazione](#) predefinita.

Se non compaiono errori nell'output, AWS IoT Greengrass dovrebbe essere in grado di funzionare correttamente sul dispositivo.

Important

Questo tutorial richiede il runtime di Python 3.7 per eseguire le funzioni Lambda locali. Quando il gestore di flusso è abilitato, richiede anche il runtime Java 8. Se lo script `check_ggc_dependencies` produce avvisi su questi prerequisiti di runtime mancanti, assicurati di installarli prima di continuare. Puoi ignorare gli avvisi relativi ad altri prerequisiti di runtime facoltativi mancanti.

Per l'elenco dei AWS IoT Greengrass requisiti e delle dipendenze, vedi. [the section called "Piattaforme supportate e requisiti"](#)

Modulo 2: Installazione del software AWS IoT Greengrass Core

Questo modulo mostra come installare il software AWS IoT Greengrass Core sul dispositivo scelto. In questo modulo, si crea innanzitutto un gruppo e un core di Greengrass. Quindi, scaricare, configurare

e avviare il software sul dispositivo core. Per ulteriori informazioni sulle funzionalità del software AWS IoT Greengrass Core, consulta [the section called “Configura il AWS IoT Greengrass core”](#).

Prima di iniziare, assicurarsi di aver completato la procedura di configurazione nel [Modulo 1](#) per il dispositivo scelto.

 Tip

AWS IoT Greengrass fornisce anche altre opzioni per l'installazione del software AWS IoT Greengrass Core. Ad esempio, è possibile utilizzare la [configurazione del dispositivo Greengrass](#) per configurare l'ambiente e installare l'ultima versione del software AWS IoT Greengrass Core. Oppure, sulle piattaforme Debian supportate, è possibile utilizzare il [gestore di pacchetti APT](#) per installare o aggiornare il AWS IoT Greengrass software Core. Per ulteriori informazioni, consulta [the section called “Installare il software AWS IoT Greengrass Core”](#).

Il completamento di questo modulo dovrebbe richiedere meno di 30 minuti.

Argomenti

- [Fornitura di AWS IoT qualsiasi cosa da utilizzare come core Greengrass](#)
- [Crea un AWS IoT Greengrass gruppo per il core](#)
- [Installa ed esegui AWS IoT Greengrass sul dispositivo principale](#)

Fornitura di AWS IoT qualsiasi cosa da utilizzare come core Greengrass

I core Greengrass sono dispositivi che eseguono il software AWS IoT Greengrass Core per gestire i processi IoT locali. Per configurare un core Greengrass, crei un AWS IoT oggetto, che rappresenta un dispositivo o un'entità logica a cui si connette. AWS IoT Quando si registra un dispositivo come AWS IoT oggetto, quel dispositivo può utilizzare un certificato digitale e delle chiavi che gli consentono l'accesso AWS IoT. Si utilizza una [AWS IoT policy](#) per consentire al dispositivo di comunicare con i AWS IoT Greengrass servizi AWS IoT and.

In questa sezione, registri il tuo dispositivo come AWS IoT qualsiasi cosa per usarlo come core Greengrass.

Per creare qualsiasi cosa AWS IoT

1. Passare alla [console AWS IoT](#).
2. In Gestisci, espandi Tutti i dispositivi, quindi scegli Cose.
3. Sulla pagina Things (Oggetti), scegli Create things (Creazione di oggetti).
4. Nella pagina Crea elementi, scegli Crea un singolo elemento, quindi scegli Avanti.
5. Nella pagina Specificare le proprietà dell'oggetto, procedi come segue:
 - a. Per Nome oggetto, immettere un nome che rappresenti il dispositivo, ad esempio **MyGreengrassV1Core**.
 - b. Scegli Next (Successivo).
6. Nella pagina Configura il certificato del dispositivo, scegli Avanti.
7. Nella pagina Allega le politiche al certificato, esegui una delle seguenti operazioni:
 - Seleziona una politica esistente che conceda le autorizzazioni richieste dai core, quindi scegli Crea oggetto.

Si apre una finestra modale in cui è possibile scaricare i certificati e le chiavi utilizzati dal dispositivo per connettersi a Cloud AWS

- Crea e allega una nuova politica che conceda le autorizzazioni principali del dispositivo. Esegui questa operazione:
 - a. Scegli Create Policy (Crea policy).

La pagina Create policy (Crea policy) viene aperta in una nuova scheda.

- b. Nella pagina Create policy (Crea policy), eseguire le operazioni seguenti:
 - i. Per Nome della politica, inserisci un nome che descriva la politica, ad esempio. **GreengrassV1CorePolicy**
 - ii. Nella scheda Dichiarazioni politiche, in Documento di politica, scegli JSON.
 - iii. Inserisci il seguente documento di policy. Questa policy consente al core di comunicare con il AWS IoT Core servizio, interagire con le ombre dei dispositivi e comunicare con il AWS IoT Greengrass servizio. Per informazioni su come limitare l'accesso a questa policy in base al tuo caso d'uso, consulta [AWS IoT Politica minima per il dispositivo AWS IoT Greengrass principale](#).

```
{
```

```

"Version": "2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "iot:Publish",
      "iot:Subscribe",
      "iot:Connect",
      "iot:Receive"
    ],
    "Resource": [
      "*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "iot:GetThingShadow",
      "iot:UpdateThingShadow",
      "iot:DeleteThingShadow"
    ],
    "Resource": [
      "*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "greengrass:*"
    ],
    "Resource": [
      "*"
    ]
  }
]
}

```

- iv. Scegliere Crea per creare la policy.
- c. Torna alla scheda del browser con la pagina Allega politiche al certificato aperta. Esegui questa operazione:
 - i. Nell'elenco Politiche, seleziona la politica che hai creato, ad esempio GreengrassV1CorePolicy.

Se non vedi la politica, scegli il pulsante di aggiornamento.

- ii. Scegli Create thing (Crea oggetto).

Si apre una finestra modale in cui è possibile scaricare i certificati e le chiavi a cui il core utilizza per connettersi. AWS IoT

8. Torna alla scheda del browser con la pagina Allega politiche al certificato aperta. Esegui questa operazione:

- a. Nell'elenco Politiche, seleziona la politica che hai creato, ad esempio GreengrassV1CorePolicy.

Se non vedi la politica, scegli il pulsante di aggiornamento.

- b. Scegli Create thing (Crea oggetto).

Si apre una finestra modale in cui è possibile scaricare i certificati e le chiavi a cui il core utilizza per connettersi. AWS IoT

9. Nella modalità Scarica certificati e chiavi, scarica i certificati del dispositivo.

 Important

Prima di scegliere Fine, scarica le risorse di sicurezza.

Esegui questa operazione:

- a. Per Certificato del dispositivo, scegli Scarica per scaricare il certificato del dispositivo.
- b. Per il file della chiave pubblica, scegli Scarica per scaricare la chiave pubblica per il certificato.
- c. Per File di chiave privata, scegli Scarica per scaricare il file di chiave privata per il certificato.
- d. Consulta [l'autenticazione del server](#) nella Guida per gli AWS IoT sviluppatori e scegli il certificato CA principale appropriato. Ti consigliamo di utilizzare gli endpoint Amazon Trust Services (ATS) e i certificati CA root ATS. In Certificati CA root, scegli Scarica per un certificato CA root.
- e. Seleziona Fatto.

Prendi nota dell'ID del certificato che è comune nei nomi dei file per il certificato e le chiavi del dispositivo. perché sarà necessaria in seguito.

Crea un AWS IoT Greengrass gruppo per il core

AWS IoT Greengrass i gruppi contengono impostazioni e altre informazioni sui relativi componenti, come dispositivi client, funzioni Lambda e connettori. Un gruppo definisce la configurazione di un core, incluso il modo in cui i suoi componenti possono interagire tra loro.

In questa sezione, crei un gruppo per il tuo core.

Tip

Per un esempio che utilizza l' AWS IoT Greengrass API per creare e distribuire un gruppo, consulta il repository [gg_group_setup](#) su. GitHub

Per creare un gruppo per il core

1. Passare alla [console AWS IoT](#).
2. In Gestisci, espandi i dispositivi Greengrass e scegli Gruppi (V1).

Note

Se non vedi il menu dei dispositivi Greengrass, passa a una Regione AWS che supporti. AWS IoT Greengrass V1 Per l'elenco delle regioni supportate, consulta [Endpoint e quote AWS IoT Greengrass V1](#) in Riferimenti generali di AWS. Devi [creare il dispositivo per AWS IoT il tuo core](#) in una regione in cui AWS IoT Greengrass V1 è disponibile.

3. Nella pagina dei gruppi Greengrass, scegli Crea gruppo.
4. Nella pagina del gruppo Create Greengrass, effettuate le seguenti operazioni:
 - a. Per il nome del gruppo Greengrass, inserisci un nome che descriva il gruppo, ad esempio. **MyGreengrassGroup**
 - b. Per Greengrass core, scegli AWS IoT ciò che hai creato in precedenza, ad esempio **MyGreengrassV1Core**.

La console seleziona automaticamente il certificato del dispositivo dell'oggetto.

- c. Seleziona Crea gruppo.

Installa ed esegui AWS IoT Greengrass sul dispositivo principale

Note

Questo tutorial fornisce istruzioni per eseguire il software AWS IoT Greengrass Core su un Raspberry Pi, ma puoi utilizzare qualsiasi dispositivo supportato.

In questa sezione, configuri, installi ed esegui il software AWS IoT Greengrass Core sul tuo dispositivo principale.

Per installare ed eseguire AWS IoT Greengrass

1. Dalla sezione [AWS IoT Greengrass Core software](#) di questa guida, scaricate il pacchetto di installazione del software AWS IoT Greengrass Core. Scegli il pacchetto che meglio si adatta all'architettura della CPU, alla distribuzione e al sistema operativo del tuo dispositivo core.
 - Per Raspberry Pi, scarica il pacchetto per l'architettura ARMv7L e il sistema operativo Linux.
 - Per un' EC2 istanza Amazon, scarica il pacchetto per l'architettura x86_64 e il sistema operativo Linux.
 - Per NVIDIA Jetson TX2, scarica il pacchetto per l'architettura Armv8 () e il sistema operativo Linux. AArch64
 - Per Intel Atom, scarica il pacchetto per l'architettura x86_64 e il sistema operativo Linux.
2. Nei passaggi precedenti, hai scaricato cinque file sul tuo computer:
 - `greengrass-OS-architecture-1.11.6.tar.gz`— Questo file compresso contiene il software AWS IoT Greengrass Core che viene eseguito sul dispositivo principale.
 - `certificateId-certificate.pem.crt`— Il file di certificato del dispositivo.
 - `certificateId-public.pem.key`— Il file della chiave pubblica del certificato del dispositivo.
 - `certificateId-private.pem.key`— Il file della chiave privata del certificato del dispositivo.

- AmazonRootCA1.pem— Il file Amazon Root Certificate Authority (CA).

In questa fase, trasferisci questi file dal computer al dispositivo principale. Esegui questa operazione:

- a. Se non conosci l'indirizzo IP del tuo dispositivo principale Greengrass, apri un terminale sul dispositivo principale ed esegui il seguente comando.

 Note

Questo comando potrebbe non restituire l'indirizzo IP corretto per alcuni dispositivi. Consulta la documentazione del dispositivo per recuperare l'indirizzo IP del dispositivo.

```
hostname -I
```

- b. Trasferisci questi file dal computer al dispositivo principale. Le fasi di trasferimento dei file variano a seconda del sistema operativo del computer. Scegli il sistema operativo per le fasi che mostrano come trasferire i file sul tuo dispositivo Raspberry Pi.

 Note

Per un Raspberry Pi, il nome utente predefinito è **pi** e la password predefinita è **raspberrypi**.

Per un NVIDIA Jetson TX2, il nome utente predefinito è **nvidia** e la password predefinita è **nvidia**.

Windows

Per trasferire i file compressi dal computer a un dispositivo principale Raspberry Pi, utilizza uno strumento come [WinSCP](#) oppure il comando [PuTTY](#) pscp. Per utilizzare il comando pscp, apri una finestra del prompt dei comandi sul computer ed esegui:

```
cd path-to-downloaded-files  
pscp -pw Pi-password greengrass-OS-architecture-1.11.6.tar.gz pi@IP-address:/home/pi
```

```
pscp -pw Pi-password certificateId-certificate.pem.crt pi@IP-address:/home/pi  
pscp -pw Pi-password certificateId-public.pem.key pi@IP-address:/home/pi  
pscp -pw Pi-password certificateId-private.pem.key pi@IP-address:/home/pi  
pscp -pw Pi-password AmazonRootCA1.pem pi@IP-address:/home/pi
```

 Note

Il numero di versione in questo comando deve corrispondere alla versione del pacchetto software AWS IoT Greengrass Core.

macOS

Per trasferire i file compressi dal Mac a un dispositivo principale Raspberry Pi, apri una finestra del terminale sul computer ed esegui i comandi seguenti. In genere *path-to-downloaded-files* è ~/Downloads.

 Note

Potrebbe esserti richiesto di immettere due password. In questo caso, la prima password è per il comando sudo del Mac e la seconda è la password per il Raspberry Pi.

```
cd path-to-downloaded-files  
scp greengrass-OS-architecture-1.11.6.tar.gz pi@IP-address:/home/pi  
scp certificateId-certificate.pem.crt pi@IP-address:/home/pi  
scp certificateId-public.pem.key pi@IP-address:/home/pi  
scp certificateId-private.pem.key pi@IP-address:/home/pi  
scp AmazonRootCA1.pem pi@IP-address:/home/pi
```

 Note

Il numero di versione in questo comando deve corrispondere alla versione del pacchetto software AWS IoT Greengrass Core.

UNIX-like system

Per trasferire i file compressi dal computer a un dispositivo principale Raspberry Pi, apri una finestra del terminale sul computer ed esegui i comandi seguenti:

```
cd path-to-downloaded-files
scp greengrass-OS-architecture-1.11.6.tar.gz pi@IP-address:/home/pi
scp certificateId-certificate.pem.crt pi@IP-address:/home/pi
scp certificateId-public.pem.key pi@IP-address:/home/pi
scp certificateId-private.pem.key pi@IP-address:/home/pi
scp AmazonRootCA1.pem pi@IP-address:/home/pi
```

Note

Il numero di versione di questo comando deve corrispondere alla versione del pacchetto software AWS IoT Greengrass Core.

Raspberry Pi web browser

Se hai utilizzato il browser web del Raspberry Pi per scaricare i file compressi, i file dovrebbero trovarsi nella ~/Downloads cartella del Raspberry Pi, ad esempio. /home/pi/Downloads Altrimenti, i file compressi dovrebbero trovarsi nella ~ cartella del Pi, ad esempio. /home/pi

3. Sul dispositivo principale Greengrass, apri un terminale e vai alla cartella che contiene il software e i AWS IoT Greengrass certificati Core. Sostituiscilo *path-to-transferred-files* con il percorso in cui hai trasferito i file sul dispositivo principale. Ad esempio, su un Raspberry Pi, cd /home/pi esegui.

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

4. Disimballate il software AWS IoT Greengrass Core sul dispositivo principale. Esegui il comando seguente per decomprimere l'archivio software trasferito sul dispositivo principale. Questo comando utilizza l'-C /argomento per creare la /greengrass cartella nella cartella principale del dispositivo principale.

```
sudo tar -xzf greengrass-OS-architecture-1.11.6.tar.gz -C /
```

 Note

Il numero di versione di questo comando deve corrispondere alla versione del pacchetto software AWS IoT Greengrass Core.

5. Spostate i certificati e le chiavi nella cartella del software AWS IoT Greengrass Core. Esegui i seguenti comandi per creare una cartella per i certificati e spostarvi i certificati e le chiavi. Sostituiscila *path-to-transferred-files* con il percorso in cui hai trasferito i file sul dispositivo principale e *certificateId* sostituiscila con l'ID del certificato nei nomi dei file. Ad esempio, su un Raspberry Pi, sostituisci *path-to-transferred-files* con **/home/pi**

```
sudo mv path-to-transferred-files/certificateId-certificate.pem.crt /greengrass/certs
sudo mv path-to-transferred-files/certificateId-public.pem.key /greengrass/certs
sudo mv path-to-transferred-files/certificateId-private.pem.key /greengrass/certs
sudo mv path-to-transferred-files/AmazonRootCA1.pem /greengrass/certs
```

6. Il software AWS IoT Greengrass Core utilizza un file di configurazione che specifica i parametri del software. Questo file di configurazione specifica i percorsi dei file di certificato e gli Cloud AWS endpoint da utilizzare. In questo passaggio, crei il file di configurazione del software AWS IoT Greengrass Core per il tuo core. Esegui questa operazione:
 - a. Ottieni l'Amazon Resource Name (ARN) per le tue attività principali. AWS IoT Esegui questa operazione:
 - i. Nella [AWS IoT console](#), in Gestione, in Dispositivi Greengrass, scegli Gruppi (V1).
 - ii. Nella pagina dei gruppi Greengrass, scegli il gruppo che hai creato in precedenza.
 - iii. In Panoramica, scegli Greengrass core.
 - iv. Nella pagina dei dettagli principali, copia l'ARN dell'AWS IoT oggetto e salvalo per utilizzarlo nel file di configurazione AWS IoT Greengrass Core.
 - b. Ottieni l'endpoint dei dati del AWS IoT dispositivo per la tua Account AWS regione corrente. I dispositivi utilizzano questo endpoint per connettersi a oggetti simili AWS . AWS IoT Esegui questa operazione:
 - i. Nella [AWS IoT console](#), scegli Impostazioni.
 - ii. In Device data endpoint, copia l'endpoint e salvalo per utilizzarlo nel file di configurazione AWS IoT Greengrass principale.

- c. Crea il file di configurazione del software AWS IoT Greengrass Core. Ad esempio, è possibile eseguire il comando seguente per utilizzare GNU nano per creare il file.

```
sudo nano /greengrass/config/config.json
```

Sostituisci il contenuto del file con il seguente documento JSON.

```
{
  "coreThing" : {
    "caPath": "AmazonRootCA1.pem",
    "certPath": "certificateId-certificate.pem.crt",
    "keyPath": "certificateId-private.pem.key",
    "thingArn": "arn:aws:iot:region:account-id:thing/MyGreengrassV1Core",
    "iotHost": "device-data-prefix-ats.iot.region.amazonaws.com",
    "ggHost": "greengrass-ats.iot.region.amazonaws.com",
    "keepAlive": 600
  },
  "runtime": {
    "cgroup": {
      "useSystemd": "yes"
    }
  },
  "managedRespawn": false,
  "crypto": {
    "caPath": "file:///greengrass/certs/AmazonRootCA1.pem",
    "principals": {
      "SecretsManager": {
        "privateKeyPath": "file:///greengrass/certs/certificateId-private.pem.key"
      },
      "IoTCertificate": {
        "privateKeyPath": "file:///greengrass/certs/certificateId-private.pem.key",
        "certificatePath": "file:///greengrass/certs/certificateId-certificate.pem.crt"
      }
    }
  }
}
```

Successivamente, esegui queste operazioni:

- Se hai scaricato un certificato Amazon Root CA diverso da Amazon Root CA 1, sostituisci ogni istanza di *AmazonRootCA1.pem* con il nome del file Amazon root CA.
- Sostituisci ogni istanza di *certificateId* con l'ID del certificato nel nome del certificato e dei file chiave.
- Sostituiscilo *arn:aws:iot:region:account-id:thing/MyGreengrassV1Core* con l'ARN del file principale che hai salvato in precedenza.
- Sostituiscilo *MyGreengrassV1core* con il nome dell'oggetto del tuo core.
- *device-data-prefix-ats.iot.region.amazonaws.com* Sostituiscilo con l'endpoint dei dati del AWS IoT dispositivo che hai salvato in precedenza.
- Sostituisci *region* con il tuo. Regione AWS

Per ulteriori informazioni sulle opzioni di configurazione che è possibile specificare in questo file di configurazione, vedere [AWS IoT Greengrass file di configurazione principale](#).

7. Assicurati che il dispositivo principale sia connesso a Internet. Quindi, avvia AWS IoT Greengrass dal tuo dispositivo principale.

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

Viene visualizzato un messaggio `Greengrass successfully started`. Prendere nota del PID.

Note

Per configurare il dispositivo principale in modo che si avvii AWS IoT Greengrass all'avvio del sistema, consulta [the section called “Lancio di Greengrass all'avvio del sistema”](#).

È possibile eseguire il comando seguente per confermare il funzionamento del software AWS IoT Greengrass Core (daemon Greengrass). Sostituisci *PID-number* con il tuo PID:

```
ps aux | grep PID-number
```

Dovresti visualizzare una voce per il PID con un percorso per il daemon Greengrass in esecuzione (ad esempio, /greengrass/ggc/packages/1.11.6/bin/daemon). Se riscontri problemi di avvio AWS IoT Greengrass, consulta [Risoluzione dei problemi](#).

Modulo 3 (parte 1): funzioni Lambda su AWS IoT Greengrass

Questo modulo mostra come creare e implementare una funzione Lambda che invia messaggi MQTT dal AWS IoT Greengrass dispositivo principale. Il modulo descrive le configurazioni delle funzioni Lambda, gli abbonamenti utilizzati per consentire la messaggistica MQTT e le implementazioni su un dispositivo principale.

[Il Modulo 3 \(parte 2\)](#) illustra le differenze tra le funzioni Lambda on-demand e quelle a lunga durata eseguite sul core. AWS IoT Greengrass

Prima di iniziare, assicurati di aver completato il [Modulo 1 e il Modulo 2](#) e di avere un dispositivo principale funzionante AWS IoT Greengrass .

Tip

Oppure, per utilizzare uno script che configuri automaticamente il dispositivo core, consulta [the section called “Avvio rapido: configurazione dispositivo Greengrass”](#). Lo script può anche creare e distribuire la funzione Lambda utilizzata in questo modulo.

Il completamento di questo modulo richiede circa 30 minuti.

Argomenti

- [Creare e impacchettare una funzione Lambda](#)
- [Configurare la funzione Lambda per AWS IoT Greengrass](#)
- [Distribuzione delle configurazioni cloud su un dispositivo core di Greengrass](#)
- [Verifica che la funzione Lambda sia in esecuzione sul dispositivo principale](#)

Creare e impacchettare una funzione Lambda

La funzione Python Lambda di esempio in questo modulo utilizza Core [AWS IoT Greengrass SDK for Python per](#) pubblicare messaggi MQTT.

In questa fase, si:

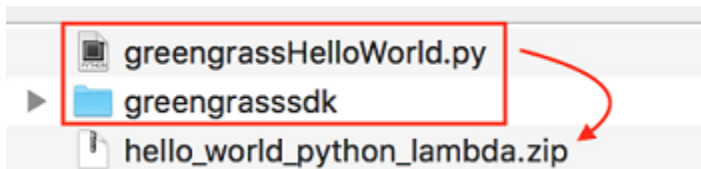
- Scarica il AWS IoT Greengrass Core SDK per Python sul tuo computer (non AWS IoT Greengrass sul dispositivo principale).
- Crea un pacchetto di distribuzione della funzione Lambda che contenga il codice della funzione e le dipendenze.
- Usa la console Lambda per creare una funzione Lambda e caricare il pacchetto di distribuzione.
- Pubblica una versione della funzione Lambda e crea un alias che punti alla versione.

Per completare questo modulo, Python 3.7 deve essere installato sul dispositivo principale.

1. Dalla pagina dei download di [AWS IoT Greengrass Core SDK](#), scarica AWS IoT Greengrass Core SDK per Python sul tuo computer.
2. Decomprimi il pacchetto scaricato per ottenere il codice della funzione Lambda e l'SDK.

La funzione Lambda in questo modulo utilizza:

- Il file `greengrassHelloWorld.py` in `examples\HelloWorld`. Questo è il codice della tua funzione Lambda. Ogni cinque secondi la funzione pubblica uno tra due possibili messaggi all'argomento `hello/world`.
 - La cartella `greengrasssdk`. Questo è l'SDK.
3. Copiare la cartella `greengrasssdk` nella cartella `HelloWorld` contenente `greengrassHelloWorld.py`.
 4. Per creare il pacchetto di distribuzione della funzione Lambda, salva `greengrassHelloWorld.py` e la `greengrasssdk` cartella in un file compresso denominato `zip hello_world_python_lambda.zip`. Il file `py` e la cartella `greengrasssdk` devono trovarsi nella radice della directory.



Sui sistemi di tipo UNIX (tra cui il terminale Mac), puoi utilizzare il seguente comando per creare il pacchetto del file e della cartella:

```
zip -r hello_world_python_lambda.zip greengrasssdk greengrassHelloWorld.py
```

Note

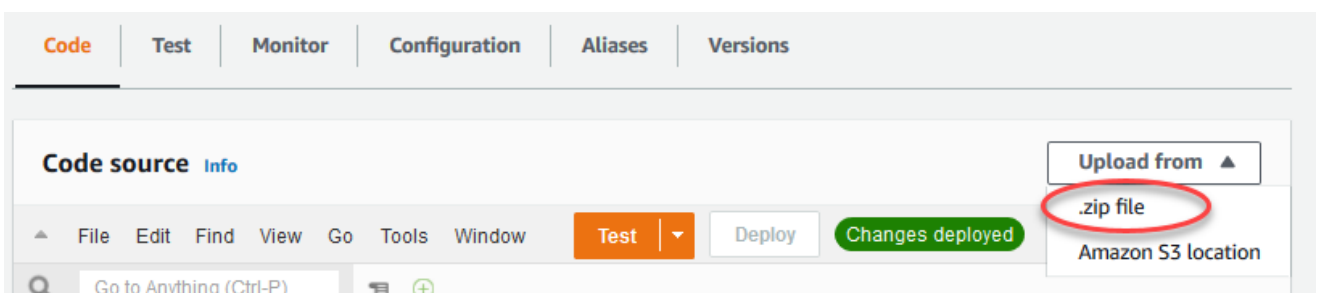
In base alla distribuzione, potrebbe essere necessario installare prima zip (ad esempio, eseguendo `sudo apt-get install zip`). Il comando di installazione per la distribuzione potrebbe essere diverso.

Ora sei pronto per creare la tua funzione Lambda e caricare il pacchetto di distribuzione.

5. Apri la console Lambda e scegli Crea funzione.
6. Scegli Crea da zero.
7. Dai alla funzione il nome **Greengrass_HelloWorld** e imposta i campi rimanenti come segue:
 - In Runtime, scegliere Python 3.7.
 - Per le autorizzazioni, mantieni l'impostazione predefinita. Questo crea un ruolo di esecuzione che concede le autorizzazioni Lambda di base. Questo ruolo non viene utilizzato da AWS IoT Greengrass

Scegli Crea funzione.

8. Carica il pacchetto di distribuzione della funzione Lambda:
 - a. Nella scheda Codice, in Codice sorgente, scegli Carica da. Dal menu a discesa, scegli un file.zip.



- b. Scegli Carica, quindi scegli il pacchetto di `hello_world_python_lambda.zip` distribuzione. Quindi, scegliere Save (Salva).
 - c. Nella scheda Codice della funzione, in Impostazioni di runtime, scegliete Modifica, quindi immettete i seguenti valori.

- In Runtime, scegliere Python 3.7.
- In Handler (Gestore), immetti **greengrassHelloWorld.function_handler**

Runtime settings [Info](#)

Runtime

Python 3.7 ▼

Handler [Info](#)

greengrassHelloWorld.function_handler

- d. Seleziona Salva.

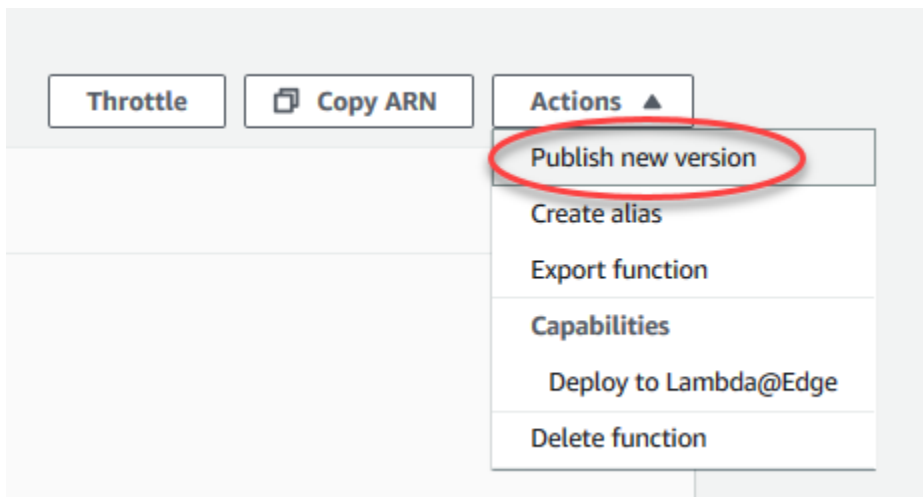
Note

Il pulsante Test sulla console non funziona con questa funzione AWS Lambda . Il AWS IoT Greengrass Core SDK non contiene moduli necessari per eseguire le funzioni Greengrass Lambda in modo indipendente nella console. AWS Lambda Questi moduli (ad esempio,greengrass_common) vengono forniti alle funzioni dopo essere stati distribuiti nel core Greengrass.

9.

Pubblica la funzione Lambda:

- a. Dal menu Azioni nella parte superiore della pagina, scegli Pubblica nuova versione.



- b. Per Version description (Descrizione versione), immettere **First version**, quindi scegliere Publish (Pubblica).

Publish new version from \$LATEST ✕

Publishing a new version will save a "snapshot" of the code and configuration of the \$LATEST version. You will be unable to edit the new version's code. Please click to confirm.

Version description

Cancel Publish

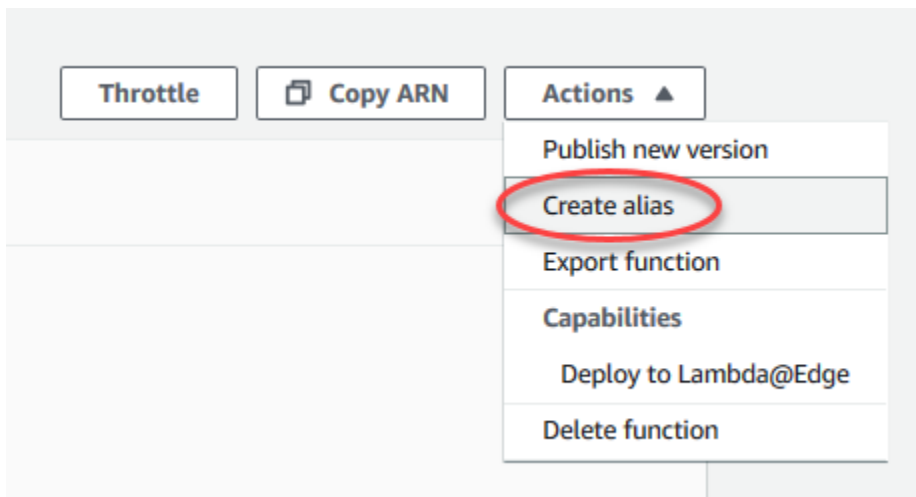
10.

[Crea un alias per la versione della funzione Lambda:](#)

Note

I gruppi Greengrass possono fare riferimento a una funzione Lambda tramite alias (consigliato) o per versione. L'utilizzo di un alias semplifica la gestione degli aggiornamenti del codice perché non è necessario modificare la tabella di sottoscrizione o la definizione del gruppo quando il codice della funzione viene aggiornato. Invece, è sufficiente indirizzare l'alias alla nuova versione della funzione.

- a. Dal menu Azioni nella parte superiore della pagina, scegli Crea alias.



- b. Assegna un nome all'alias **GG_HelloWorld**, imposta la versione su **1** (che corrisponde alla versione che hai appena pubblicato), quindi scegli Salva.

Note

AWS IoT Greengrass non supporta gli alias Lambda per le versioni \$LATEST.

Create alias

Alias configuration

An alias is a pointer to one or two versions. Choose each version that you want the alias to point to.

Name

Description - optional

Version

► Weighted alias

Cancel

Save

Configurare la funzione Lambda per AWS IoT Greengrass

Ora sei pronto per configurare la tua funzione Lambda per. AWS IoT Greengrass

In questa fase, si:

- Usa la AWS IoT console per aggiungere la funzione Lambda al tuo gruppo Greengrass.
 - Configura le impostazioni specifiche del gruppo per la funzione Lambda.
 - Aggiungi un abbonamento al gruppo che consente alla funzione Lambda di pubblicare messaggi MQTT. AWS IoT
 - Configurano le impostazioni del log locale per il gruppo.
1. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).
 2. In Gruppi Greengrass, scegli il gruppo che hai creato nel [Modulo 2](#).
 3. Nella pagina di configurazione del gruppo, scegli la scheda Funzioni Lambda, quindi scorri verso il basso fino alla sezione Le mie funzioni Lambda e scegli Aggiungi funzione Lambda.
 4. Seleziona il nome della funzione Lambda che hai creato nel passaggio precedente (Greengrass_HelloWorld, non il nome dell'alias).
 5. Per la versione, scegliete Alias: GG_. HelloWorld
 6. Nella sezione di configurazione della funzione Lambda, apporta le seguenti modifiche:
 - Imposta l'utente e il gruppo di sistema su Use group default.
 - Imposta la containerizzazione della funzione Lambda su Use group default.
 - Imposta Timeout a 25 secondi. Questa funzione Lambda dorme per 5 secondi prima di ogni chiamata.
 - Per Pinned, scegli True.

Note

Una funzione Lambda di lunga durata (o bloccata) si avvia automaticamente dopo l'avvio e continua AWS IoT Greengrass a funzionare nel proprio contenitore. Ciò è in

contrasto con una funzione Lambda su richiesta, che si avvia quando viene richiamata e si interrompe quando non ci sono più attività da eseguire. Per ulteriori informazioni, consulta [the section called “Configurazione del ciclo di vita”](#).

7. Scegli la funzione Aggiungi Lambda per salvare le modifiche. Per informazioni sulle proprietà della funzione Lambda, vedere. [the section called “Controllo dell'esecuzione della funzione Greengrass Lambda”](#)

Quindi, crea un abbonamento che consenta alla funzione Lambda di inviare messaggi [MQTT](#) a AWS IoT Core

Una funzione Greengrass Lambda può scambiare messaggi MQTT con:

- [Dispositivi](#) nel gruppo Greengrass.
- [Connettori](#) nel gruppo.
- Altre funzioni Lambda del gruppo.
- AWS IoT Core.
- Il servizio shadow locale. Per ulteriori informazioni, consulta [the section called “Modulo 5: interazione con dispositivi ombra”](#).

Il gruppo utilizza le sottoscrizioni per controllare in che modo queste entità possono comunicare tra loro. Le sottoscrizioni forniscono interazioni prevedibili e un livello di sicurezza.

Una sottoscrizione è costituita da origine, destinazione e argomento. L'origine è l'autore del messaggio. La destinazione è il ricevente del messaggio. L'argomento consente di filtrare i dati inviati dall'origine alla destinazione. L'origine o la destinazione può essere un dispositivo Greengrass, una funzione Lambda, un connettore, un'ombra del dispositivo o AWS IoT Core

Note

Una sottoscrizione è diretta, nel senso che il flusso di messaggi ha una direzione specifica: dall'origine alla destinazione. Per consentire la comunicazione bidirezionale, è necessario configurare due sottoscrizioni.

 Note

Attualmente, il filtro degli argomenti di sottoscrizione non consente più di un singolo + carattere in un argomento. Il filtro degli argomenti consente un solo # carattere alla fine di un argomento.

La funzione `Greengrass_HelloWorld` Lambda invia messaggi solo all'`hello/world` argomento in AWS IoT Core, quindi è sufficiente creare un solo abbonamento dalla funzione Lambda a. AWS IoT Core Questa operazione viene eseguita nella fase successiva.

8. Nella pagina di configurazione del gruppo, scegli la scheda Abbonamenti, quindi scegli Aggiungi abbonamento.

Per un esempio che mostra come creare un abbonamento utilizzando il AWS CLI, vedi [create-subscription-definition](#) nel AWS CLI Command Reference.

9. Nel tipo di origine, scegli la funzione Lambda e, per la sorgente, scegli `Greengrass_. HelloWorld`
10. Per il tipo di Target, scegli Service e, per Target, seleziona IoT Cloud.
11. Per il filtro Argomento **hello/world**, inserisci, quindi scegli Crea abbonamento.
12. Configurare le impostazioni di registrazione del gruppo. In questo tutorial, configurerai i componenti di AWS IoT Greengrass sistema e le funzioni Lambda definite dall'utente per scrivere i log nel file system del dispositivo principale.
 - a. Nella pagina di configurazione del gruppo, scegli la scheda Registri.
 - b. Nella sezione Configurazione dei registri locali, scegli Modifica.
 - c. Nella finestra di dialogo Modifica configurazione dei log locali, mantieni i valori predefiniti sia per i livelli di registro che per le dimensioni di archiviazione, quindi scegli Salva.

È possibile utilizzare i registri per risolvere eventuali problemi che potrebbero verificarsi durante l'esecuzione di questa esercitazione. Durante la risoluzione dei problemi, è possibile modificare temporaneamente il livello di registrazione in Debug. Per ulteriori informazioni, consulta [the section called "Accesso ai log del file system"](#).

13. Se il runtime Java 8 non è installato sul dispositivo core, è necessario installarlo o disabilitare Gestore di flussi.

 Note

Questo tutorial non utilizza Gestore di flussi, ma utilizza il flusso di lavoro Creazione gruppo predefinito che abilita Gestore di flussi per impostazione predefinita. Se Gestore di flussi è abilitato ma Java 8 non è installato, la distribuzione del gruppo ha esito negativo. Per ulteriori informazioni, consulta i [requisiti per Gestore di flussi](#).

Per disabilitare Gestore di flussi:

- a. Nella pagina delle impostazioni del gruppo, scegli la scheda Funzioni Lambda.
- b. Nella sezione Funzioni System Lambda, seleziona Stream manager e scegli Modifica.
- c. Scegliere Abilita, quindi Salva.

Distribuzione delle configurazioni cloud su un dispositivo core di Greengrass

1. Assicurati che il dispositivo core di Greengrass sia connesso a Internet. Ad esempio, provare a passare correttamente a una pagina Web.
2. Assicurati che il daemon Greengrass sia in esecuzione sul dispositivo core. Nel terminale del dispositivo principale, esegui i seguenti comandi per verificare se il demone è in esecuzione e avviarlo, se necessario.
 - a. Per controllare se il daemon è in esecuzione:

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

Se l'output contiene una voce root per /greengrass/ggc/packages/1.11.6/bin/daemon, allora il daemon è in esecuzione.

- b. Per avviare il demone:

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

Ora sei pronto per implementare la funzione Lambda e le configurazioni di abbonamento sul tuo dispositivo principale Greengrass.

3. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).
4. In Gruppi Greengrass, scegli il gruppo che hai creato nel [Modulo 2](#).
5. Nella pagina di configurazione del gruppo, scegli Deploy.
6. Nella scheda Funzioni Lambda, nella sezione Funzioni System Lambda, scegli Rilevatore IP.
7. Scegli Modifica e seleziona Rileva e sostituisci automaticamente gli endpoint del broker MQTT. Questo consente ai dispositivi di acquisire automaticamente informazioni di base sulla connettività, come, ad esempio indirizzo IP, DNS e numero della porta. È consigliato il rilevamento automatico, ma AWS IoT Greengrass supporta anche endpoint specifici manualmente. Ti viene chiesto il metodo di individuazione solo la prima volta che il gruppo viene distribuito.

La prima distribuzione potrebbe richiedere alcuni minuti. Al termine della distribuzione, dovresti visualizzare Successfully completed (Completata con successo) nella colonna Status (Stato) della pagina Deployments (Distribuzioni):

 Note

Lo stato della distribuzione è visualizzato anche sotto il nome del gruppo nell'intestazione di pagina.

Per la risoluzione dei problemi, consultare [Risoluzione dei problemi](#).

Verifica che la funzione Lambda sia in esecuzione sul dispositivo principale

1. Dal pannello di navigazione della [AWS IoT console](#), in Test, scegli MQTT test client.
2. Scegli la scheda Iscriviti all'argomento.
3. Entra **hello/world** nel filtro Argomento ed espandi la Configurazione aggiuntiva.
4. Inserisci le informazioni elencate in ciascuno dei seguenti campi:
 - Per Quality of Service (Qualità del servizio), scegli 0.
 - Per MQTT payload display (Visualizzazione payload MQTT), scegli Display payloads as strings (Visualizza payload come stringhe).

5. Scegliere Subscribe (Effettua sottoscrizione).

Supponendo che la funzione Lambda sia in esecuzione sul dispositivo, pubblica messaggi simili ai seguenti nell'argomento: `hello/world`

The screenshot shows the AWS IoT Greengrass console interface. On the left, there's a sidebar with 'Subscriptions' selected. The main area shows a subscription for the topic 'hello/world'. Above the subscription details are buttons for 'Pause', 'Clear', 'Export', and 'Edit'. The subscription details include a dropdown for the topic 'hello/world' and a timestamp 'April 29, 2021, 17:35:40 (UTC-0400)'. Below this, a JSON message is displayed:

```
{  "message": "Hello world! Sent from Greengrass Core running on platform: Linux-4.9.30-v7+-armv7l-with-debian-8.0"}
```

Sebbene la funzione Lambda continui a inviare messaggi MQTT all'argomento `hello/world`, non arrestate il demone. AWS IoT Greengrass I restanti moduli vengono scritti presumendo che sia in esecuzione.

È possibile eliminare la funzione e la sottoscrizione dal gruppo:

- Nella pagina di configurazione dei gruppi, nella scheda Funzioni Lambda, seleziona la funzione Lambda che desideri rimuovere e scegli Rimuovi.
- Nella pagina di configurazione dei gruppi, nella scheda Sottoscrizioni, scegli l'abbonamento, quindi scegli Elimina.

La funzione e la sottoscrizione vengono eliminate dal core durante la successiva distribuzione di gruppo.

Modulo 3 (parte 2): funzioni Lambda su AWS IoT Greengrass

Questo modulo esplora le differenze tra le funzioni Lambda on-demand e quelle a lunga durata eseguite sul core. AWS IoT Greengrass

Prima di iniziare, eseguire lo script di [installazione dispositivo Greengrass](#) o assicurarsi di aver completato il [modulo 1](#), il [modulo 2](#), e il [modulo 3 \(parte 1\)](#).

Il completamento di questo modulo richiede circa 30 minuti.

Argomenti

- [Crea e impacchetta la funzione Lambda](#)
- [Configura funzioni Lambda di lunga durata per AWS IoT Greengrass](#)
- [Prova le funzioni Lambda di lunga durata](#)
- [Testa le funzioni Lambda su richiesta](#)

Crea e impacchetta la funzione Lambda

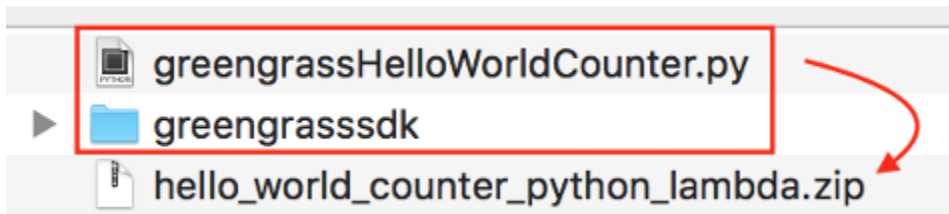
In questa fase, si:

- Crea un pacchetto di distribuzione della funzione Lambda che contenga il codice della funzione e le dipendenze.
- Usa la console Lambda per creare una funzione Lambda e caricare il pacchetto di distribuzione.
- Pubblica una versione della funzione Lambda e crea un alias che punti alla versione.

1. Sul tuo computer, vai al AWS IoT Greengrass Core SDK per Python che hai scaricato ed estratto [the section called “Creare e impacchettare una funzione Lambda”](#) nel Modulo 3-1.

La funzione Lambda in questo modulo utilizza:

- Il file `greengrassHelloWorldCounter.py` in `examples\HelloWorldCounter`. Questo è il codice della tua funzione Lambda.
 - La cartella `greengrasssdk`. Questo è l'SDK.
2. Crea un pacchetto di distribuzione della funzione Lambda:
 - a. Copiare la cartella `greengrasssdk` nella cartella `HelloWorldCounter` contenente `greengrassHelloWorldCounter.py`.
 - b. Salvare `greengrassHelloWorldCounter.py` e la cartella `greengrasssdk` in un file zip denominato `hello_world_counter_python_lambda.zip`. Il file py e la cartella `greengrasssdk` devono trovarsi nella radice della directory.



Sui sistemi di tipo UNIX (tra cui il terminale Mac) che hanno zip installato, puoi utilizzare il seguente comando per creare il pacchetto del file e della cartella:

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

Ora sei pronto per creare la tua funzione Lambda e caricare il pacchetto di distribuzione.

3. Apri la console Lambda e scegli Crea funzione.
4. Scegli Crea da zero.
5. Dai alla funzione il nome **Greengrass_HelloWorld_Counter** e imposta i campi rimanenti come segue:
 - In Runtime, scegliere Python 3.7.
 - Per le autorizzazioni, mantieni l'impostazione predefinita. Questo crea un ruolo di esecuzione che concede le autorizzazioni Lambda di base. Questo ruolo non viene utilizzato da AWS IoT Greengrass. In alternativa, puoi riutilizzare il ruolo creato nel Modulo 3-1.

Scegli Crea funzione.

Basic information

Function name
Enter a name that describes the purpose of your function.

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime [Info](#)
Choose the language to use to write your function.

Python 3.7 ▼

Permissions [Info](#)
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

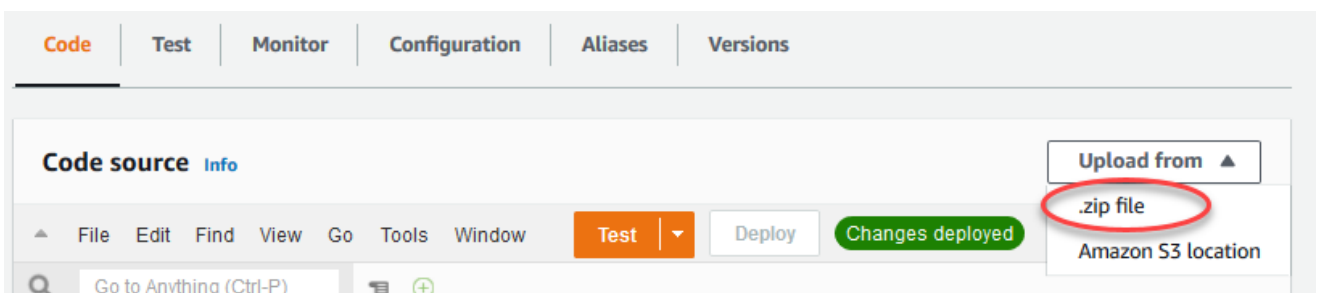
► **Change default execution role**

► **Advanced settings**

Cancel **Create function**

6. Carica il tuo pacchetto di implementazione della funzione Lambda.

- a. Nella scheda Codice, in Codice sorgente, scegli Carica da. Dal menu a discesa, scegli un file.zip.



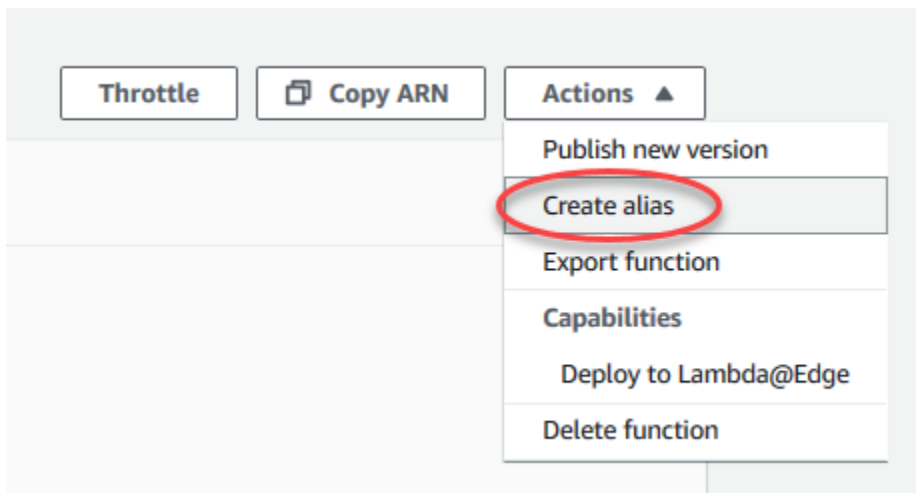
- b. Scegli Carica, quindi scegli il pacchetto di `hello_world_counter_python_lambda.zip` distribuzione. Quindi, scegliere Save (Salva).
- c. Nella scheda Codice della funzione, in Impostazioni di runtime, scegliete Modifica, quindi immettete i seguenti valori.
- In Runtime, scegliere Python 3.7.

- In Handler (Gestore), immetti **greengrassHelloWorldCounter.function_handler**
- d. Seleziona Salva.

 Note

Il pulsante Test sulla AWS Lambda console non funziona con questa funzione. Il AWS IoT Greengrass Core SDK non contiene moduli necessari per eseguire le funzioni Greengrass Lambda in modo indipendente nella console. AWS Lambda Questi moduli (ad esempio,greengrass_common) vengono forniti alle funzioni dopo essere stati distribuiti nel core Greengrass.

7. Pubblicare la prima versione della funzione.
- a. Dal menu Azioni nella parte superiore della pagina, scegli **Pubblica nuova versione**. In Version description (Descrizione versione), immetti **First version**.
 - b. Seleziona Publish (Pubblica).
8. Crea un alias per la versione della funzione.
- a. Dal menu Azioni nella parte superiore della pagina, scegli **Crea alias**.



- b. Per Nome, immetti **GG_HW_Counter**.
- c. In Version (Versione), selezionare 1.
- d. Seleziona Salva.

Create alias

Alias configuration

An alias is a pointer to one or two versions. Choose each version that you want the alias to point to.

Name

Description - optional

Version

1 ▼

► Weighted alias

Cancel Save

Gli alias creano una singola entità per la funzione Lambda a cui i dispositivi Greengrass possono abbonarsi. In questo modo, non è necessario aggiornare gli abbonamenti con nuovi numeri di versione della funzione Lambda ogni volta che la funzione viene modificata.

Configura funzioni Lambda di lunga durata per AWS IoT Greengrass

Ora sei pronto per configurare la tua funzione Lambda per AWS IoT Greengrass

1. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).
2. In Gruppi Greengrass, scegli il gruppo che hai creato nel [Modulo 2](#).
3. Nella pagina di configurazione del gruppo, scegli la scheda Funzioni Lambda, quindi in Le mie funzioni Lambda, scegli Aggiungi.
4. Per la funzione Lambda, scegli HelloWorldGreengrass__Counter.
5. Per la versione della funzione Lambda, scegli l'alias della versione che hai pubblicato.
6. Per Timeout (secondi), immettere. **25** Questa funzione Lambda dorme per 20 secondi prima di ogni chiamata.

7. Per Pinned, scegli True.
8. Mantieni i valori predefiniti per tutti gli altri campi e scegli Aggiungi funzione Lambda.

Prova le funzioni Lambda di lunga durata

Una funzione Lambda di [lunga durata](#) si avvia automaticamente AWS IoT Greengrass all'avvio del core e viene eseguita in un singolo contenitore (o sandbox). Qualsiasi variabile o logica di preelaborazione definite al di fuori del gestore della funzione sono conservate per ogni richiamo del gestore della funzione. Chiamate multiple del gestore della funzione vengono messe in coda finché le chiamate precedenti non sono state eseguite.

Il codice `greengrassHelloWorldCounter.py` utilizzato in questo modulo definisce una variabile `my_counter` al di fuori del gestore della funzione.

Note

Puoi visualizzare il codice nella AWS Lambda console o nel [AWS IoT Greengrass Core SDK for GitHub Python on](#).

In questo passaggio, si creano abbonamenti che consentono la funzione Lambda AWS IoT e lo scambio di messaggi MQTT. Quindi si procede alla distribuzione del gruppo e al test della funzione.

1. Nella pagina di configurazione del gruppo, scegli Sottoscrizioni, quindi scegli Aggiungi.
2. In Tipo di origine, scegli Funzione Lambda, quindi scegli HelloWorldGreengrass_ _Counter.
3. In Tipo di destinazione, scegli Servizio, scegli IoT Cloud.
4. In Topic filter (Filtro argomento), immettere **hello/world/counter**.
5. Scegli Crea sottoscrizione.

Questo abbonamento singolo va in una sola direzione: dalla funzione Greengrass_HelloWorld_Counter Lambda a. AWS IoT Per richiamare (o attivare) questa funzione Lambda dal cloud, devi creare un abbonamento nella direzione opposta.

6. Segui i passaggi da 1 a 5 per aggiungere un altro abbonamento che utilizzi i seguenti valori. Questo abbonamento consente alla funzione Lambda di ricevere messaggi da. AWS IoT Si utilizza questo abbonamento quando si invia un messaggio dalla AWS IoT console che richiama la funzione.

- Come fonte, scegli Servizio, quindi scegli IoT Cloud.
- Per la destinazione, scegli la funzione Lambda, quindi scegli HelloWorldGreengrass_
_Counter.
- Per il filtro di argomenti, immetti **hello/world/counter/trigger**.

In questo filtro dell'argomento viene usata l'estensione `/trigger` perché hai creato due sottoscrizioni ed è necessario che non interferiscano tra loro.

7. Assicuratevi che il demone Greengrass sia in esecuzione, come descritto in. [Distribuire configurazioni cloud su un dispositivo core](#)
8. Nella pagina di configurazione del gruppo, scegliete Deploy.
9. Una volta completata la distribuzione, torna alla home page della AWS IoT console e scegli Test.
10. Configura i campi seguenti:
 - Per Argomento sottoscrizione, immetti **hello/world/counter**.
 - Per Quality of Service (Qualità del servizio), scegli 0.
 - Per MQTT payload display (Visualizzazione payload MQTT), scegli Display payloads as strings (Visualizza payload come stringhe).
11. Scegliere Subscribe (Effettua sottoscrizione).

A differenza della [Parte 1](#) di questo modulo, non dovresti visualizzare messaggi dopo aver effettuato la sottoscrizione a `hello/world/counter`. Questo perché il codice `greengrassHelloWorldCounter.py` che pubblica nell'argomento `hello/world/counter` si trova all'interno del gestore della funzione, che viene eseguito solo quando viene richiamata la funzione.

In questo modulo, avete configurato la funzione `Greengrass_HelloWorld_Counter` Lambda per essere richiamata quando riceve un messaggio MQTT sull'argomento. `hello/world/counter/trigger`

L'abbonamento `Greengrass_HelloWorld_Counter` to IoT Cloud consente la funzione di inviare messaggi AWS IoT sull'argomento. `hello/world/counter` L'abbonamento IoT Cloud to `Greengrass_HelloWorld_Counter` consente di AWS IoT inviare messaggi alla funzione sull'argomento. `hello/world/counter/trigger`

12. Per testare il ciclo di vita di lunga durata, richiama la funzione Lambda pubblicando un messaggio sull'argomento. `hello/world/counter/trigger` Puoi usare il messaggio predefinito.

Subscribe to a topic

Publish to a topic

Topic name

The topic name identifies the message. The message payload will be published to this topic with a Quality of Service (QoS) of 0.

Message payload

Additional configuration

Publish

Note

La funzione `Greengrass_HelloWorld_Counter` ignora il contenuto dei messaggi ricevuti. Esegue solamente il codice in `function_handler`, che invia un messaggio all'argomento `hello/world/counter`. Puoi rivedere questo codice dal [AWS IoT Greengrass Core SDK for GitHub Python on](#).

Ogni volta che un messaggio viene pubblicato nell'argomento `hello/world/counter/trigger`, la variabile `my_counter` viene incrementata. Questo conteggio delle chiamate viene visualizzato nei messaggi inviati dalla funzione Lambda. Poiché il gestore della funzione include un ciclo di sonno di 20 secondi (`time.sleep(20)`), l'attivazione ripetuta del gestore mette in coda le risposte provenienti dal core. AWS IoT Greengrass

The screenshot shows the 'Subscriptions' page in the AWS IoT Greengrass console. On the left, there's a sidebar with 'Subscriptions' and a list of subscriptions, including 'hello/world/counter'. The main area shows details for the 'hello/world/counter' subscription, with buttons for 'Pause', 'Clear', 'Export', and 'Edit'. Below this, there are three entries for the subscription, each showing a timestamp and a JSON message. The 'Invocation Count' is highlighted in red in each message.

Subscription	Timestamp	Message
hello/world/counter	May 03, 2021, 10:05:00 (UTC-0400)	<pre>{ "message": "Hello world! Sent from Greengrass Core running on platform: Linux-4.9.30-v7+-armv7l-with-Debian-8.0" }</pre> Invocation Count: 3
hello/world/counter	May 03, 2021, 10:04:40 (UTC-0400)	<pre>{ "message": "Hello world! Sent from Greengrass Core running on platform: Linux-4.9.30-v7+-armv7l-with-Debian-8.0" }</pre> Invocation Count: 2
hello/world/counter	May 03, 2021, 10:04:20 (UTC-0400)	<pre>{ "message": "Hello world! Sent from Greengrass Core running on platform: Linux-4.9.30-v7+-armv7l-with-Debian-8.0" }</pre> Invocation Count: 1

Testa le funzioni Lambda su richiesta

Una funzione Lambda [su richiesta](#) ha funzionalità simili a quelle di una AWS Lambda funzione basata su cloud. È possibile eseguire più chiamate di una funzione Lambda su richiesta in parallelo. Una chiamata della funzione Lambda crea un contenitore separato per elaborare le chiamate o riutilizza un contenitore esistente, se le risorse lo consentono. Qualsiasi variabile o preelaborazione definite al di fuori del gestore della funzione non vengono conservate quando vengono creati i container.

1. Nella pagina di configurazione del gruppo, scegli la scheda Funzioni Lambda.
2. In Funzioni My Lambda, scegli Greengrass_HelloWorld_CounterFunzione Lambda.
3. Sul Greengrass_HelloWorld_Counter nella pagina dei dettagli, scegli Modifica.
4. Per Bloccato, scegli False, quindi scegli Salva.

5. Nella pagina di configurazione del gruppo, scegli Deploy.
6. Una volta completata la distribuzione, torna alla home page della AWS IoT console e scegli Test.
7. Configura i campi seguenti:
 - Per Argomento sottoscrizione, immetti **hello/world/counter**.
 - Per Quality of Service (Qualità del servizio), scegli 0.
 - Per MQTT payload display (Visualizzazione payload MQTT), scegli Display payloads as strings (Visualizza payload come stringhe).

Subscribe to a topic | **Publish to a topic**

Topic filter [Info](#)
The topic filter describes the topic(s) to which you want to subscribe. The topic filter can include MQTT wildcard characters.

hello/world/counter

▼ **Additional configuration**

Number of messages to keep
The MQTT test client keeps this many of the most recent messages published to a topic that matches this topic filter.

100

Quality of service
When subscribing to a topic, quality of service 0 will be chosen by default.

☒ Quality of Service 0 - Message will be delivered at most once

☐ Quality of Service 1 - Message will be delivered at least once

MQTT payload display

☒ Display payloads as strings (more accurate)

☐ Auto-format JSON payloads (improves readability)

☐ Display raw payloads (displays binary data as hexadecimal values)

Subscribe

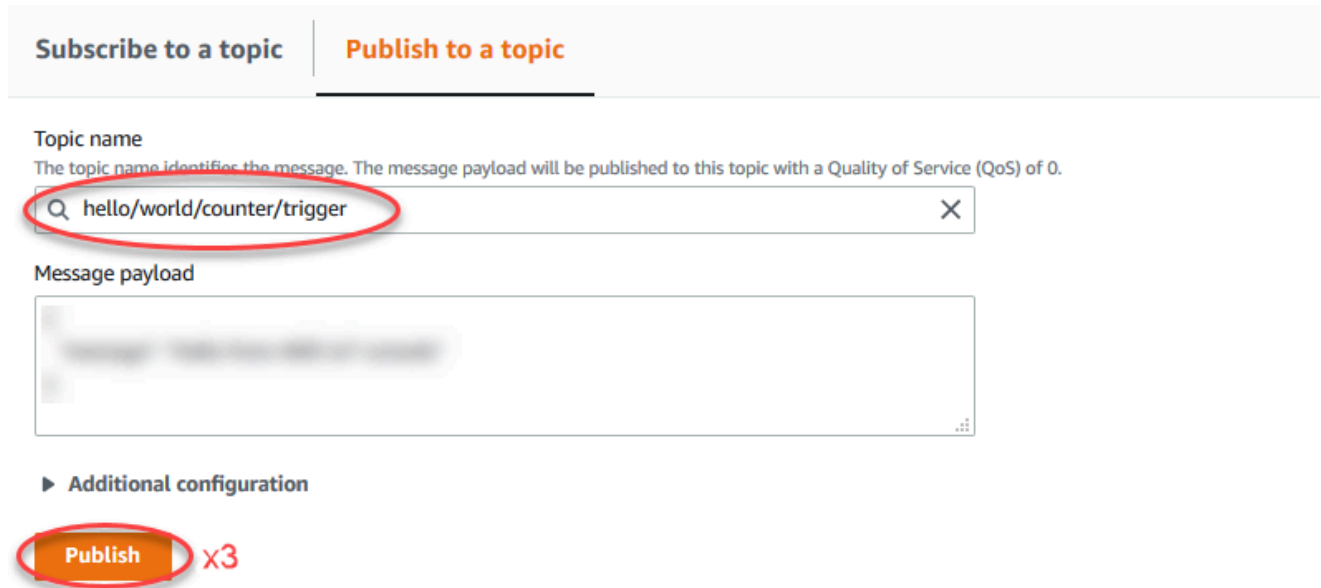
8. Scegliere Subscribe (Effettua sottoscrizione).

Note

Non dovresti visualizzare alcun messaggio dopo la sottoscrizione.

9. Per testare il ciclo di vita on demand, richiama la funzione pubblicando un messaggio nell'argomento hello/world/counter/trigger. Puoi usare il messaggio predefinito.

- a. Scegli **Pubblica** tre volte velocemente, entro cinque secondi da ogni pressione del pulsante.



The screenshot shows the AWS IoT Greengrass console interface for publishing a message. At the top, there are two tabs: 'Subscribe to a topic' and 'Publish to a topic'. The 'Publish to a topic' tab is selected. Below the tabs, there is a 'Topic name' field with the text 'hello/world/counter/trigger' entered. A red circle highlights this field. Below the topic name field is a 'Message payload' field, which is currently empty. At the bottom of the interface, there is a 'Publish' button, also circled in red, with a red 'x3' next to it, indicating that the button should be clicked three times.

Ogni pubblicazione richiama il gestore della funzione e crea un container per ogni chiamata. Il conteggio delle chiamate non viene incrementato per le tre volte che hai attivato la funzione perché ogni funzione Lambda su richiesta ha il proprio contenitore/sandbox.

The screenshot shows the AWS IoT Greengrass console interface. On the left, the 'Subscriptions' tab is selected, showing a list of subscriptions for the 'hello/world/counter' function. The main panel displays three subscription events, each with a timestamp and a JSON message. The 'Invocation Count: 1' field in each message is circled in red.

Subscriptions

hello/world/counter Pause Clear Export Edit

hello/world/counter ♡ ✕

▼ hello/world/counter May 03, 2021, 10:14:13 (UTC-0400)

```
{
  "message": "Hello world! Sent from Greengrass Core running on platform: Linux-4.9.30-v7+-armv7l-with-Debian-8.0. Invocation Count: 1"
}
```

▼ hello/world/counter May 03, 2021, 10:14:12 (UTC-0400)

```
{
  "message": "Hello world! Sent from Greengrass Core running on platform: Linux-4.9.30-v7+-armv7l-with-Debian-8.0. Invocation Count: 1"
}
```

▼ hello/world/counter May 03, 2021, 10:14:11 (UTC-0400)

```
{
  "message": "Hello world! Sent from Greengrass Core running on platform: Linux-4.9.30-v7+-armv7l-with-Debian-8.0. Invocation Count: 1"
}
```

- b. Dopo circa 30 secondi, scegli **Pubblica** nell'argomento. Il conteggio delle chiamate dovrebbe diventare 2. Questo indica che viene riutilizzato un container creato da una chiamata precedente e che le variabili di preelaborazione al di fuori del gestore della funzione sono state memorizzate.

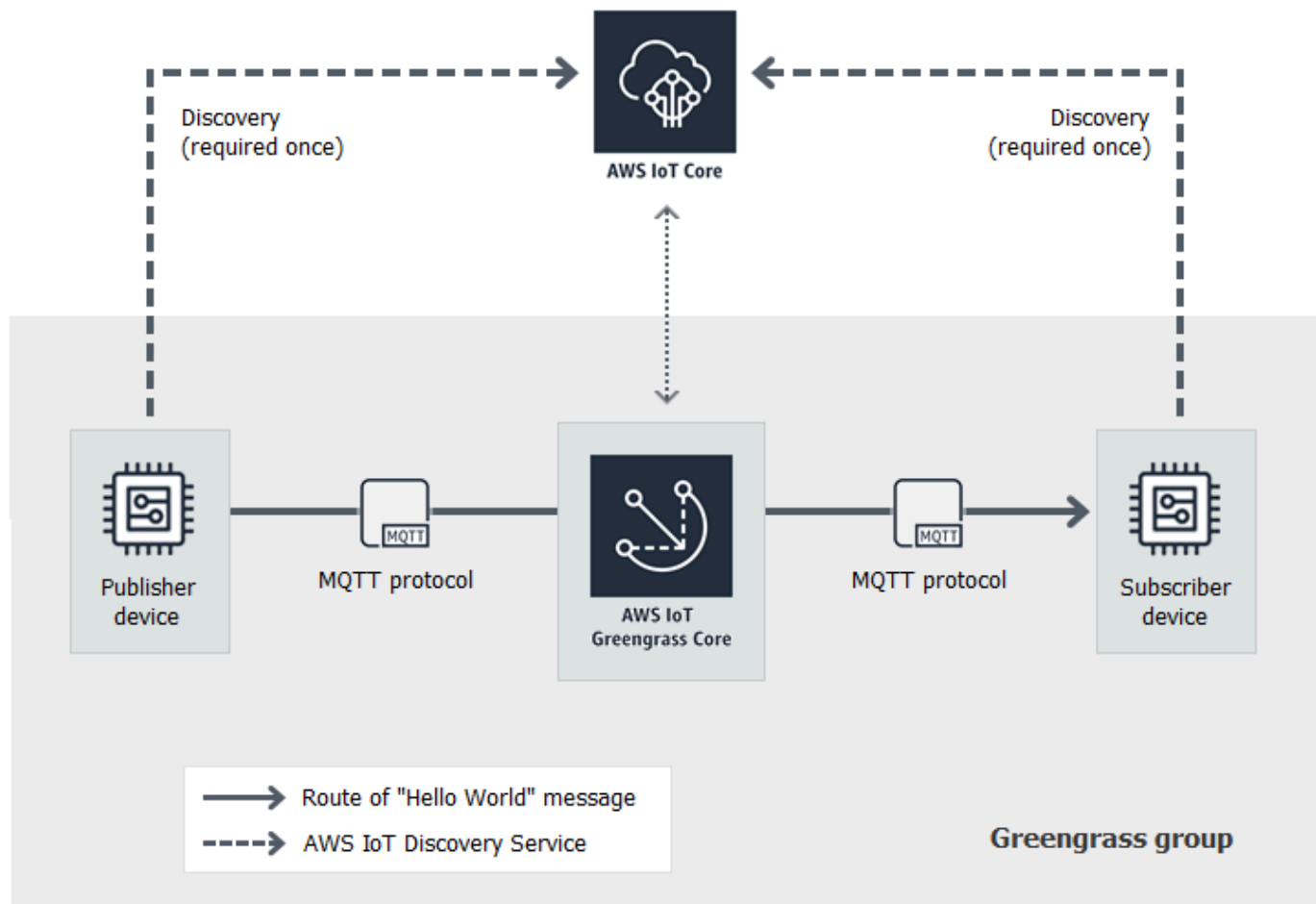
The screenshot displays the AWS IoT Greengrass console interface. On the left, a sidebar titled 'Subscriptions' lists the selected subscription 'hello/world/counter'. The main panel shows the details for this subscription, including a title bar with 'hello/world/counter' and buttons for 'Pause', 'Clear', 'Export', and 'Edit'. Below, two event logs are shown. Each log entry includes a timestamp and a JSON message. The first event, dated May 03, 2021, 10:14:40 (UTC-0400), contains the text 'Invocation Count: 2'. The second event, dated May 03, 2021, 10:14:11 (UTC-0400), contains the text 'Invocation Count: 1'. In both cases, the 'Invocation Count' value is circled in red.

Subscription	Timestamp	Message
hello/world/counter	May 03, 2021, 10:14:40 (UTC-0400)	<pre>{ "message": "Hello world! Sent from Greengrass Core running on platform: Linux-4.9.30-v7+-armv7l-with-Debian-8.0. Invocation Count: 2" }</pre>
hello/world/counter	May 03, 2021, 10:14:11 (UTC-0400)	<pre>{ "message": "Hello world! Sent from Greengrass Core running on platform: Linux-4.9.30-v7+-armv7l-with-Debian-8.0. Invocation Count: 1" }</pre>

Ora dovresti comprendere i due tipi di funzioni Lambda che possono essere eseguite sul AWS IoT Greengrass core. Il modulo successivo, [Modulo 4](#), mostra come i dispositivi IoT locali possono interagire in un AWS IoT Greengrass gruppo.

Modulo 4: Interazione con i dispositivi client in un gruppo AWS IoT Greengrass

Questo modulo mostra come i dispositivi IoT locali, chiamati dispositivi o dispositivi client, possono connettersi e comunicare con un dispositivo AWS IoT Greengrass principale. I dispositivi client che si connettono a un AWS IoT Greengrass core fanno parte di un AWS IoT Greengrass gruppo e possono partecipare al paradigma AWS IoT Greengrass di programmazione. In questo modulo, un dispositivo client invia un messaggio Hello World a un altro dispositivo client del gruppo Greengrass.



Prima di iniziare, eseguire lo script [Greengrass Device Setup](#) o completare il [modulo 1](#) e il [modulo 2](#). Questo modulo crea due dispositivi client simulati. Non sono necessari altri componenti o dispositivi.

Il completamento di questo modulo dovrebbe richiedere meno di 30 minuti.

Argomenti

- [Creare dispositivi client in un AWS IoT Greengrass gruppo](#)
- [Configurazione delle sottoscrizioni](#)
- [Installa il SDK per dispositivi AWS IoT for Python](#)
- [Test delle comunicazioni](#)

Creare dispositivi client in un AWS IoT Greengrass gruppo

In questo passaggio, aggiungi due dispositivi client al tuo gruppo Greengrass. Questo processo include la registrazione dei dispositivi come AWS IoT oggetti e la configurazione di certificati e chiavi per consentire loro di connettersi. AWS IoT Greengrass

1. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).
2. Scegliere il gruppo target.
3. Nella pagina di configurazione del gruppo, scegli Dispositivi client, quindi scegli Associa.
4. Nella finestra Modale Associa un dispositivo client a questo gruppo, scegli Crea nuovo AWS IoT elemento.

La pagina Crea oggetti si apre in una nuova scheda.

5. Nella pagina Crea elementi, scegli Crea elemento singolo, quindi scegli Avanti.
6. Nella pagina Specificare le proprietà dell'oggetto, registra questo dispositivo client come **HelloWorld_Publisher**, quindi scegli Avanti.
7. Nella pagina Configura il certificato del dispositivo, scegli Avanti.
8. Nella pagina Allega le politiche al certificato, esegui una delle seguenti operazioni:
 - Seleziona una politica esistente che conceda le autorizzazioni richieste dai dispositivi client, quindi scegli Crea oggetto.

Si apre una finestra modale in cui è possibile scaricare i certificati e le chiavi utilizzati dal dispositivo per connettersi al core Cloud AWS .

- Crea e allega una nuova policy che conceda le autorizzazioni per i dispositivi client. Esegui questa operazione:
 - a. Scegli Create Policy (Crea policy).

La pagina Create policy (Crea policy) viene aperta in una nuova scheda.

- b. Nella pagina Create policy (Crea policy), eseguire le operazioni seguenti:
 - i. Per Nome della politica, inserisci un nome che descriva la politica, ad esempio.
GreengrassV1ClientDevicePolicy
 - ii. Nella scheda Dichiarazioni politiche, in Documento di politica, scegli JSON.

- iii. Inserisci il seguente documento di policy. Questa politica consente al dispositivo client di scoprire i core Greengrass e comunicare su tutti gli argomenti MQTT. Per informazioni su come limitare l'accesso a questa politica, vedere. [Autenticazione e autorizzazione del dispositivo per AWS IoT Greengrass](#)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Subscribe",
        "iot:Connect",
        "iot:Receive"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "greengrass:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

- iv. Scegliere Crea per creare la policy.
- c. Torna alla scheda del browser con la pagina Allega politiche al certificato aperta. Esegui questa operazione:
 - i. Nell'elenco Politiche, seleziona la politica che hai creato, ad esempio GreengrassV1ClientDevicePolicy.

Se non vedi la politica, scegli il pulsante di aggiornamento.
 - ii. Scegli Create thing (Crea oggetto).

Si apre una finestra modale in cui puoi scaricare i certificati e le chiavi che il dispositivo utilizza per connettersi al Cloud AWS core.

9. Nella modalità Scarica certificati e chiavi, scarica i certificati del dispositivo.

 Important

Prima di scegliere Fine, scarica le risorse di sicurezza.

Esegui questa operazione:

- a. Per Certificato del dispositivo, scegli Scarica per scaricare il certificato del dispositivo.
- b. Per il file della chiave pubblica, scegli Scarica per scaricare la chiave pubblica per il certificato.
- c. Per File di chiave privata, scegli Scarica per scaricare il file di chiave privata per il certificato.
- d. Consulta [l'autenticazione del server](#) nella Guida per gli AWS IoT sviluppatori e scegli il certificato CA principale appropriato. Ti consigliamo di utilizzare gli endpoint Amazon Trust Services (ATS) e i certificati CA root ATS. In Certificati CA root, scegli Scarica per un certificato CA root.
- e. Seleziona Fatto.

Prendi nota dell'ID del certificato che è comune nei nomi dei file per il certificato e le chiavi del dispositivo. perché sarà necessaria in seguito.

10. Torna alla scheda del browser con la modalità Associa un dispositivo client a questo gruppo aperta. Esegui questa operazione:

- a. Come nome della AWS IoT cosa, scegli HelloWorld_Publisher cosa che hai creato.

Se non vedi l'oggetto, scegli il pulsante di aggiornamento.

- b. Selezionare Associate (Associa).

11. Ripeti i passaggi da 3 a 10 per aggiungere un secondo dispositivo client al gruppo.

Assegna un nome a questo dispositivo client **HelloWorld_Subscriber**. Scarica i certificati e le chiavi per questo dispositivo client sul tuo computer. Ancora una volta, prendi nota dell'ID del certificato che è comune nei nomi di file per il dispositivo HelloWorld_Subscriber.

Ora dovresti avere due dispositivi client nel tuo gruppo Greengrass:

- HelloWorld_Editore
- HelloWorld_Sottoscrittore

12. Crea una cartella sul tuo computer per le credenziali di sicurezza di questi dispositivi client. Copia i certificati e le chiavi in questa cartella.

Configurazione delle sottoscrizioni

In questo passaggio, si abilita il dispositivo client HelloWorld_Publisher per inviare messaggi MQTT al dispositivo client HelloWorld_Subscriber.

1. Nella pagina di configurazione del gruppo, scegli la scheda Abbonamenti, quindi scegli Aggiungi.
2. Nella pagina Crea un abbonamento, procedi come segue per configurare l'abbonamento:
 - a. Per Tipo di origine, scegli Dispositivo client, quindi scegli HelloWorld_Publisher.
 - b. In Tipo di destinazione, scegli Dispositivo client, quindi scegli HelloWorld_Subscriber.
 - c. In Topic filter (Filtro argomento), immettere **hello/world/pubsub**.

Note

Adesso puoi eliminare le sottoscrizioni indicate nei moduli precedenti. Nella pagina Sottoscrizioni del gruppo, seleziona gli abbonamenti da eliminare, quindi scegli Elimina.

- d. Scegli Crea sottoscrizione.
3. Assicurati che il rilevamento automatico sia abilitato in modo che il core di Greengrass possa pubblicare un elenco dei suoi indirizzi IP. I dispositivi client utilizzano queste informazioni per scoprire il core. Esegui questa operazione:
 - a. Nella pagina di configurazione del gruppo, scegli la scheda Funzioni Lambda.
 - b. In Funzioni System Lambda, scegli Rilevatore IP, quindi scegli Modifica.
 - c. In Modifica le impostazioni del rilevatore IP, scegli Rileva automaticamente e sostituisci gli endpoint del broker MQTT, quindi scegli Salva.
 4. Assicuratevi che il demone Greengrass sia in esecuzione, come descritto in. [Distribuire configurazioni cloud su un dispositivo core](#)

5. Nella pagina di configurazione del gruppo, scegliete Deploy.

Lo stato della distribuzione è visualizzato sotto il nome del gruppo nell'intestazione di pagina. Per visualizzare i dettagli della distribuzione, scegli la scheda Distribuzioni.

Installa il SDK per dispositivi AWS IoT for Python

I dispositivi client possono utilizzare SDK per dispositivi AWS IoT for Python per comunicare con AWS IoT i dispositivi AWS IoT Greengrass principali (utilizzando il linguaggio di programmazione Python). Per ulteriori informazioni, compresi i requisiti, consultate il SDK per dispositivi AWS IoT [Readme](#) for Python on GitHub

In questo passaggio, installerai l'SDK e otterrai la funzione di `basicDiscovery.py` esempio utilizzata dai dispositivi client simulati sul tuo computer.

1. Per installare l'SDK sul computer con tutti i componenti richiesti, scegliere il sistema operativo:

Windows

1. Apri un [prompt di comandi elevati](#) ed esegui il seguente comando:

```
python --version
```

Se non vengono restituite le informazioni relative alla versione o se il numero di versione è inferiore a 2.7 per Python 2 o meno di 3.3 per Python 3, seguire le istruzioni in [Download di Python](#) per installare Python 2.7+ o Python 3.3+. Per ulteriori informazioni sulla manutenzione Windows, consulta [Utilizzo di Python su Windows](#).

2. Scaricate il zip file [SDK per dispositivi AWS IoT for Python](#) ed estraetelo in una posizione appropriata sul vostro computer.

Prendi nota del percorso file alla cartella `aws-iot-device-sdk-python-master` estratta che contiene il file `setup.py`. Nel passaggio successivo, questo percorso del file è indicato da *`path-to-SDK-folder`*.

3. Dal prompt di comandi elevati, esegui il comando seguente:

```
cd path-to-SDK-folder  
python setup.py install
```

macOS

1. Apri una finestra del terminale ed esegui il comando seguente:

```
python --version
```

Se non vengono restituite le informazioni relative alla versione o se il numero di versione è inferiore a 2.7 per Python 2 o meno di 3.3 per Python 3, seguire le istruzioni in [Download di Python](#) per installare Python 2.7+ o Python 3.3+. Per ulteriori informazioni sulla manutenzione Windows, consulta [Utilizzo di Python su Macintosh](#).

2. Nella finestra del terminale, esegui i seguenti comandi per determinare la versione di OpenSSL:

```
python
>>>import ssl
>>>print ssl.OPENSSL_VERSION
```

Annota il valore della versione di OpenSSL.

Note

Se Python 3 è in esecuzione, utilizza `print(ssl.OPENSSL_VERSION)`.

Per chiudere lo shell di Python, esegui il seguente comando:

```
>>>exit()
```

Se la versione di OpenSSL è 1.0.1 o successiva, vai alla [fase c](#). In caso contrario, procedi come descritto:

- Nella finestra del terminale, esegui il comando seguente per determinare se il computer sta utilizzando Simple Python Version Management:

```
which pyenv
```

Se viene restituito un percorso di file, scegli la pyenv scheda Utilizzo. Se non viene restituito nulla, scegli la pyenv scheda Non in uso.

Using pyenv

1. Consulta [Versioni di Python per Mac OS X](#) (o simile) per determinare la versione stabile più recente di Python. Nell'esempio seguente, questo valore è indicato da *latest-Python-version*.
2. Dalla finestra di terminale, esegui i comandi seguenti:

```
pyenv install latest-Python-version  
pyenv global latest-Python-version
```

Ad esempio, se la versione più recente per Python 2 è 2.7.14, i comandi sono:

```
pyenv install 2.7.14  
pyenv global 2.7.14
```

3. Chiudi e riapri la finestra del terminale ed esegui i comandi seguenti:

```
python  
>>>import ssl  
>>>print ssl.OPENSSL_VERSION
```

La versione di OpenSSL deve essere almeno 1.0.1. Se la versione è precedente a 1.0.1, l'aggiornamento non è riuscito. Controlla la versione di Python usata nei comandi pyenv install e pyenv global e riprova.

4. Esegui il comando seguente per uscire dallo shell di Python:

```
exit()
```

Not using pyenv

1. Da una finestra del terminale, esegui il comando seguente per determinare se [brew](#) è installato:

```
which brew
```

Se non viene restituito un percorso di file, installa brew come segue:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

 Note

Segui le istruzioni di installazione. Il download degli strumenti a riga di comando Xcode può richiedere del tempo.

2. Esegui i comandi seguenti:

```
brew update  
brew install openssl  
brew install python@2
```

SDK per dispositivi AWS IoT for Python richiede OpenSSL versione 1.0.1 (o successiva) compilata con l'eseguibile Python. Il comando `brew install python` installa un eseguibile `python2` che soddisfa questa esigenza. L'eseguibile `python2` è installato nella directory `/usr/local/bin`, che dovrebbe essere parte della variabile di ambiente `PATH`. Per averne la conferma, esegui il comando seguente:

```
python2 --version
```

Se le informazioni di versione di `python2` vengono fornite, vai alla fase successiva. In caso contrario, aggiungi definitivamente il percorso `/usr/local/bin` alla variabile di ambiente `PATH` aggiungendo la seguente riga al profilo shell:

```
export PATH="/usr/local/bin:$PATH"
```

Ad esempio, se stai utilizzando `.bash_profile` o non disponi di un profilo shell, esegui il seguente comando da una finestra del terminale:

```
echo 'export PATH="/usr/local/bin:$PATH"' >> ~/.bash_profile
```

Quindi, [rintraccia l'origine](#) del profilo shell e conferma che `python2 --version` fornisca le informazioni sulla versione. Ad esempio, se utilizzi `.bash_profile`, esegui i comandi seguenti:

```
source ~/.bash_profile
python2 --version
```

Le informazioni sulla versione di `python2` dovrebbero essere restituite.

3. Aggiungi la riga seguente al profilo shell:

```
alias python="python2"
```

Ad esempio, se stai utilizzando `.bash_profile` o non disponi di un profilo shell, esegui il seguente comando:

```
echo 'alias python="python2"' >> ~/.bash_profile
```

4. Quindi, [rintraccia l'origine](#) del profilo shell. Ad esempio, se utilizzi `.bash_profile`, esegui il comando seguente:

```
source ~/.bash_profile
```

Richiamando il comando `python` verrà avviato l'eseguibile Python che contiene la versione di OpenSSL richiesta (`python2`).

5. Esegui i comandi seguenti:

```
python
import ssl
print ssl.OPENSSL_VERSION
```

La versione di OpenSSL deve essere almeno 1.0.1. o successiva.

6. Per uscire dallo shell di Python, esegui il comando seguente:

```
exit()
```

3. Esegui i seguenti comandi per installare il SDK per dispositivi AWS IoT for Python:

```
cd ~  
git clone https://github.com/aws/aws-iot-device-sdk-python.git  
cd aws-iot-device-sdk-python  
sudo python setup.py install
```

UNIX-like system

1. Dalla finestra di un terminale, eseguire il comando riportato qui sotto:

```
python --version
```

Se non vengono restituite le informazioni relative alla versione o se il numero di versione è inferiore a 2.7 per Python 2 o meno di 3.3 per Python 3, seguire le istruzioni in [Download di Python](#) per installare Python 2.7+ o Python 3.3+. Per ulteriori informazioni, consulta [Utilizzo di Python su piattaforme Unix](#).

2. Nel terminale, esegui i seguenti comandi per determinare la versione di OpenSSL:

```
python  
>>>import ssl  
>>>print ssl.OPENSSL_VERSION
```

Annota il valore della versione di OpenSSL.

Note

Se Python 3 è in esecuzione, utilizza `print(ssl.OPENSSL_VERSION)`.

Per chiudere lo shell di Python, esegui il seguente comando:

```
exit()
```

Se la versione di OpenSSL è 1.0.1 o successiva, vai alla fase successiva. In caso contrario, esegui il comando/i per aggiornare OpenSSL per la distribuzione (ad esempio, `sudo yum update openssl`, `sudo apt-get update` e così via).

Verifica che la versione di OpenSSL sia 1.0.1 o successiva eseguendo i comandi seguenti:

```
python
>>>import ssl
>>>print ssl.OPENSSL_VERSION
>>>exit()
```

3. Esegui i seguenti comandi per installare il SDK per dispositivi AWS IoT for Python:

```
cd ~
git clone https://github.com/aws/aws-iot-device-sdk-python.git
cd aws-iot-device-sdk-python
sudo python setup.py install
```

2. Dopo aver installato SDK per dispositivi AWS IoT for Python, vai alla `samples` cartella e apri la `greengrass` cartella.

Per questo tutorial si copia la funzione di esempio `basicDiscovery.py`, che utilizza i certificati e le chiavi che sono stati scaricati in [the section called “Creare dispositivi client in un AWS IoT Greengrass gruppo”](#).

3. Copia `basicDiscovery.py` nella cartella che contiene i certificati e le chiavi dei dispositivi `HelloWorld_Publisher` e `HelloWorld_Subscriber`.

Test delle comunicazioni

1. Assicurati che il computer e il dispositivo AWS IoT Greengrass principale siano connessi a Internet utilizzando la stessa rete.
 - a. Sul dispositivo AWS IoT Greengrass principale, esegui il comando seguente per trovare il relativo indirizzo IP.

```
hostname -I
```

- b. Sul computer, esegui il comando seguente utilizzando l'indirizzo IP del core. È possibile utilizzare `Ctrl + C` per arrestare il comando `ping`.

```
ping IP-address
```

Un output simile al seguente indica una comunicazione riuscita tra il computer e il dispositivo AWS IoT Greengrass principale (perdita di pacchetti pari allo 0%):

```
$ping 176.32.103.205
PING 176.32.103.205 (176.32.103.205) 56(84) bytes of data.
64 bytes from 176.32.103.205: icmp_seq=1 ttl=230 time=77.2 ms
64 bytes from 176.32.103.205: icmp_seq=2 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=3 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=4 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=5 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=6 ttl=230 time=77.1 ms
^C
--- 176.32.103.205 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5549ms
rtt min/avg/max/mdev = 77.107/77.172/77.256/0.361 ms
```

Note

Se non riesci a eseguire il ping di un' EC2 istanza in esecuzione AWS IoT Greengrass, assicurati che le regole del gruppo di sicurezza in entrata relative all'istanza consentano il traffico ICMP per i messaggi di richiesta [Echo](#). Per ulteriori informazioni, consulta [Aggiungere regole a un gruppo di sicurezza](#) nella Amazon EC2 User Guide.

Sui computer host Windows, nell'app Windows Firewall with Advanced Security, potrebbe anche essere necessario abilitare una regola in entrata che consenta le richieste echo in entrata (ad esempio, condivisione di file e stampanti (Echo Request - ICMPv4 -In)) o crearne una.

2. Ottieni il tuo endpoint. AWS IoT

- a. Dal pannello di navigazione [AWS IoT della console](#), scegli Impostazioni.
- b. In Device data endpoint, prendi nota del valore di Endpoint. Si utilizza questo valore per sostituire il `AWS_IOT_ENDPOINT` segnato nei comandi nei passaggi seguenti.

Note

Assicurati che gli [endpoint corrispondano al tipo di certificato](#).

3. Sul computer (non sul dispositivo AWS IoT Greengrass principale), apri due finestre della [riga di comando](#) (terminale o prompt dei comandi). Una finestra rappresenta il dispositivo client HelloWorld_Publisher e l'altra rappresenta il dispositivo client _Subscriber. HelloWorld

Al momento dell'esecuzione, basicDiscovery.py tenta di raccogliere informazioni sulla posizione del AWS IoT Greengrass core nei relativi punti terminali. Queste informazioni vengono memorizzate dopo che il dispositivo client è stato scoperto e connesso correttamente al core. In questo modo le operazioni e la messaggistica future saranno eseguite in locale (senza bisogno di una connessione a Internet).

Note

Il client IDs utilizzato per le connessioni MQTT deve corrispondere al nome dell'oggetto del dispositivo client. Lo basicDiscovery.py script imposta l'ID client per le connessioni MQTT sul nome dell'oggetto specificato quando si esegue lo script. Eseguite il comando seguente dalla cartella che contiene il basicDiscovery.py file per informazioni dettagliate sull'utilizzo dello script:

```
python basicDiscovery.py --help
```

4. Dalla finestra del dispositivo client HelloWorld_Publisher, esegui i comandi seguenti.
 - Sostituisci *path-to-certs-folder* con il percorso della cartella che contiene i certificati, le chiavi e basicDiscovery.py.
 - Sostituisci *AWS_IOT_ENDPOINT* con il tuo endpoint.
 - Sostituisci le due *publisherCertId* istanze con l'ID del certificato nel nome del file per il tuo dispositivo client HelloWorld_Publisher.

```
cd path-to-certs-folder
python basicDiscovery.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem
--cert publisherCertId-certificate.pem.crt --key publisherCertId-private.pem.key
--thingName HelloWorld_Publisher --topic 'hello/world/pubsub' --mode publish --
message 'Hello, World! Sent from HelloWorld_Publisher'
```

L'output risultante dovrebbe essere analogo al seguente, con voci come Published topic 'hello/world/pubsub': {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 1}.

Note

Se lo script restituisce un messaggio `error: unrecognized arguments`, cambia le virgolette singole in virgolette doppie per i parametri `--topic` e `--message` ed esegui nuovamente il comando.

Per risolvere un problema di connessione, puoi provare a usare il [rilevamento IP manuale](#).

```
Published topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 0}
2017-11-13 21:12:26,296 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [puback] event
2017-11-13 21:12:26,297 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [puback] event
2017-11-13 21:12:27,301 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
Published topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 1}
2017-11-13 21:12:27,302 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [puback] event
2017-11-13 21:12:27,303 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [puback] event
2017-11-13 21:12:28,305 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
Published topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 2}
2017-11-13 21:12:28,306 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [puback] event
2017-11-13 21:12:28,307 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [puback] event
2017-11-13 21:12:29,310 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
Published topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 3}
```

5. Dalla finestra del dispositivo client HelloWorld_Subscriber, esegui i seguenti comandi.

- Sostituisci *path-to-certs-folder* con il percorso della cartella che contiene i certificati, le chiavi e. `basicDiscovery.py`
- Sostituisci *AWS_IOT_ENDPOINT* con il tuo endpoint.
- Sostituisci le due *subscriberCertId* istanze con l'ID del certificato nel nome del file per il tuo dispositivo client HelloWorld_Subscriber.

```
cd path-to-certs-folder
python basicDiscovery.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem --
cert subscriberCertId-certificate.pem.crt --key subscriberCertId-private.pem.key --
thingName HelloWorld_Subscriber --topic 'hello/world/pubsub' --mode subscribe
```

L'output risultante dovrebbe essere il seguente, con voci come `Received message on topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 1}`.

```

Received message on topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 0}
2017-11-13 21:12:27,435 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [message] event
2017-11-13 21:12:27,435 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [message] event
2017-11-13 21:12:27,436 - AWSIoTPythonSDK.core.protocol.internal.clients - DEBUG - Invoking custom event callback...
Received message on topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 1}
2017-11-13 21:12:28,320 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [message] event
2017-11-13 21:12:28,324 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [message] event
2017-11-13 21:12:28,324 - AWSIoTPythonSDK.core.protocol.internal.clients - DEBUG - Invoking custom event callback...
Received message on topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 2}
2017-11-13 21:12:29,547 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [message] event
2017-11-13 21:12:29,552 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [message] event
2017-11-13 21:12:29,552 - AWSIoTPythonSDK.core.protocol.internal.clients - DEBUG - Invoking custom event callback...

```

Chiudi il HelloWorld_Publisher finestra per impedire l'accumulo di messaggi nel HelloWorld_Subscriber finestra.

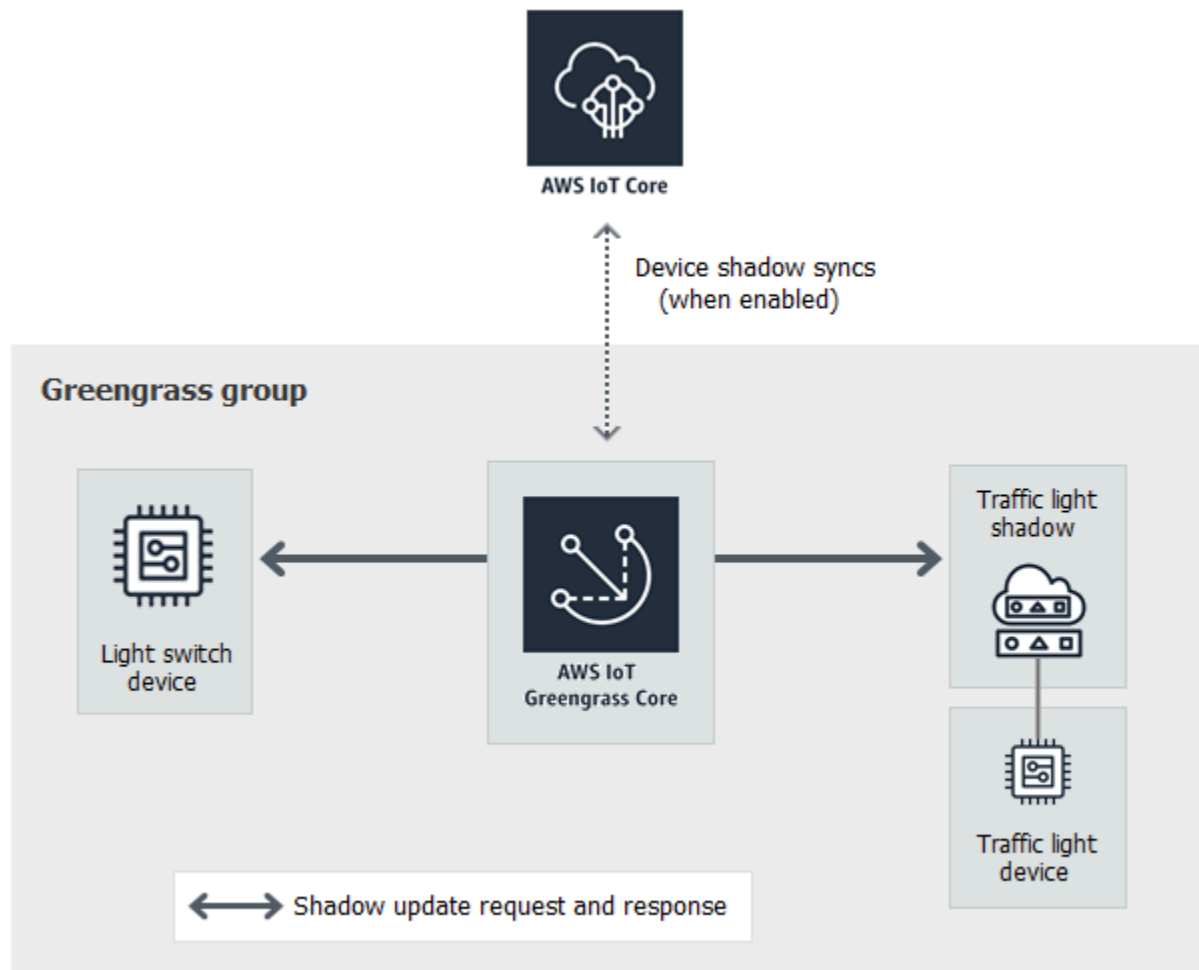
Il test su una rete aziendale potrebbe interferire con la connessione al core. Come soluzione alternativa, puoi immettere manualmente l'endpoint. Ciò garantisce che lo `basicDiscovery.py` script si connetta all'indirizzo IP corretto del dispositivo AWS IoT Greengrass principale.

Per immettere manualmente l'endpoint

1. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).
2. In Gruppi Greengrass, scegli il tuo gruppo.
3. Configura il core per gestire manualmente gli endpoint del broker MQTT. Esegui questa operazione:
 - a. Nella pagina di configurazione del gruppo, scegli la scheda Funzioni Lambda.
 - b. In Funzioni System Lambda, scegli Rilevatore IP, quindi scegli Modifica.
 - c. In Modifica le impostazioni del rilevatore IP, scegli Gestisci manualmente gli endpoint del broker MQTT, quindi scegli Salva.
4. Inserisci l'endpoint del broker MQTT per il core. Esegui questa operazione:
 - a. In Panoramica, scegli il core Greengrass.
 - b. In Endpoints del broker MQTT, scegli Gestisci endpoints.
 - c. Scegli Aggiungi endpoint e assicurati di avere un solo valore di endpoint. Questo valore deve essere l'endpoint dell'indirizzo IP per la porta 8883 del dispositivo AWS IoT Greengrass principale (ad esempio,). 192.168.1.4
 - d. Scegli Aggiorna.

Modulo 5: interazione con dispositivi ombra

Questo modulo avanzato mostra come i dispositivi client possono interagire con [le ombre dei AWS IoT dispositivi](#) in un AWS IoT Greengrass gruppo. Una shadow è un documento JSON utilizzato per archiviare informazioni di stato correnti o desiderate per un oggetto. In questo modulo, scoprite come un dispositivo client (GG_Switch) può modificare lo stato di un altro dispositivo client (GG_TrafficLight) e come questi stati possono essere sincronizzati con il AWS IoT Greengrass cloud:



Prima di iniziare, eseguire lo script [Greengrass Device Setup](#) o assicurarsi di aver completato il [modulo 1](#) e il [modulo 2](#). Dovresti anche capire come connettere i dispositivi client a un AWS IoT Greengrass core ([Modulo 4](#)). Non sono necessari altri componenti o dispositivi.

Il completamento di questo modulo richiede circa 30 minuti.

Argomenti

- [Configurazione di dispositivi e sottoscrizioni](#)
- [Scarica i file richiesti](#)
- [Test delle comunicazioni \(sincronizzazione dispositivi disattivata\)](#)
- [Test delle comunicazioni \(sincronizzazione dispositivi attivata\)](#)

Configurazione di dispositivi e sottoscrizioni

Le ombre possono essere sincronizzate AWS IoT quando il AWS IoT Greengrass core è connesso a Internet. In questo modulo utilizzerai prima di tutto le copie shadow locali senza sincronizzarle con il cloud. In seguito, abiliterai la sincronizzazione cloud.

Ogni dispositivo client ha la propria ombra. Per ulteriori informazioni, consulta [Device shadow service AWS IoT](#) nella AWS IoT Device Shadow Guide.

1. Nella pagina di configurazione del gruppo, scegli la scheda Dispositivi client.
2. Dalla scheda Dispositivi client, aggiungi due nuovi dispositivi client al tuo AWS IoT Greengrass gruppo. Per una procedura dettagliata di questo processo, consulta [the section called “Creare dispositivi client in un AWS IoT Greengrass gruppo”](#).
 - Assegna un nome ai dispositivi client **GG_Switch** e **GG_TrafficLight**.
 - Genera e scarica le risorse di sicurezza per entrambi i dispositivi client.
 - Prendi nota dell'ID del certificato nei nomi dei file delle risorse di sicurezza per i dispositivi client. Questi valori verranno usati in seguito.
3. Crea una cartella sul tuo computer per le credenziali di sicurezza di questi dispositivi client. Copia i certificati e le chiavi in questa cartella.
4. Assicurati che i dispositivi client siano impostati per utilizzare le ombre locali e non si sincronizzino con. Cloud AWS In caso contrario, seleziona il dispositivo client, scegli Sincronizza shadow, quindi scegli Disabilita la sincronizzazione delle ombre con il cloud.
5. Aggiungi le sottoscrizioni nella tabella seguente al tuo gruppo. Ad esempio, per creare la prima sottoscrizione:
 - a. Nella pagina di configurazione del gruppo, scegli la scheda Abbonamenti, quindi scegli Aggiungi.
 - b. Per Tipo di sorgente, scegli Dispositivo client, quindi scegli GG_Switch.

- c. Per Tipo di destinazione, scegliete Servizio, quindi scegliete Local Shadow Service.
- d. In Filtro di argomenti, immetti **\$aws/things/GG_TrafficLight/shadow/update**.
- e. Scegli Crea sottoscrizione.

Gli argomenti devono essere immessi esattamente come mostrato nella tabella. Anche se è possibile utilizzare i caratteri jolly per consolidare alcune delle sottoscrizioni, è sconsigliato ricorrere a questa pratica. Per ulteriori informazioni, consultate gli [argomenti relativi a Shadow MQTT](#) nella AWS IoT Developer Guide.

Origine	Target	Argomento	Note
GG_Switch	Servizio shadow locale	\$aws/things/GG_TrafficLight/shadow/update	GG_Switch invia una richiesta di aggiornamento per aggiornare l'argomento.
Servizio shadow locale	GG_Switch	\$aws/things/GG_TrafficLight/shadow/update/accepted	GG_Switch deve sapere se la richiesta di aggiornamento è stata accettata.
Servizio shadow locale	GG_Switch	\$aws/things/GG_TrafficLight/shadow/update/rejected	GG_Switch deve sapere se la richiesta di aggiornamento è stata rifiutata.
GG_TrafficLight	Servizio shadow locale	\$aws/things/GG_TrafficLight/shadow/update	Il GG_TrafficLight invia un aggiornamento del suo stato all'argomento dell'aggiornamento.

Origine	Target	Argomento	Note
Servizio shadow locale	GG_TrafficLight	\$aws/things/GG_TrafficLight/shadow/update/delta	Il Local Shadow Service invia un aggiornamento ricevuto a GG_TrafficLight tramite l'argomento delta.
Servizio shadow locale	GG_TrafficLight	\$aws/things/GG_TrafficLight/shadow/update/accepted	Il GG_TrafficLight deve sapere se il suo aggiornamento di stato è stato accettato.
Servizio shadow locale	GG_TrafficLight	\$aws/things/GG_TrafficLight/shadow/update/rejected	Il GG_TrafficLight deve sapere se il suo aggiornamento di stato è stato rifiutato.

I nuovi abbonamenti vengono visualizzati nella scheda Abbonamenti.

Note

Per ulteriori informazioni sul carattere \$, consulta [Argomenti riservati](#).

6. Assicurati che il rilevamento automatico sia abilitato in modo che il core di Greengrass possa pubblicare un elenco dei suoi indirizzi IP. I dispositivi client utilizzano queste informazioni per scoprire il core. Esegui questa operazione:
 - a. Nella pagina di configurazione del gruppo, scegli la scheda Funzioni Lambda.
 - b. In Funzioni System Lambda, scegli Rilevatore IP, quindi scegli Modifica.
 - c. In Modifica le impostazioni del rilevatore IP, scegli Rileva automaticamente e sostituisci gli endpoint del broker MQTT, quindi scegli Salva.
7. Assicuratevi che il demone Greengrass sia in esecuzione, come descritto in [Distribuire configurazioni cloud su un dispositivo core](#)

8. Nella pagina di configurazione del gruppo, scegliete Deploy.

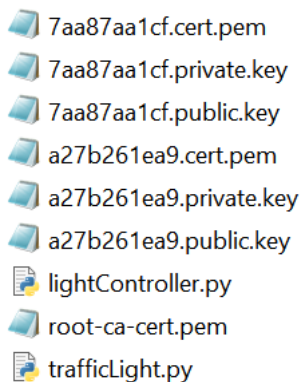
Scarica i file richiesti

1. Se non l'hai già fatto, installa il SDK per dispositivi AWS IoT for Python. Per istruzioni, consulta il passaggio 1 in [the section called “Installa il SDK per dispositivi AWS IoT for Python”](#).

Questo SDK viene utilizzato dai dispositivi client per comunicare con AWS IoT e con i dispositivi AWS IoT Greengrass principali.

2. Dalla cartella degli [TrafficLight](#) esempi in poi GitHub, scarica i `trafficLight.py` file `lightController.py` and sul tuo computer. Salvateli nella cartella che contiene i certificati e le chiavi dei dispositivi TrafficLight client `GG_Switch` e `GG_`.

Lo `lightController.py` script corrisponde al dispositivo client `GG_Switch` e lo script `trafficLight.py` corrisponde al dispositivo client `GG_`.



Note

I file Python di esempio sono archiviati nel repository AWS IoT Greengrass Core SDK for Python per comodità, ma non utilizzano Core SDK. AWS IoT Greengrass

Test delle comunicazioni (sincronizzazione dispositivi disattivata)

1. Assicurati che il computer e il dispositivo AWS IoT Greengrass principale siano connessi a Internet utilizzando la stessa rete.

- a. Sul dispositivo AWS IoT Greengrass principale, esegui il comando seguente per trovare il relativo indirizzo IP.

```
hostname -I
```

- b. Sul computer, esegui il comando seguente utilizzando l'indirizzo IP del core. È possibile utilizzare Ctrl + C per arrestare il comando ping.

```
ping IP-address
```

Un output simile al seguente indica una comunicazione riuscita tra il computer e il dispositivo AWS IoT Greengrass principale (perdita di pacchetti pari allo 0%):

```
$ping 176.32.103.205
PING 176.32.103.205 (176.32.103.205) 56(84) bytes of data.
64 bytes from 176.32.103.205: icmp_seq=1 ttl=230 time=77.2 ms
64 bytes from 176.32.103.205: icmp_seq=2 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=3 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=4 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=5 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=6 ttl=230 time=77.1 ms
^C
--- 176.32.103.205 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5549ms
rtt min/avg/max/mdev = 77.107/77.172/77.256/0.361 ms
```

 Note

Se non riesci a eseguire il ping di un' EC2 istanza in esecuzione AWS IoT Greengrass, assicurati che le regole del gruppo di sicurezza in entrata relative all'istanza consentano il traffico ICMP per i messaggi di richiesta [Echo](#). Per ulteriori informazioni, consulta [Aggiungere regole a un gruppo di sicurezza](#) nella Amazon EC2 User Guide.

Sui computer host Windows, nell'app Windows Firewall with Advanced Security, potrebbe anche essere necessario abilitare una regola in entrata che consenta le richieste echo in entrata (ad esempio, condivisione di file e stampanti (Echo Request - ICMPv4 -In)) o crearne una.

2. Ottieni il tuo endpoint. AWS IoT

- a. Dal pannello di navigazione [AWS IoT della console](#), scegli Impostazioni.
- b. In Device data endpoint, prendi nota del valore di Endpoint. Si utilizza questo valore per sostituire il *AWS_IOT_ENDPOINT* segnaposto nei comandi nei passaggi seguenti.

 Note

Assicurati che gli [endpoint corrispondano al tipo di certificato](#).

3. Sul computer (non sul dispositivo AWS IoT Greengrass principale), apri due finestre della [riga di comando](#) (terminale o prompt dei comandi). Una finestra rappresenta il dispositivo client GG_Switch e l'altra rappresenta il dispositivo client GG_TrafficLight
 - a. Dalla finestra del dispositivo client GG_Switch, esegui i seguenti comandi.
 - Sostituisci *path-to-certs-folder* con il percorso della cartella che contiene i certificati, le chiavi e i file Python.
 - *AWS_IOT_ENDPOINT* Sostituiscilo con il tuo endpoint.
 - Sostituisci le due *switchCertId* istanze con l'ID del certificato nel nome del file per il tuo dispositivo client GG_Switch.

```
cd path-to-certs-folder  
python lightController.py --endpoint AWS_IOT_ENDPOINT --rootCA  
AmazonRootCA1.pem --cert switchCertId-certificate.pem.crt --key switchCertId-  
private.pem.key --thingName GG_TrafficLight --clientId GG_Switch
```

- b. Dalla finestra del dispositivo TrafficLight client GG_, esegui i seguenti comandi.
 - Sostituisci *path-to-certs-folder* con il percorso della cartella che contiene i certificati, le chiavi e i file Python.
 - *AWS_IOT_ENDPOINT* Sostituiscilo con il tuo endpoint.
 - Sostituisci le due *lightCertId* istanze con l'ID del certificato nel nome del file per il tuo dispositivo client GG_TrafficLight .

```
cd path-to-certs-folder
```

```
python trafficLight.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem
--cert lightCertId-certificate.pem.crt --key lightCertId-private.pem.key --
thingName GG_TrafficLight --clientId GG_TrafficLight
```

Ogni 20 secondi, lo switch aggiorna lo stato shadow a G, Y e R e la luce mostra il nuovo stato, come mostrato di seguito.

Output di GG_Switch:

```
{"state":{"desired":{"property":"R"}}}
2018-12-20 12:23:01,446 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
~~~~~Shadow Update Accepted~~~~~
Update request with token: 3b22e27c-930d-4c6a-8562-9f86088249f4 accepted!
property: R
~~~~~
```

Uscita GG_: TrafficLight

```
+++++++ Received Shadow Delta ++++++++
{'u'state': {'u'property': 'R'}, 'u'metadata': {'u'property': {'u'timestamp': 1545337381}}, 'u'version': 33, 'u'clientToken':
'u'3b22e27c-930d-4c6a-8562-9f86088249f4'}
property: R
version: 33
+++++++

Light changed to: R
{"state":{"reported":{"property":"R"}}}
2018-12-20 12:23:01,539 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
~~~~~Shadow Update Accepted~~~~~
Update request with token: f552109f-c1c2-4ae6-a841-8443506eefcb accepted!
property: R
~~~~~
```

Quando viene eseguito per la prima volta, ogni script del dispositivo client esegue il servizio di AWS IoT Greengrass rilevamento per connettersi al AWS IoT Greengrass core (tramite Internet). Dopo che un dispositivo client è stato scoperto e connesso correttamente al AWS IoT Greengrass core, le operazioni future possono essere eseguite localmente.

Note

Gli script `trafficLight.py` e `lightController.py` memorizzano informazioni di connessione nella cartella `groupCA`, che viene creata nella stessa cartella degli script. Se ricevi errori di connessione, assicurati che l'indirizzo IP nel `ggc-host` file corrisponda all'endpoint dell'indirizzo IP del core.

4. Nella AWS IoT console, scegli il tuo AWS IoT Greengrass gruppo, scegli la scheda Dispositivi client, quindi scegli GG_TrafficLight per aprire la pagina dei dettagli dell' AWS IoT oggetto del dispositivo client.
5. Scegli la scheda Device Shadows. Dopo che GG_Switch ha cambiato lo stato, non dovrebbero esserci aggiornamenti a questa ombra. Questo perché GG_TrafficLight è impostato su Disabilita la sincronizzazione delle ombre con il cloud.
6. Premete Ctrl + C nella finestra del dispositivo client GG_switch ()lightController.py. Dovresti vedere che la finestra GG_TrafficLight (trafficLight.py) smette di ricevere messaggi di cambio di stato.

Tieni aperte queste finestre per poter eseguire i comandi nella sezione successiva.

Test delle comunicazioni (sincronizzazione dispositivi attivata)

Per questo test, si configura l'ombra del TrafficLight dispositivo GG_ con cui sincronizzarsi. AWS IoT Si eseguono gli stessi comandi del test precedente, ma questa volta lo stato shadow nel cloud viene stato aggiornato quando GG_Switch invia una richiesta di aggiornamento.

1. Nella AWS IoT console, scegli il tuo AWS IoT Greengrass gruppo, quindi scegli la scheda Dispositivi client.
2. Seleziona il TrafficLight dispositivo GG_, scegli Sincronizza shadow, quindi scegli Abilita la sincronizzazione delle ombre con il cloud.

Dovresti ricevere una notifica che lo stato di sincronizzazione dello shadow del dispositivo è stato aggiornato.

3. Nella pagina di configurazione del gruppo, scegli Deploy.
4. [Nelle due finestre a riga di comando, esegui i comandi del test precedente per i dispositivi client GG_Switch e GG_. TrafficLight](#)
5. Ora controlla lo stato dell'ombra nella console. AWS IoT Scegli il tuo AWS IoT Greengrass gruppo, scegli la scheda Dispositivi client, scegli GG_TrafficLight, scegli la scheda Device Shadows, quindi scegli Classic Shadow.

Poiché hai abilitato la sincronizzazione di GG_TrafficLight shadow con AWS IoT, lo stato ombra nel cloud dovrebbe essere aggiornato ogni volta che GG_Switch invia un aggiornamento. Questa funzionalità può essere utilizzata per esporre lo stato di un dispositivo client a. AWS IoT

 Note

Se necessario, è possibile risolvere i problemi visualizzando i log AWS IoT Greengrass principali, in particolare: `runtime.log`

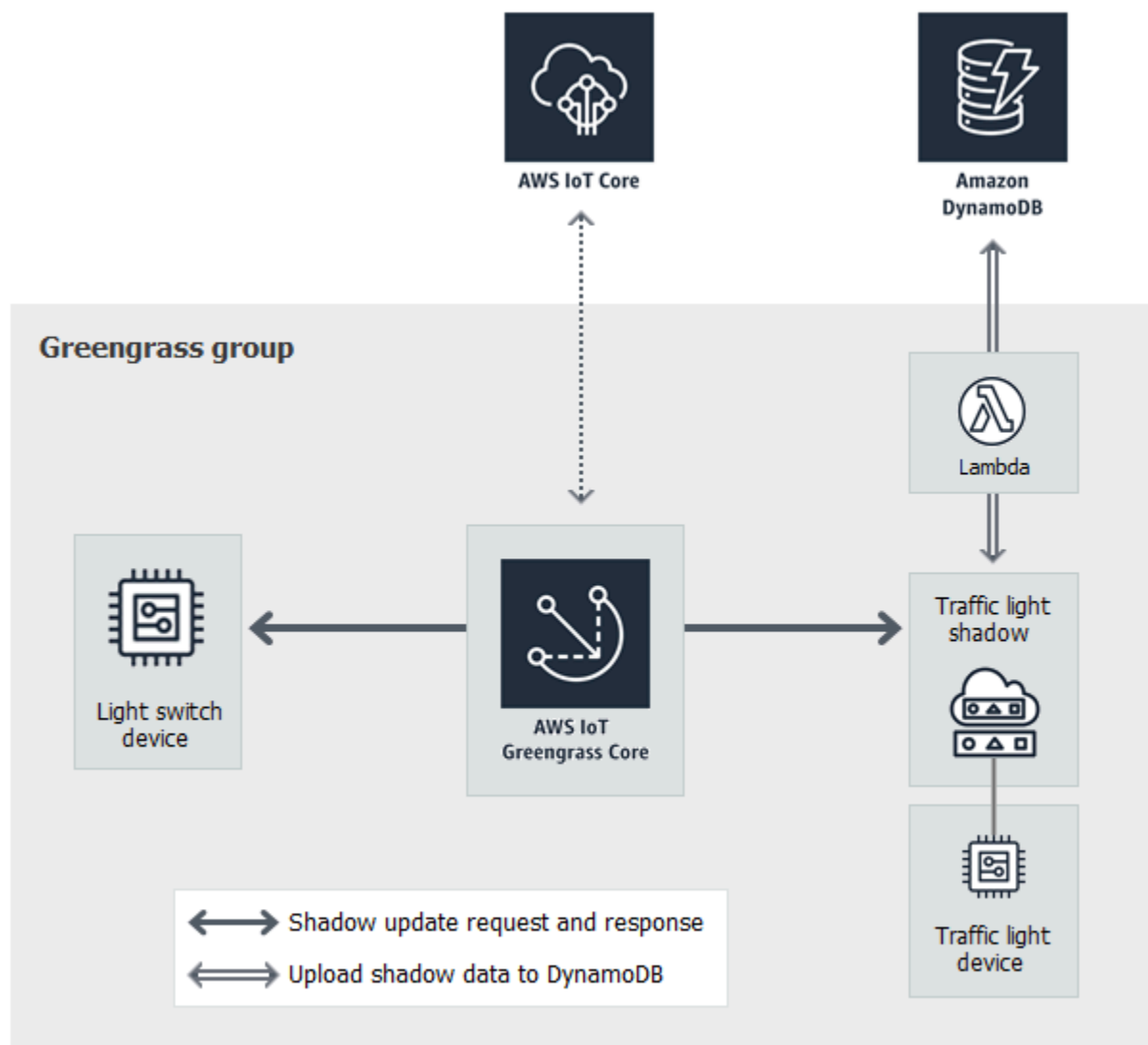
```
cd /greengrass/ggc/var/log  
sudo cat system/runtime.log | more
```

Puoi anche visualizzare `GGShadowSyncManager.log` e `GGShadowService.log`. Per ulteriori informazioni, consulta [Risoluzione dei problemi](#).

Mantieni configurati i dispositivi client e gli abbonamenti. Li utilizzerai nel modulo successiva. Eseguirai anche gli stessi comandi.

Modulo 6: Accesso ad altri AWS servizi

Questo modulo avanzato mostra come AWS IoT Greengrass i core possono interagire con altri AWS servizi nel cloud. Si basa sull'esempio del semaforo tratto dal [Modulo 5](#) e aggiunge una funzione Lambda che elabora gli stati ombra e carica un riepilogo in una tabella Amazon DynamoDB.



Prima di iniziare, eseguire lo script [Greengrass Device Setup](#) o assicurarsi di aver completato il [modulo 1](#) e il [modulo 2](#). È inoltre necessario completare il [modulo 5](#). Non sono necessari altri componenti o dispositivi.

Il completamento di questo modulo richiede circa 30 minuti.

Note

Questo modulo crea e aggiorna una tabella in DynamoDB. Sebbene la maggior parte delle operazioni sia di piccole dimensioni e rientri nel piano gratuito di Amazon Web Services,

l'esecuzione di alcuni passaggi di questo modulo potrebbe comportare addebiti sul tuo account. Per informazioni sui prezzi, consulta la documentazione sui prezzi di [DynamoDB](#).

Argomenti

- [Configurazione del ruolo del gruppo](#)
- [Creare e configurare la funzione Lambda](#)
- [Configurazione delle sottoscrizioni](#)
- [Test delle comunicazioni](#)

Configurazione del ruolo del gruppo

Il ruolo di gruppo è un [ruolo IAM](#) che crei e colleghi al tuo gruppo Greengrass. Questo ruolo contiene le autorizzazioni utilizzate dalle funzioni Lambda distribuite (e altre AWS IoT Greengrass funzionalità) per accedere ai servizi. AWS Per ulteriori informazioni, consulta [the section called “Ruolo del gruppo Greengrass”](#).

Utilizza i seguenti passaggi di alto livello per creare un ruolo di gruppo nella console IAM.

1. Creare una policy che consente o rifiuta operazioni su una o più risorse.
2. Creare un ruolo che utilizza il servizio Greengrass come un'entità attendibile.
3. Collegare la policy al ruolo.

Quindi, nella AWS IoT console, aggiungi il ruolo al gruppo Greengrass.

Note

Un gruppo Greengrass dispone di un ruolo del gruppo. Se desideri aggiungere autorizzazioni, puoi modificare le policy collegate o collegare altre policy.

Per questo tutorial, crei una politica di autorizzazioni che consenta di descrivere, creare e aggiornare azioni su una tabella Amazon DynamoDB. La policy viene quindi collegata a un nuovo ruolo e questo viene associato al gruppo Greengrass.

Innanzitutto, crea una politica gestita dal cliente che conceda le autorizzazioni richieste dalla funzione Lambda in questo modulo.

1. Nella console IAM, nel pannello di navigazione, scegli Policies, quindi scegli Crea policy.
2. Nella scheda JSON, sostituire il contenuto del segnaposto con la seguente policy. La funzione Lambda di questo modulo utilizza queste autorizzazioni per creare e aggiornare una tabella DynamoDB denominata. CarStats

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PermissionsForModule6",
      "Effect": "Allow",
      "Action": [
        "dynamodb:DescribeTable",
        "dynamodb:CreateTable",
        "dynamodb:PutItem"
      ],
      "Resource": "arn:aws:dynamodb:*:*:table/CarStats"
    }
  ]
}
```

3. Scegli Successivo: Tag, quindi Successivo: Rivedi. In questo tutorial non vengono utilizzati tag.
4. Per Name (Nome), immettere **greengrass_CarStats_Table**, quindi scegliere Create policy (Crea criterio).

Quindi, creare un ruolo che utilizza la nuova policy.

5. Nel riquadro di navigazione, scegli Ruoli e quindi Crea ruolo.
6. In Tipo di entità attendibile, scegli Servizio AWS .
7. In Caso d'uso, Casi d'uso per altri AWS servizi scegli Greengrass, seleziona Greengrass e quindi scegli Avanti.
8. In Politiche di autorizzazione, seleziona la nuova **greengrass_CarStats_Table** politica, quindi scegli Avanti.
9. Per Nome ruolo, inserisci **Greengrass_Group_Role**.

10. Per Descrizione, inserisci **Greengrass group role for connectors and user-defined Lambda functions**.
11. Scegliere Crea ruolo.

Ora, aggiungi il ruolo al gruppo Greengrass.
12. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).
13. In Gruppi Greengrass, scegli il tuo gruppo.
14. Scegli Impostazioni, quindi scegli Ruolo associato.
15. Scegliere Greengrass_Group_Roledal tuo elenco di ruoli, quindi scegli Ruolo associato.

Creare e configurare la funzione Lambda

In questo passaggio, crei una funzione Lambda che tiene traccia del numero di auto che superano il semaforo. Ogni volta che passa allo stato GG_TrafficLight ombraG, la funzione Lambda simula il passaggio di un numero casuale di auto (da 1 a 20). Ogni terzo cambio di G luce, la funzione Lambda invia statistiche di base, come min e max, a una tabella DynamoDB.

1. Sul computer, crea una cartella denominata `car_aggregator`.
2. Dalla cartella degli [TrafficLight](#) esempi in poi GitHub, scarica il `carAggregator.py` file nella cartella. `car_aggregator` Questo è il codice della tua funzione Lambda.

Note

Questo file Python di esempio è archiviato nel repository AWS IoT Greengrass Core SDK per comodità, ma non utilizza Core SDK. AWS IoT Greengrass

3. Se non lavorate nella regione Stati Uniti orientali (Virginia settentrionale), aprite `carAggregator.py` e modificate la riga seguente con Regione AWS quella attualmente selezionata `region_name` nella console. AWS IoT Per l'elenco dei servizi supportati Regione AWS, vedi [AWS IoT Greengrass](#)in. Riferimenti generali di Amazon Web Services

```
dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
```

4. Esegui il comando seguente in una finestra [della riga](#) di comando per installare il [AWS SDK per Python \(Boto3\)](#) pacchetto e le sue dipendenze nella cartella. `car_aggregator` Le funzioni

Greengrass Lambda utilizzano l' AWS SDK per accedere ad altri servizi. AWS Per Windows, utilizza un [prompt di comandi elevati](#).

```
pip install boto3 -t path-to-car_aggregator-folder
```

Questo risulta in un elenco di directory simile al seguente:

Name	Date modified	Type
 bin	12/31/2018 2:27 PM	File folder
 boto3	12/31/2018 2:27 PM	File folder
 boto3-1.9.71.dist-info	12/31/2018 2:27 PM	File folder
 botocore	12/31/2018 2:27 PM	File folder
 botocore-1.12.71.dist-info	12/31/2018 2:27 PM	File folder
 concurrent	12/31/2018 2:27 PM	File folder
 dateutil	12/31/2018 2:27 PM	File folder
 docutils	12/31/2018 2:27 PM	File folder
 docutils-0.14.dist-info	12/31/2018 2:27 PM	File folder
 futures-3.2.0.dist-info	12/31/2018 2:27 PM	File folder
 jmespath	12/31/2018 2:27 PM	File folder
 jmespath-0.9.3.dist-info	12/31/2018 2:27 PM	File folder
 python_dateutil-2.7.5.dist-info	12/31/2018 2:27 PM	File folder
 s3transfer	12/31/2018 2:27 PM	File folder
 s3transfer-0.1.13.dist-info	12/31/2018 2:27 PM	File folder
 six-1.12.0.dist-info	12/31/2018 2:27 PM	File folder
 urllib3	12/31/2018 2:27 PM	File folder
 urllib3-1.24.1.dist-info	12/31/2018 2:27 PM	File folder
 carAggregator.py	12/31/2018 2:25 PM	PY File
 six.py	12/31/2018 2:27 PM	PY File
 six.pyc	12/31/2018 2:27 PM	Compiled Python ...

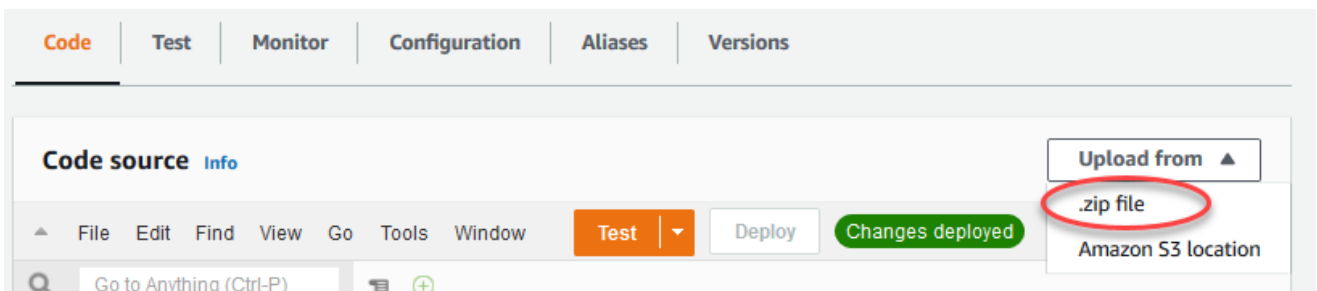
- Comprimi i contenuti della cartella `car_aggregator` in un file `.zip` denominato `car_aggregator.zip`. Comprimi il contenuto della cartella, non la cartella stessa. Questo è il tuo pacchetto di implementazione della funzione Lambda.
- Nella console Lambda, crea una funzione denominata **GG_Car_Aggregator** e imposta i campi rimanenti come segue:
 - In Runtime, scegliere Python 3.7.
 - Per le autorizzazioni, mantieni l'impostazione predefinita. Questo crea un ruolo di esecuzione che concede le autorizzazioni Lambda di base. Questo ruolo non viene utilizzato da AWS IoT Greengrass

Scegli Crea funzione.

The screenshot shows the 'Basic information' tab of the AWS Lambda 'Create function' wizard. It includes a 'Function name' field with the value 'GG_Car_Aggregator', a 'Runtime' dropdown menu set to 'Python 3.7', and a 'Permissions' section with a link to 'Choose or create an execution role'. At the bottom right, there are 'Cancel' and 'Create function' buttons, with the latter highlighted by a red rectangle.

7. Carica il pacchetto di distribuzione della funzione Lambda:

- Nella scheda Codice, in Codice sorgente, scegli Carica da. Dal menu a discesa, scegli un file.zip.



- Scegli il caricamento, quindi scegli il pacchetto di `car_aggregator.zip` distribuzione. Quindi, scegliere Save (Salva).
 - Nella scheda Codice della funzione, in Impostazioni di runtime, scegliete Modifica, quindi immettete i seguenti valori.
 - In Runtime, scegliere Python 3.7.
 - In Handler (Gestore), immetti **`carAggregator.function_handler`**
 - Seleziona Salva.
8. Pubblica la funzione Lambda, quindi crea un alias denominato. **GG_CarAggregator** Per step-by-step istruzioni, consulta i passaggi per [pubblicare la funzione Lambda](#) e [creare un alias](#) nel Modulo 3 (parte 1).

9. Nella AWS IoT console, aggiungi la funzione Lambda che hai appena creato al tuo AWS IoT Greengrass gruppo:
 - a. Nella pagina di configurazione del gruppo, scegli Funzioni Lambda, quindi in Le mie funzioni Lambda, scegli Aggiungi.
 - b. Per la funzione Lambda, scegli GG_CAR_Aggregator.
 - c. Per la versione della funzione Lambda, scegli l'alias della versione che hai pubblicato.
 - d. Per Memory limit (Limite memoria), immettere **64 MB**.
 - e. Per Pinned, scegli True.
 - f. Scegli Aggiungi funzione Lambda.

 Note

Puoi rimuovere altre funzioni Lambda dai moduli precedenti.

Configurazione delle sottoscrizioni

In questo passaggio, si crea una sottoscrizione che consente a GG_TrafficLight shadow di inviare informazioni aggiornate sullo stato alla funzione GG_CAR_Aggregator Lambda. Questa sottoscrizione viene aggiunta alle sottoscrizioni create nel [Modulo 5](#), che sono tutte necessarie per questo modulo.

1. Nella pagina di configurazione del gruppo, scegli la scheda Sottoscrizioni, quindi scegli Aggiungi.
2. Nella pagina Crea un abbonamento, procedi come segue:
 - a. Per Tipo di origine, scegli Servizio, quindi scegli Local Shadow Service.
 - b. Per Tipo di destinazione, scegli la funzione Lambda, quindi scegli GG_Car_Aggregator.
 - c. In Filtro di argomenti, immetti **\$aws/things/GG_TrafficLight/shadow/update/documents**.
 - d. Scegli Crea sottoscrizione.

Questo modulo richiede la nuova sottoscrizione e le [sottoscrizioni](#) create nel Modulo 5.

3. Assicuratevi che il demone Greengrass sia in esecuzione, come descritto in. [Distribuire configurazioni cloud su un dispositivo core](#)
4. Nella pagina di configurazione del gruppo, scegliete Deploy.

Test delle comunicazioni

1. Sul computer, apri due finestre [a riga di comando](#). Proprio come nel [Modulo 5](#), una finestra è per il dispositivo client GG_Switch e l'altra è per il dispositivo client TrafficLight GG_. Vengono usati per eseguire gli stessi comandi eseguiti nel Modulo 5.

Esegui i seguenti comandi per il dispositivo client GG_Switch:

```
cd path-to-certs-folder  
python lightController.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem  
--cert switchCertId-certificate.pem.crt --key switchCertId-private.pem.key --  
thingName GG_TrafficLight --clientId GG_Switch
```

Esegui i seguenti comandi per il dispositivo client GG_: TrafficLight

```
cd path-to-certs-folder  
python trafficLight.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem --  
cert lightCertId-certificate.pem.crt --key lightCertId-private.pem.key --thingName  
GG_TrafficLight --clientId GG_TrafficLight
```

Ogni 20 secondi, lo switch aggiorna lo stato shadow a G, Y e R e la luce mostra il nuovo stato.

2. Il gestore di funzioni della funzione Lambda viene attivato ogni tre semafori verdi (ogni tre minuti) e viene creato un nuovo record DynamoDB. Dopo `trafficLight.py` aver eseguito `lightController.py` per tre minuti, vai alla console DynamoDB e apri la console DynamoDB. AWS Management Console
3. Scegli Stati Uniti orientali (Virginia settentrionale) nel menu. Regione AWS Questa è la regione dove la funzione GG_Car_Aggregator crea la tabella.
4. Nel riquadro di navigazione, scegli Tabelle, quindi scegli la CarStatstabella.
5. Scegli Visualizza elementi per visualizzare le voci nella tabella.

Dovresti visualizzare le voci con statistiche di base sulle auto passate (una voce ogni tre minuti). Potrebbe essere necessario scegliere il pulsante di aggiornamento per visualizzare gli aggiornamenti alla tabella.

6. Se il test non viene completato correttamente, è possibile cercare le informazioni di risoluzione dei problemi nei log di Greengrass.

- a. Passare all'utente root e navigare alla directory log. L'accesso ai AWS IoT Greengrass log richiede i permessi di root.

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

- b. Controllare in runtime.log se ci sono errori.

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

- c. Controlla il registro generato dalla funzione Lambda.

```
cat user/region/account-id/GG_Car_Aggregator.log
```

Gli script `trafficLight.py` e `lightController.py` memorizzano informazioni di connessione nella cartella `groupCA`, che viene creata nella stessa cartella degli script. Se ricevi errori di connessione, assicurati che l'indirizzo IP nel `ggc-host` file corrisponda all'endpoint dell'indirizzo IP del tuo core.

Per ulteriori informazioni, consulta [Risoluzione dei problemi](#).

Questo è il termine del tutorial di base. Ora dovresti comprendere il modello di AWS IoT Greengrass programmazione e i suoi concetti fondamentali, inclusi AWS IoT Greengrass core, gruppi, sottoscrizioni, dispositivi client e il processo di distribuzione delle funzioni Lambda in esecuzione sull'edge.

È possibile eliminare la tabella DynamoDB e le funzioni e gli abbonamenti Greengrass Lambda. Per interrompere le comunicazioni tra il dispositivo AWS IoT Greengrass principale e il AWS IoT cloud, apri un terminale sul dispositivo principale ed esegui uno dei seguenti comandi:

- Per spegnere il dispositivo AWS IoT Greengrass principale:

```
sudo halt
```

- Per fermare il AWS IoT Greengrass demone:

```
cd /greengrass/ggc/core/
```



```
sudo ./greengrassd stop
```

Modulo 7: simulazione dell'integrazione di sicurezza hardware

Questa funzionalità è disponibile per AWS IoT Greengrass Core v1.7 e versioni successive.

In questo modulo avanzato viene illustrato come configurare un modulo di sicurezza hardware (HSM) simulato per l'uso con un core Greengrass. La configurazione utilizza SoftHSM, che è un'implementazione software che utilizza l'API [PKCS#11](#). Lo scopo di questo modulo è quello di consentirti di configurare un ambiente in cui potrai apprendere ed effettuare i test iniziali grazie a un'implementazione software dell'API PKCS#11. Il modulo viene fornito solo a scopo informativo e per il test iniziale e non per scopi produttivi di alcun tipo.

Puoi utilizzare questa configurazione per effettuare esperimenti con l'utilizzo di un servizio compatibile con PKCS#11 per l'archiviazione delle chiavi private. Per ulteriori informazioni sulle implementazioni software, consulta [SoftHSM](#). Per ulteriori informazioni sull'integrazione della sicurezza hardware su un AWS IoT Greengrass core, inclusi i requisiti generali, consulta [the section called "Integrazione della sicurezza hardware"](#)

Important

Questo modulo viene fornito unicamente a scopo di sperimentazione. Sconsigliamo l'utilizzo di SoftHSM in un ambiente di produzione, in quanto potrebbe fornire un falso senso di maggiore sicurezza. La configurazione risultante non fornisce prestazioni di sicurezza effettive. Le chiavi archiviate in SoftHSM non sono archiviate in modo più sicuro rispetto ad altri mezzi di storage dei segreti dell'ambiente Greengrass.

Lo scopo di questo modulo è quello di fornire nozioni di base sulla specifica PKCS#11 e di eseguire il test iniziale del software, se prevedi di utilizzare in futuro un vero HSM basato sull'hardware.

Dovrai effettuare il test della futura implementazione hardware separatamente e in modo completo prima di qualsiasi utilizzo in produzione, poiché potrebbero esservi differenze tra l'implementazione PKCS#11 fornita in SoftHSM e un'implementazione basata sull'hardware.

Se hai bisogno di assistenza per l'onboarding di un [modulo di sicurezza hardware supportato](#), contatta il tuo rappresentante Enterprise AWS Support.

Prima di iniziare, eseguire lo script di [installazione dispositivo Greengrass](#) o assicurarsi di aver completato il [modulo 1](#) e il [modulo 2](#) dell'esercitazione introduttiva. In questo modulo, si presuppone che il core sia già predisposto e con cui sia in comunicazione. AWS Il completamento di questo modulo richiede circa 30 minuti.

Installazione del software SoftHSM

In questa fase, installerai SoftHSM e gli strumenti pkcs11, utilizzati per gestire l'istanza SoftHSM.

- In un terminale del dispositivo AWS IoT Greengrass principale, esegui il seguente comando:

```
sudo apt-get install softhsm2 libsofthsm2-dev pkcs11-dump
```

Per ulteriori informazioni su questi pacchetti, consulta [Installazione di softhsm2](#), [Installazione di libsofthsm2-dev](#) e [Installazione di pkcs11-dump](#).

Note

Se si riscontrano problemi durante l'utilizzo di questo comando sul sistema, vedere [SoftHSM versione 2](#) in poi. GitHub Questo sito fornisce ulteriori informazioni di installazione, incluse quelle relative alla compilazione a partire dal codice sorgente.

Configurazione di SoftHSM

In questa fase, [configurerai SoftHSM](#).

1. Accedi come utente root.

```
sudo su
```

2. Utilizza la pagina del manuale per trovare la posizione `softhsm2.conf` a livello di sistema. Una posizione comune è `/etc/softhsm/softhsm2.conf`, ma la posizione potrebbe essere diversa su alcuni sistemi.

```
man softhsm2.conf
```

3. Crea la directory per il file di configurazione `softhsm2` nella posizione a livello di sistema. In questo esempio si presuppone che la posizione sia `/etc/softhsm/softhsm2.conf`.

```
mkdir -p /etc/softhsm
```

4. Crea la directory del token nella directory /greengrass.

Note

Se questa fase viene ignorata, softhsm2-util riporta `ERROR: Could not initialize the library.`

```
mkdir -p /greengrass/softhsm2/tokens
```

5. Configurare la directory del token.

```
echo "directories.tokenidir = /greengrass/softhsm2/tokens" > /etc/softhsm/softhsm2.conf
```

6. Configurare un back-end basato su file.

```
echo "objectstore.backend = file" >> /etc/softhsm/softhsm2.conf
```

Note

Queste impostazioni di configurazione vengono fornite unicamente a scopo di sperimentazione. Per visualizzare tutte le opzioni di configurazione, leggi la pagina del manuale del file di configurazione.

```
man softhsm2.conf
```

Importazione della chiave privata in SoftHSM.

In questa fase, inizierai il token SoftHSM, convertirai il formato della chiave privata, quindi importerai la chiave privata.

1. Inizializzare il token SoftHSM.

```
softhsm2-util --init-token --slot 0 --label greengrass --so-pin 12345 --pin 1234
```

Note

Se richiesto, immettere il PIN SO 12345 e il PIN utente 1234. AWS IoT Greengrass non utilizza il PIN SO (supervisore), pertanto è possibile usare qualsiasi valore.

Se si riceve l'errore `CKR_SLOT_ID_INVALID: Slot 0 does not exist`, provare il seguente comando:

```
softhsm2-util --init-token --free --label greengrass --so-pin 12345 --pin 1234
```

- Convertire la chiave privata in un formato che possa essere utilizzato dallo strumento di importazione di SoftHSM. In questo tutorial, convertirai la chiave privata ottenuta con l'opzione Creazione gruppo predefinito del [Modulo 2](#) del tutorial Nozioni di base.

```
openssl pkcs8 -topk8 -inform PEM -outform PEM -nocrypt -in hash.private.key -  
out hash.private.pem
```

- Importare la chiave privata in SoftHSM. Eseguire solo uno dei seguenti comandi, a seconda della versione di softhsm2-util.

Sintassi Raspbian softhsm2-util v2.2.0

```
softhsm2-util --import hash.private.pem --token greengrass --label iotkey --id  
0000 --pin 12340
```

Sintassi Ubuntu softhsm2-util v2.0.0

```
softhsm2-util --import hash.private.pem --slot 0 --label iotkey --id 0000 --pin  
1234
```

Questo comando identifica lo slot come 0 e definisce l'etichetta della chiave come iotkey. Utilizzerai questi valori nella sezione successiva.

Dopo avere importato la chiave privata, potrai rimuoverla facoltativamente dalla directory `/greengrass/certs`. Assicurati di conservare il certificato CA root e quello del dispositivo nella directory.

Configurazione del core Greengrass per l'utilizzo di SoftHSM

In questa fase, modificherai la configurazione core di Greengrass per utilizzare SoftHSM.

1. Individuare il percorso della libreria del provider SoftHSM (`libsoftism2.so`) nel sistema:

- a. Ottenere l'elenco dei pacchetti installati per la libreria.

```
sudo dpkg -l libsoftism2
```

Il file `libsoftism2.so` è disponibile nella directory `softism`.

- b. Copiare il percorso completo del file (ad esempio, `/usr/lib/x86_64-linux-gnu/softism/libsoftism2.so`). Utilizzerai questo valore in un secondo momento.

2. Arrestare il daemon Greengrass.

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

3. Aprire il file di configurazione Greengrass. Si tratta del file [config.json](#) nella directory `/greengrass/config`.

Note

Gli esempi in questa procedura sono scritti presupponendo che il file `config.json` utilizzi il formato generato dall'opzione Creazione gruppo definito del [Modulo 2](#) del tutorial Nozioni di base.

4. Nell'oggetto `crypto.principals`, inserire il seguente oggetto del certificato del server MQTT. Aggiungere una virgola dove necessario per creare un file JSON valido.

```
"MQTTServerCertificate": {  
  "privateKeyPath": "path-to-private-key"  
}
```

5. Nell'oggetto `crypto`, inserisci il seguente oggetto PKCS11. Aggiungere una virgola dove necessario per creare un file JSON valido.

```
"PKCS11": {
  "P11Provider": "/path-to-pkcs11-provider-so",
  "slotLabel": "crypto-token-name",
  "slotUserPin": "crypto-token-user-pin"
}
```

Il file si presenta in maniera simile a quanto riportato di seguito:

```
{
  "coreThing" : {
    "caPath" : "root.ca.pem",
    "certPath" : "hash.cert.pem",
    "keyPath" : "hash.private.key",
    "thingArn" : "arn:partition:iot:region:account-id:thing/core-thing-name",
    "iotHost" : "host-prefix.iot.region.amazonaws.com",
    "ggHost" : "greengrass.iot.region.amazonaws.com",
    "keepAlive" : 600
  },
  "runtime" : {
    "cgroup" : {
      "useSystemd" : "yes"
    }
  },
  "managedRespawn" : false,
  "crypto": {
    "PKCS11": {
      "P11Provider": "/path-to-pkcs11-provider-so",
      "slotLabel": "crypto-token-name",
      "slotUserPin": "crypto-token-user-pin"
    },
    "principals" : {
      "MQTTServerCertificate": {
        "privateKeyPath": "path-to-private-key"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key",
        "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
      },
      "SecretsManager" : {
```

```
    "privateKeyPath" : "file:///greengrass/certs/hash.private.key"
  },
  "caPath" : "file:///greengrass/certs/root.ca.pem"
}
```

Note

Per utilizzare gli aggiornamenti over-the-air (OTA) con sicurezza hardware, l'PKCS11oggetto deve contenere anche la OpenSSLEngine proprietà. Per ulteriori informazioni, consulta [the section called "Configurazione degli aggiornamenti OTA"](#).

6. Modificare l'oggetto crypto:

a. Configura l'oggetto PKCS11.

- In P11Provider, immettere il percorso completo di libsofthsm2.so.
- In slotLabel, immettere greengrass.
- In slotUserPin, immettere 1234.

b. Configurare i percorsi delle chiavi private dell'oggetto principals. Non modificare la proprietà certificatePath.

- Nelle proprietà privateKeyPath, immettere il seguente percorso PKCS#11 RFC 7512 (che specifica l'etichetta della chiave). Effettuare questa operazione per i principali IoTCertificate, SecretsManager e MQTTServerCertificate.

```
pkcs11:object=iotkey;type=private
```

c. Controllare l'oggetto crypto. La schermata visualizzata dovrebbe risultare simile a quella nell'immagine seguente:

```
"crypto": {
  "PKCS11": {
    "P11Provider": "/usr/lib/x86_64-linux-gnu/softhsm/libsofthsm2.so",
    "slotLabel": "greengrass",
    "slotUserPin": "1234"
  },
  "principals": {
    "MQTTServerCertificate": {
```

```

    "privateKeyPath": "pkcs11:object=iotkey;type=private"
  },
  "SecretsManager": {
    "privateKeyPath": "pkcs11:object=iotkey;type=private"
  },
  "IoTCertificate": {
    "certificatePath": "file://certs/core.crt",
    "privateKeyPath": "pkcs11:object=iotkey;type=private"
  }
},
"caPath": "file://certs/root.ca.pem"
}

```

7. Rimuovere i valori `caPath`, `certPath` e `keyPath` dall'oggetto `coreThing`. La schermata visualizzata dovrebbe risultare simile a quella nell'immagine seguente:

```

"coreThing" : {
  "thingArn" : "arn:partition:iot:region:account-id:thing/core-thing-name",
  "iotHost" : "host-prefix-ats.iot.region.amazonaws.com",
  "ggHost" : "greengrass-ats.iot.region.amazonaws.com",
  "keepAlive" : 600
}

```

Note

In questo tutorial, specificherai la stessa chiave privata per tutti i principali. Per ulteriori informazioni sulla scelta della chiave privata per il server MQTT locale, consulta [Prestazioni](#). Per ulteriori informazioni sul Secrets Manager locale, consulta [Distribuzione dei segreti nel core](#).

Test della configurazione

- Avviare il daemon Greengrass.

```

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

```

Se il daemon viene avviato, il core è stato configurato con successo.

A questo punto puoi acquisire familiarità con la specifica PKCS#11 ed effettuare il test iniziale con l'API PKCS#11 fornita dall'implementazione SoftHSM.

 Important

Ribadiamo che è estremamente importante sapere che questo modulo è destinato solamente all'apprendimento e al test. Non aumenta la sicurezza dell'ambiente Greengrass.

Lo scopo del modulo è quello di consentirti di acquisire familiarità e di effettuare test in vista dell'utilizzo futuro di un vero HSM basato sull'hardware. In quel momento, dovrai effettuare il test del software rispetto all'HSM basato sull'hardware separatamente e in modo completo prima di qualsiasi utilizzo produttivo, poiché potrebbero esservi differenze tra l'implementazione PKCS#11 fornita in SoftHSM e una basata sull'hardware.

Consulta anche

- PKCS #11 Cryptographic Token Interface Usage Guide Version 2.40. Pubblicato da John Leiseboer e Robert Griffin. 16 Novembre 2014. Nota 02 del Comitato OASIS. <http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/cn02/pkcs11-ug-v2.40-cn02.html>. Ultima versione: <http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html>.
- [RFC 7512](#)

Aggiornamenti OTA del software AWS IoT Greengrass Core

Il pacchetto software AWS IoT Greengrass Core include un agente di aggiornamento in grado di eseguire aggiornamenti over-the-air (OTA) del AWS IoT Greengrass software. È possibile utilizzare gli aggiornamenti OTA per installare la versione più recente del software AWS IoT Greengrass Core o del software OTA Update Agent su uno o più core. Con gli aggiornamenti OTA, i dispositivi core non devono essere fisicamente presenti.

Si consiglia di utilizzare gli aggiornamenti OTA quando possibile. Forniscono un meccanismo che è possibile utilizzare per tenere traccia dello stato degli aggiornamenti e della relativa cronologia. Se si verifica un aggiornamento non riuscito, l'agente di aggiornamento OTA torna alla versione precedente del software.

Note

Gli aggiornamenti OTA non sono supportati quando si utilizza apt per installare il software AWS IoT Greengrass Core. Per queste installazioni, si consiglia di utilizzare apt per aggiornare il software. Per ulteriori informazioni, consulta [the section called “Installazione da un repository APT”](#).

Gli aggiornamenti OTA rendono più efficiente:

- la correzione delle vulnerabilità in termini di sicurezza;
- la risoluzione dei problemi di stabilità del software;
- la distribuzione delle funzionalità nuove o migliorate.

Questa funzionalità si integra con i processi [AWS IoT](#).

Requisiti

I seguenti requisiti si applicano agli aggiornamenti OTA del AWS IoT Greengrass software.

- Il core Greengrass deve avere almeno 400 MB di spazio su disco disponibile nella memoria locale. L'agente di aggiornamento OTA richiede circa tre volte il requisito di utilizzo del runtime del software AWS IoT Greengrass Core. Per ulteriori informazioni, consulta [Quote di servizio](#) per il core di Greengrass in Riferimenti generali di Amazon Web Services.

- Il core Greengrass deve avere una connessione con. Cloud AWS
- Il core Greengrass deve essere configurato correttamente e dotato di certificati e chiavi per l'autenticazione con e. AWS IoT Core AWS IoT Greengrass Per ulteriori informazioni, consulta [the section called “Certificati X.509”](#).
- Il core di Greengrass non può essere configurato per utilizzare un proxy di rete.

 Note

A partire dalla AWS IoT Greengrass versione 1.9.3, gli aggiornamenti OTA sono supportati sui core che configurano il traffico MQTT per utilizzare la porta 443 anziché la porta predefinita 8883. Tuttavia, l'agente di aggiornamento OTA non supporta gli aggiornamenti tramite un proxy di rete. Per ulteriori informazioni, consulta [the section called “Connessione alla porta 443 o tramite un proxy di rete”](#).

- L'avvio affidabile non può essere abilitato nella partizione che contiene il software AWS IoT Greengrass Core.

 Note

È possibile installare ed eseguire il software AWS IoT Greengrass Core su una partizione con avvio affidabile abilitato, ma gli aggiornamenti OTA non sono supportati.

- AWS IoT Greengrass deve disporre delle autorizzazioni di lettura/scrittura sulla partizione che contiene il software Core. AWS IoT Greengrass
- Se si utilizza un sistema init per gestire il core di Greengrass, è necessario configurare gli aggiornamenti OTA per l'integrazione con il sistema init. Per ulteriori informazioni, consulta [the section called “Integrazione con i sistemi di inizializzazione”](#).
- È necessario creare un ruolo che venga utilizzato per preassegnare Amazon URLs S3 AWS IoT Greengrass agli artefatti di aggiornamento software. Questo ruolo di firmatario consente di accedere AWS IoT Core agli elementi di aggiornamento software archiviati in Amazon S3 per tuo conto. Per ulteriori informazioni, consulta [the section called “Autorizzazioni IAM per gli aggiornamenti OTA”](#).

Autorizzazioni IAM per gli aggiornamenti OTA

Quando AWS IoT Greengrass rilascia una nuova versione del software AWS IoT Greengrass Core, AWS IoT Greengrass aggiorna gli artefatti software archiviati in Amazon S3 che vengono utilizzati per l'aggiornamento OTA.

Account AWS È necessario includere un ruolo di firmatario URL di Amazon S3 che può essere utilizzato per accedere a questi elementi. Il ruolo deve avere una politica di autorizzazioni che consenta l'`s3:GetObject` sui bucket nelle destinazioni. Regione AWS Il ruolo deve inoltre disporre di una policy di attendibilità che consenta a `iot.amazonaws.com` di assumere il ruolo come entità attendibile.

Policy delle autorizzazioni

Per le autorizzazioni dei ruoli, puoi utilizzare la politica AWS gestita o creare una politica personalizzata.

- Utilizza la politica AWS gestita

La politica `OTAUpdate ArtifactAccess` gestita da [Greengrass](#) è fornita da AWS IoT Greengrass. Utilizza questa politica se desideri consentire l'accesso in tutte le regioni di Amazon Web Services supportate da AWS IoT Greengrass, attuali e future.

- Creare una policy personalizzata

È necessario creare una policy personalizzata se si desidera specificare in modo esplicito le regioni Amazon Web Services in cui vengono distribuiti i core. La seguente politica di esempio consente l'accesso agli aggiornamenti AWS IoT Greengrass software in sei regioni.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAccessToGreengrassOTAUpdateArtifacts",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::us-east-1-greengrass-updates/*",
        "arn:aws:s3:::us-west-2-greengrass-updates/*",
        "arn:aws:s3:::ap-northeast-1-greengrass-updates/*",
```

```

        "arn:aws:s3:::ap-southeast-2-greengrass-updates/*",
        "arn:aws:s3:::eu-central-1-greengrass-updates/*",
        "arn:aws:s3:::eu-west-1-greengrass-updates/*"
    ]
}
]
}

```

Policy di trust

La policy di attendibilità associata al ruolo deve consentire l'operazione `sts:AssumeRole` e definire `iot.amazonaws.com` come principale. Ciò consente a AWS IoT Core di assumere il ruolo come entità attendibile. Ecco un esempio di documento di policy:

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowIotToAssumeRole",
      "Action": "sts:AssumeRole",
      "Principal": {
        "Service": "iot.amazonaws.com"
      },
      "Effect": "Allow"
    }
  ]
}

```

Inoltre, l'utente che avvia un aggiornamento OTA deve disporre delle autorizzazioni per utilizzare `greengrass:CreateSoftwareUpdateJob` e `iot:CreateJob`, nonché utilizzare `iam:PassRole` per passare le autorizzazioni del ruolo firmatario. Ecco un esempio di policy IAM:

```

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

```

```
{
  {
    "Effect": "Allow",
    "Action": [
      "iot:CreateJob"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "iam:PassRole"
    ],
    "Resource": "arn-of-s3-url-signer-role"
  }
]
```

Considerazioni

Prima di avviare un aggiornamento OTA del software Greengrass Core, è necessario conoscere l'impatto sui dispositivi del gruppo Greengrass, sia sul dispositivo core che sui dispositivi client collegati localmente al core:

- Il core si arresta durante l'aggiornamento.
- Le funzioni Lambda in esecuzione nel core saranno terminate. Se queste funzioni scrivono su risorse locali, potrebbero lasciare tali risorse in uno stato errato, a meno che non vengano interrotte correttamente.
- Durante il periodo di inattività del core, tutte le sue connessioni con il core Cloud AWS vengono perse. I messaggi instradati attraverso il core dai dispositivi client vengono persi.
- Le cache delle credenziali vengono perse.
- Le code che contengono processi in sospeso per le funzioni Lambda vengono perse.
- Le funzioni Lambda di lunga durata perdono le informazioni sullo stato dinamico e tutto il lavoro in sospeso viene interrotto.

Le seguenti informazioni di stato vengono conservate durante un aggiornamento OTA:

- Configurazione Core
- Configurazione gruppo Greengrass

- Shadow locali
- Log Greengrass
- Registri dell'agente di aggiornamento OTA

Agente di aggiornamento OTA di Greengrass

L'agente di aggiornamento OTA Greengrass è il componente software del dispositivo che gestisce i processi di aggiornamento creati e distribuiti nel cloud. L'agente di aggiornamento OTA è distribuito nello stesso pacchetto software del software AWS IoT Greengrass Core. L'agente si trova in `/greengrass-root/ota/ota_agent/ggc-ota`. Scrive i registri in `/var/log/greengrass/ota/ggc_ota.txt`.

Note

`greengrass-root` rappresenta il percorso in cui il software AWS IoT Greengrass Core è installato sul dispositivo. In genere, questa è la directory `/greengrass`.

È possibile avviare l'agente di aggiornamento OTA eseguendo il file binario manualmente o integrandolo come parte di uno script di init, ad esempio un file di servizio systemd. Se si esegue il file binario manualmente, deve essere eseguito come root. All'avvio, l'agente di aggiornamento OTA ascolta i lavori di aggiornamento AWS IoT Greengrass software AWS IoT Core e li esegue in sequenza. L'agente di aggiornamento OTA ignora tutti gli altri tipi di lavoro. AWS IoT

Il seguente estratto mostra un esempio di file di servizio systemd per avviare, arrestare e riavviare l'agente di aggiornamento OTA:

```
[Unit]
Description=Greengrass OTA Daemon

[Service]
Type=forking
Restart=on-failure
ExecStart=/greengrass/ota/ota_agent/ggc-ota

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

Un core che è l'obiettivo di un aggiornamento non deve eseguire due istanze dell'agente di aggiornamento OTA. In questo modo, i due agenti elaborano gli stessi processi creando conflitti.

Integrazione con i sistemi di inizializzazione

Durante un aggiornamento OTA, l'agente di aggiornamento OTA riavvia i file binari sul dispositivo principale. Se i file binari sono in esecuzione, ciò potrebbe causare conflitti quando un sistema init monitora lo stato del software AWS IoT Greengrass Core o dell'agente durante l'aggiornamento. Per aiutarti a integrare il meccanismo di aggiornamento OTA con le tue strategie di monitoraggio init, puoi scrivere script di shell eseguibili prima e dopo un aggiornamento. Ad esempio, è possibile utilizzare `ggc_pre_update.sh` lo script per eseguire il backup dei dati o interrompere i processi prima dello spegnimento del dispositivo.

Per indicare all'agente di aggiornamento OTA di eseguire questi script, è necessario includere il `"managedRespawn" : true` flag nel [file config.json](#). Questa impostazione è mostrata nel seguente estratto:

```
{
  "coreThing": {
    ...
  },
  "runtime": {
    ...
  },
  "managedRespawn": true
  ...
}
```

Rigenerazione gestita con aggiornamenti OTA

I seguenti requisiti si applicano agli aggiornamenti OTA impostati su: `managedRespawn true`

- I seguenti script di shell devono essere presenti nella `/greengrass-root/usr/scripts` directory:
 - `ggc_pre_update.sh`
 - `ggc_post_update.sh`
 - `ota_pre_update.sh`
 - `ota_post_update.sh`

- Gli script devono restituire un codice valido.
- Gli script devono essere di proprietà del root e devono poter essere eseguiti solo da root.
- Lo `ggc_pre_update.sh` script deve fermare il demone Greengrass.
- Lo `ggc_post_update.sh` script deve avviare il demone Greengrass.

Note

Poiché l'agente di aggiornamento OTA gestisce il proprio processo, non è necessario che `ota_post_update.sh` gli script `ota_pre_update.sh` e interrompano o avviino il servizio OTA.

L'agente di aggiornamento OTA esegue gli script da. `/greengrass-root/usr/scripts` La struttura di directory deve essere simile alla seguente:

```
<greengrass_root>
|-- certs
|-- config
|   |-- config.json
|-- ggc
|-- usr/scripts
|   |-- ggc_pre_update.sh
|   |-- ggc_post_update.sh
|   |-- ota_pre_update.sh
|   |-- ota_post_update.sh
|-- ota
```

Quando `managedRespawn` è impostato su `true`, l'agente di aggiornamento OTA verifica la presenza di questi script `/greengrass-root/usr/scripts` nella directory prima e dopo l'aggiornamento del software. Se gli script non esistono, l'aggiornamento non riesce. AWS IoT Greengrass non convalida il contenuto di questi script. Come procedura consigliata, verificate che gli script funzionino correttamente e rilasciate i codici di uscita appropriati per gli errori.

Per gli aggiornamenti OTA del software AWS IoT Greengrass Core:

- Prima di avviare l'aggiornamento, l'agente esegue lo script `ggc_pre_update.sh`. Utilizzate questo script per i comandi che devono essere eseguiti prima che l'agente di aggiornamento OTA avvii l'aggiornamento del software AWS IoT Greengrass Core, ad esempio per eseguire il backup

dei dati o interrompere qualsiasi processo in esecuzione. L'esempio seguente mostra un semplice script per fermare il demone Greengrass.

```
#!/bin/bash
set -euo pipefail
systemctl stop greengrass
```

- Dopo aver completato l'aggiornamento, l'agente esegue lo script `ggc_post_update.sh`. Utilizzate questo script per i comandi che devono essere eseguiti dopo che l'agente di aggiornamento OTA ha avviato l'aggiornamento del software AWS IoT Greengrass Core, ad esempio per riavviare i processi. L'esempio seguente mostra un semplice script per avviare il demone Greengrass.

```
#!/bin/bash
set -euo pipefail
systemctl start greengrass
```

Per gli aggiornamenti OTA dell'agente di aggiornamento OTA:

- Prima di avviare l'aggiornamento, l'agente esegue lo script `ota_pre_update.sh`. Utilizzate questo script per i comandi che devono essere eseguiti prima che l'agente di aggiornamento OTA si aggiorni da solo, ad esempio per eseguire il backup dei dati o interrompere qualsiasi processo in esecuzione.
- Dopo aver completato l'aggiornamento, l'agente esegue lo script `ota_post_update.sh`. Utilizzate questo script per i comandi che devono essere eseguiti dopo l'aggiornamento dell'agente di aggiornamento OTA, ad esempio per riavviare i processi.

Note

Se `managedRespawn` è impostato su `false`, l'agente di aggiornamento OTA non esegue gli script.

Creare un aggiornamento OTA.

Segui questi passaggi per eseguire un aggiornamento OTA del AWS IoT Greengrass software su uno o più core:

1. Assicurarsi che i core soddisfino i [requisiti](#) per gli aggiornamenti OTA.

 Note

Se hai configurato un sistema di inizializzazione per gestire il software AWS IoT Greengrass Core o l'agente di aggiornamento OTA, verifica quanto segue sui core:

- Il file [config.json](#) specifica "managedRespawn" : true.
- La directory `/greengrass-root/usr/scripts` contiene i seguenti script:
 - ggc_pre_update.sh
 - ggc_post_update.sh
 - ota_pre_update.sh
 - ota_post_update.sh

Per ulteriori informazioni, consulta [the section called "Integrazione con i sistemi di inizializzazione"](#).

2. In un terminale del dispositivo principale, avvia l'agente di aggiornamento OTA.

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

 Note

`greengrass-root` rappresenta il percorso in cui il software AWS IoT Greengrass Core è installato sul dispositivo. In genere, questa è la directory `/greengrass`.

Non avviare più istanze dell'agente di aggiornamento OTA su un core perché potrebbe causare conflitti.

3. Utilizza l' AWS IoT Greengrass API per creare un processo di aggiornamento software.
 - a. Chiamata dell'API [CreateSoftwareUpdateJob](#). In questa procedura di esempio, utilizziamo AWS CLI i comandi.

Il comando seguente crea un job che aggiorna il software AWS IoT Greengrass Core su un core. Sostituire i valori di esempio e quindi eseguire il comando.

Linux or macOS terminal

```
aws greengrass create-software-update-job \  
--update-targets-architecture x86_64 \  
--update-targets ["arn:aws:iot:region:123456789012:thing/myCoreDevice\""] \  
--update-targets-operating-system ubuntu \  
--software-to-update core \  
--s3-url-signer-role arn:aws:iam::123456789012:role/myS3UrlSignerRole \  
--update-agent-log-level WARN \  
--amzn-client-token myClientToken1
```

Windows command prompt

```
aws greengrass create-software-update-job ^  
--update-targets-architecture x86_64 ^  
--update-targets ["arn:aws:iot:region:123456789012:thing/myCoreDevice\""] ^  
--update-targets-operating-system ubuntu ^  
--software-to-update core ^  
--s3-url-signer-role arn:aws:iam::123456789012:role/myS3UrlSignerRole ^  
--update-agent-log-level WARN ^  
--amzn-client-token myClientToken1
```

Questo comando restituisce la risposta seguente.

```
{  
  "IotJobId": "GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",  
  "IotJobArn": "arn:aws:iot:region:123456789012:job/  
GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",  
  "PlatformSoftwareVersion": "1.10.1"  
}
```

- b. Copiare IotJobId dalla risposta.
- c. Chiama [DescribeJob](#) AWS IoT Core API per vedere lo stato del lavoro. Sostituire il valore di esempio con l'ID del processo e quindi eseguire il comando.

```
aws iot describe-job --job-id GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-  
a1da0EXAMPLE
```

Il comando restituisce un oggetto di risposta che contiene informazioni sul processo, tra cui `status` e `jobProcessDetails`.

```
{
  "job": {
    "jobArn": "arn:aws:iot:region:123456789012:job/GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
    "jobId": "GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
    "targetSelection": "SNAPSHOT",
    "status": "IN_PROGRESS",
    "targets": [
      "arn:aws:iot:region:123456789012:thing/myCoreDevice"
    ],
    "description": "This job was created by Greengrass to update the Greengrass Cores in the targets with version 1.10.1 of the core software running on x86_64 architecture.",
    "presignedUrlConfig": {
      "roleArn": "arn:aws::iam::123456789012:role/myS3UrlSignerRole",
      "expiresInSec": 3600
    },
    "jobExecutionsRolloutConfig": {},
    "createdAt": 1588718249.079,
    "lastUpdatedAt": 1588718253.419,
    "jobProcessDetails": {
      "numberOfCanceledThings": 0,
      "numberOfSucceededThings": 0,
      "numberOfFailedThings": 0,
      "numberOfRejectedThings": 0,
      "numberOfQueuedThings": 1,
      "numberOfInProgressThings": 0,
      "numberOfRemovedThings": 0,
      "numberOfTimedOutThings": 0
    },
    "timeoutConfig": {}
  }
}
```

Per la risoluzione dei problemi, consultare [Risoluzione dei problemi](#).

CreateSoftwareUpdateJob API

Puoi utilizzare l>CreateSoftwareUpdateJobAPI per aggiornare il software AWS IoT Greengrass Core o il software OTA Update Agent sui tuoi dispositivi principali. Questa API crea un processo di snapshot AWS IoT che notifica ai dispositivi quando è disponibile un aggiornamento. Dopo la chiamataCreateSoftwareUpdateJob, puoi utilizzare altri comandi di AWS IoT job per tenere traccia dell'aggiornamento del software. Per ulteriori informazioni, consulta [Jobs](#) nella AWS IoT Developer Guide.

L'esempio seguente mostra come utilizzare il AWS CLI per creare un job che aggiorni il software AWS IoT Greengrass Core su un dispositivo principale:

```
aws greengrass create-software-update-job \
--update-targets-architecture x86_64 \
--update-targets ["arn:aws:iot:region:123456789012:thing/myCoreDevice\""] \
--update-targets-operating-system ubuntu \
--software-to-update core \
--s3-url-signer-role arn:aws:iam::123456789012:role/myS3UrlSignerRole \
--update-agent-log-level WARN \
--amzn-client-token myClientToken1
```

Il comando create-software-update-job restituisce una risposta JSON contenente l'ID processo, l'ARN del processo e la versione software che sono stati installati dall'aggiornamento:

```
{
  "IotJobId": "GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
  "IotJobArn": "arn:aws:iot:region:123456789012:job/GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
  "PlatformSoftwareVersion": "1.9.2"
}
```

Per i passaggi che illustrano come utilizzare create-software-update-job per aggiornare un dispositivo core, consulta [the section called “Creare un aggiornamento OTA.”](#).

Il comando create-software-update-job ha i seguenti parametri:

--update-targets-architecture

L'architettura del dispositivo core.

Valori validi: armv7l, armv6l, x86_64 o aarch64

--update-targets

I core da aggiornare. L'elenco può contenere ARNs singoli core e gruppi ARNs di cose i cui membri sono core. Per ulteriori informazioni sui gruppi di oggetti, consulta [Static thing group](#) nella AWS IoT Developer Guide.

--update-targets-operating-system

Il sistema operativo del dispositivo Core.

Valori validi: `ubuntu`, `amazon_linux`, `raspbian` o `openwrt`

--software-to-update

Specifica se il software del core o il software dell'agente di aggiornamento OTA devono essere aggiornati.

Valori validi: `core` o `ota_agent`

--s3-url-signer-role

L'ARN del ruolo IAM utilizzato per preassegnare l'URL di Amazon S3 che collega agli elementi dell'aggiornamento software. AWS IoT Greengrass La politica di autorizzazioni allegata al ruolo deve consentire `s3:GetObject` sui bucket nelle destinazioni. Regione AWS Il ruolo deve inoltre consentire a `iot.amazonaws.com` di assumere il ruolo come entità attendibile. Per ulteriori informazioni, consulta [the section called "Autorizzazioni IAM per gli aggiornamenti OTA"](#).

--amzn-client-token

(Facoltativo) Un token client utilizzato per effettuare le richieste. Fornisci un token unico per impedire la creazione di aggiornamenti duplicati a causa di nuovi tentativi interni.

--update-agent-log-level

(Facoltativo) Il livello di registrazione per le istruzioni di registro generate dall'agente di aggiornamento OTA. Il valore predefinito è `ERROR`.

Valori validi: `NONE`, `TRACE`, `DEBUG`, `VERBOSE`, `INFO`, `WARN`, `ERROR` o `FATAL`

Note

`CreateSoftwareUpdateJob` accetta richieste solo per le seguenti combinazioni di architettura e sistema operativo supportate:

- ubuntu/x86_64
- ubuntu/aarch64
- amazon_linux/x86_64
- raspbian/armv7l
- raspbian/armv6l
- openwrt/aarch64
- openwrt/armv7l

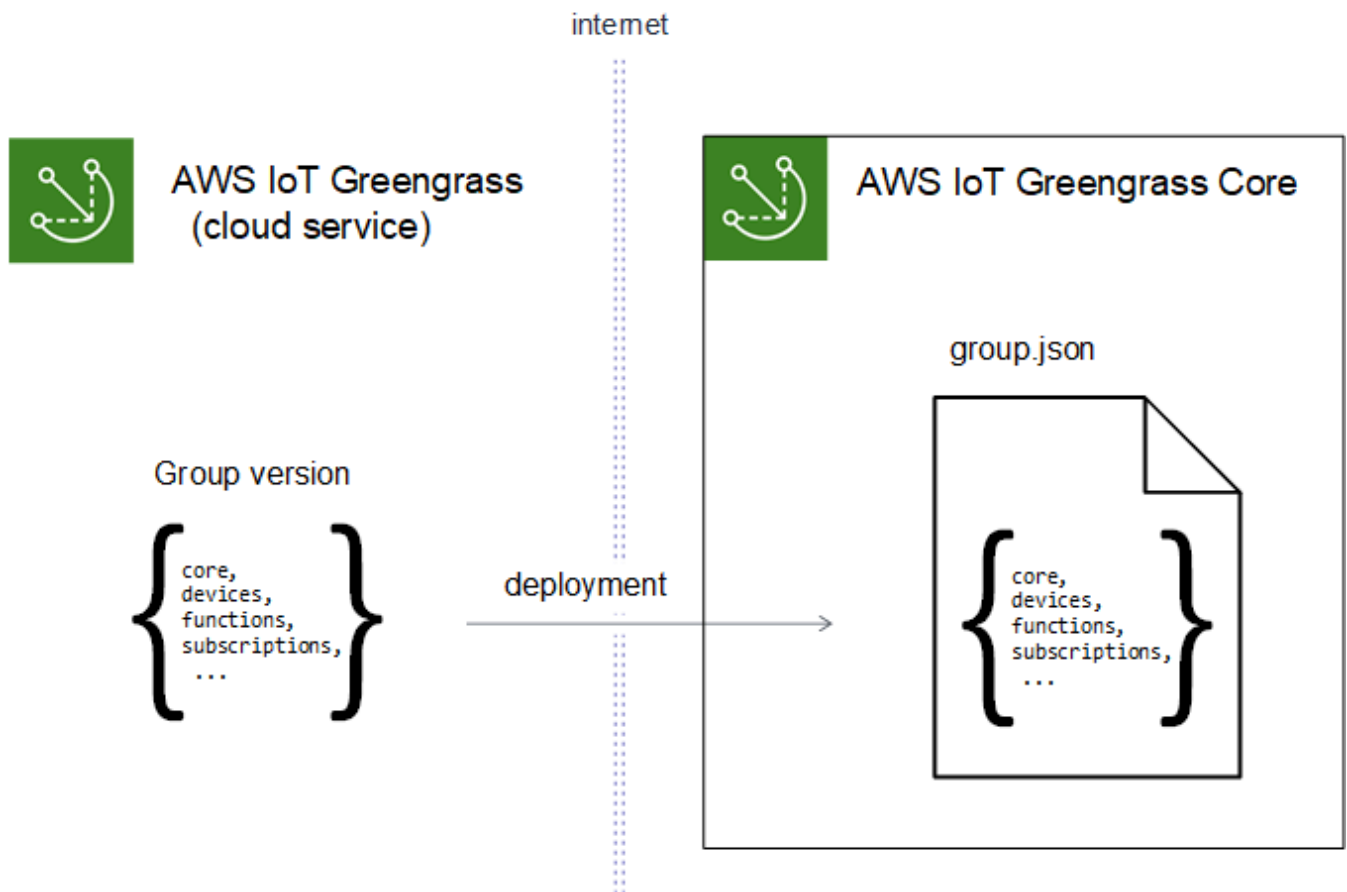
Distribuisci AWS IoT Greengrass i gruppi su un core AWS IoT Greengrass

Usa AWS IoT Greengrass i gruppi per organizzare le entità nel tuo ambiente edge. I gruppi vengono utilizzati anche per controllare il modo in cui le entità del gruppo interagiscono tra loro e con Cloud AWS. Ad esempio, solo le funzioni Lambda del gruppo vengono distribuite per l'esecuzione locale e solo i dispositivi del gruppo possono comunicare utilizzando il server MQTT locale.

Un gruppo deve includere un [core](#), ovvero un AWS IoT dispositivo che esegue il AWS IoT Greengrass software Core. Il core funge da gateway perimetrale e fornisce AWS IoT Core funzionalità nell'ambiente perimetrale. A seconda delle esigenze aziendali, puoi anche aggiungere le seguenti entità a un gruppo:

- Dispositivi client. Rappresentati come oggetti nel registro AWS IoT . Questi dispositivi devono eseguire [FreerTOS](#) o utilizzare [AWS IoT Device SDK](#) [AWS IoT Greengrass o Discovery](#) API per ottenere informazioni di connessione per il core. Solo i dispositivi client che fanno parte del gruppo possono connettersi al core.
- Funzioni Lambda. Applicazioni serverless definite dall'utente che eseguono codice sul core. Le funzioni Lambda sono create AWS Lambda e referenziate da un gruppo Greengrass. Per ulteriori informazioni, consulta [Esegui funzioni Lambda locali](#).
- Connettori. Applicazioni serverless predefinite che eseguono codice sul core. I connettori possono fornire un'integrazione integrata con l'infrastruttura locale, i protocolli dei dispositivi e altri servizi cloud. AWS Per ulteriori informazioni, consulta [Integrazione con servizi e protocolli tramite i connettori](#).
- Abbonamenti. Definiscono gli autori, i sottoscrittori e gli argomenti MQTT (o gli argomenti) autorizzati per la comunicazione MQTT.
- Risorse. Riferimenti a [dispositivi e volumi](#) locali, [modelli di apprendimento automatico](#) e [segreti](#), utilizzati per il controllo degli accessi tramite funzioni e connettori Greengrass Lambda.
- Registri. Configurazioni di registrazione per i componenti di AWS IoT Greengrass sistema e le funzioni Lambda. Per ulteriori informazioni, consulta [the section called “Monitoraggio con i log AWS IoT Greengrass”](#).

Gestisci il tuo gruppo Greengrass in Cloud AWS e poi lo distribuisce su un core. La distribuzione copia la configurazione del gruppo nel file `group.json` sul dispositivo core. Questo file si trova in `greengrass-root/ggc/deployments/group`.

**Note**

Durante una distribuzione, il processo daemon Greengrass sul dispositivo core si arresta e quindi si riavvia.

Distribuzione dei gruppi dalla console AWS IoT

È possibile distribuire un gruppo e gestirne le distribuzioni dalla pagina di configurazione del gruppo nella console. AWS IoT

Note

Per aprire questa pagina nella console, scegli Dispositivi Greengrass, quindi Gruppi (V1), quindi in Gruppi Greengrass, scegli il tuo gruppo.

Per distribuire la versione corrente del gruppo

- Dalla pagina di configurazione del gruppo, scegli Distribuisci.

Per visualizzare la cronologia della distribuzione del gruppo

La cronologia della distribuzione di un gruppo include la data e l'ora, la versione del gruppo e lo stato di ogni tentativo di distribuzione.

1. Dalla pagina di configurazione del gruppo, scegli la scheda Distribuzioni.
2. Per visualizzare ulteriori informazioni su una distribuzione, inclusi i messaggi di errore, scegli Distribuzioni dalla AWS IoT console, in Dispositivi Greengrass.

Per ripetere la distribuzione di un gruppo

Potresti voler ripetere una distribuzione se la distribuzione corrente ha esito negativo o ripristinare una versione di gruppo diversa.

1. Dalla AWS IoT console, scegli Dispositivi Greengrass, quindi scegli Gruppi (V1).
2. Seleziona la scheda Distribuzioni.
3. Scegli la distribuzione che desideri ridistribuire e scegli Redeploy.

Per reimpostare le distribuzioni di gruppo

È possibile reimpostare le distribuzioni di gruppo per spostare o eliminare un gruppo o rimuovere le informazioni sulla distribuzione. Per ulteriori informazioni, consulta [the section called "Reimpostazione delle distribuzioni"](#).

1. Dalla AWS IoT console, scegli Dispositivi Greengrass, quindi scegli Gruppi (V1).
2. Seleziona la scheda Distribuzioni.
3. Scegli la distribuzione che desideri ripristinare e scegli Ripristina distribuzioni.

Distribuzione di gruppi con l'API AWS IoT Greengrass

L' AWS IoT Greengrass API fornisce le seguenti azioni per distribuire AWS IoT Greengrass gruppi e gestire le distribuzioni di gruppo. Puoi richiamare queste azioni dall' AWS IoT Greengrass API o dall' AWS CLI SDK. AWS

Azione	Descrizione
CreateDeployment	Creare una distribuzione NewDeployment o Redeployment . Potresti voler ridistribuire una distribuzione se la distribuzione corrente ha esito negativo. In alternativa, potresti voler ridistribuire per ripristinare una versione di gruppo diversa.
GetDeploymentStatus	Restituisce lo stato di una distribuzione: Building, InProgress , Success o Failure. Puoi configurare Amazon EventBridge Events per ricevere notifiche di distribuzione. Per ulteriori informazioni, consulta the section called “Ottenere le notifiche di distribuzione”.
ListDeployments	Restituisce la cronologia della distribuzione per il gruppo.
ResetDeployments	Reimposta le distribuzioni per il gruppo. È possibile reimpostare le distribuzioni di gruppo per spostare o eliminare un gruppo o rimuovere le informazioni sulla distribuzione. Per ulteriori informazioni, consulta the section called “Reimpostazione delle distribuzioni”.

 Note

Per informazioni sulle operazioni di distribuzione in blocco, consulta [the section called “Creazione di distribuzioni in blocco”](#).

Ottenere l'ID del gruppo

L'ID del gruppo viene comunemente utilizzato nelle operazioni API. Puoi utilizzare l'[ListGroup](#)azione per trovare l'ID del gruppo target dal tuo elenco di gruppi. Ad esempio, in AWS CLI, utilizzare il `list-groups` comando.

```
aws greengrass list-groups
```

È inoltre possibile includere l'opzione `query` per filtrare i risultati. Per esempio:

- Per ottenere il gruppo creato più di recente:

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

- Per ottenere un gruppo in base al nome:

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

I nomi dei gruppi non devono essere univoci, pertanto potrebbero essere restituiti più gruppi.

Di seguito è riportata una risposta `list-groups` di esempio: Le informazioni relative a ciascun gruppo includono l'ID del gruppo (nella proprietà `Id`) e l'ID della versione più recente del gruppo (nella proprietà `LatestVersion`). Per ottenere un'altra versione IDs per un gruppo, usa l'ID del gruppo con [ListGroupVersions](#).

Note

Puoi trovare questi valori anche nella AWS IoT console. L'ID gruppo viene visualizzato nella pagina Settings (Impostazioni) del gruppo. IDs Le versioni del gruppo vengono visualizzate nella scheda Distribuzioni del gruppo.

```
{
  "Groups": [
    {
      "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE/versions/4cbc3f07-fc5e-48c4-
a50e-7d356EXAMPLE",
```

```

    "Name": "MyFirstGroup",
    "LastUpdatedTimestamp": "2019-11-11T05:47:31.435Z",
    "LatestVersion": "4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
    "CreationTimestamp": "2019-11-11T05:47:31.435Z",
    "Id": "00dedaaa-ac16-484d-ad77-c3eedEXAMPLE",
    "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-
ac16-484d-ad77-c3eedEXAMPLE"
  },
  {
    "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE/versions/8fe9e8ec-64d1-4647-
b0b0-01dc8EXAMPLE",
    "Name": "GreenhouseSensors",
    "LastUpdatedTimestamp": "2020-01-07T19:58:36.774Z",
    "LatestVersion": "8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
    "CreationTimestamp": "2020-01-07T19:58:36.774Z",
    "Id": "036ceaf9-9319-4716-ba2a-237f9EXAMPLE",
    "Arn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE"
  },
  ...
]
}

```

Se non specifichi un Regione AWS, AWS CLI i comandi utilizzano la regione predefinita del tuo profilo. Per restituire gruppi in una regione diversa, includi l'*region* opzione. Per esempio:

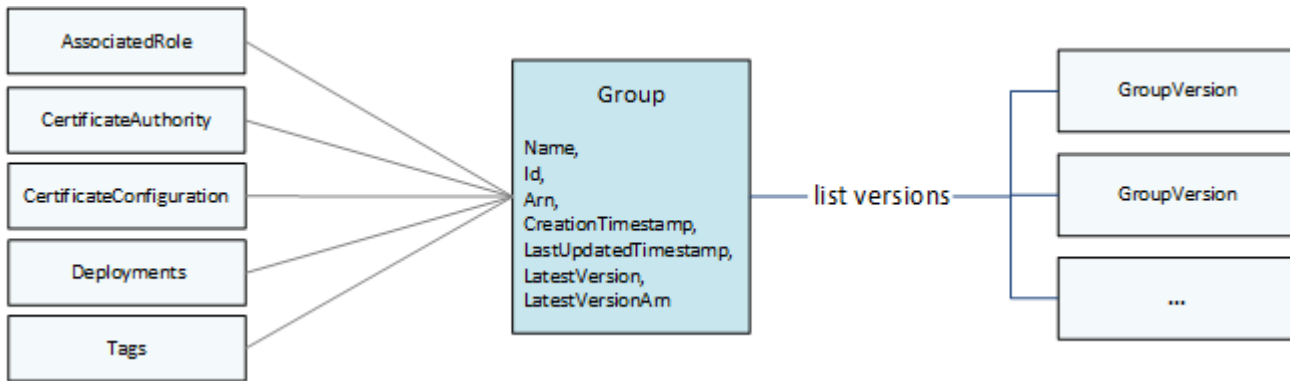
```
aws greengrass list-groups --region us-east-1
```

Panoramica del modello a oggetti di AWS IoT Greengrass gruppo

Quando si programma con l' AWS IoT Greengrass API, è utile comprendere il modello a oggetti del gruppo Greengrass.

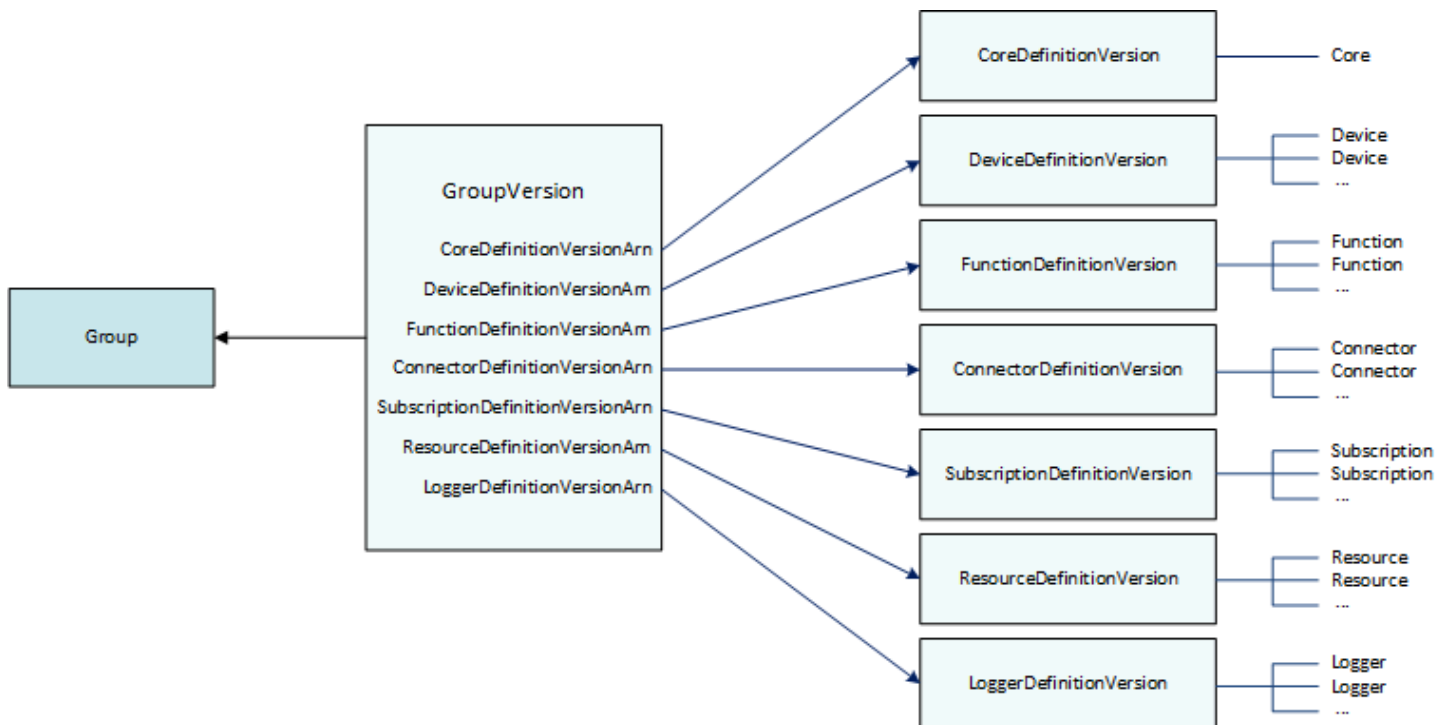
Gruppi

Nell' AWS IoT Greengrass API, l'Group oggetto di primo livello è costituito da metadati e da un elenco di oggetti. GroupVersion GroupVersion gli oggetti sono associati a un Group by ID.



Versioni del gruppo

Gli oggetti **GroupVersion** definiscono l'appartenenza al gruppo. Ogni **GroupVersion** fa riferimento a una **CoreDefinitionVersion** e ad altre versioni dei componenti per ARN. Questi riferimenti determinano quali entità includere nel gruppo.



Ad esempio, per includere tre funzioni Lambda, un dispositivo e due abbonamenti nel gruppo, i riferimenti: **GroupVersion**

- Il **CoreDefinitionVersion** che contiene il core richiesto.
- Il **FunctionDefinitionVersion** che contiene le tre funzioni.
- Il **DeviceDefinitionVersion** che contiene il dispositivo client.

- Il `SubscriptionDefinitionVersion` che contiene le due sottoscrizioni.

Il `GroupVersion` distribuito su un dispositivo core determina le entità disponibili nell'ambiente locale e il modo in cui possono interagire.

Componenti del gruppo

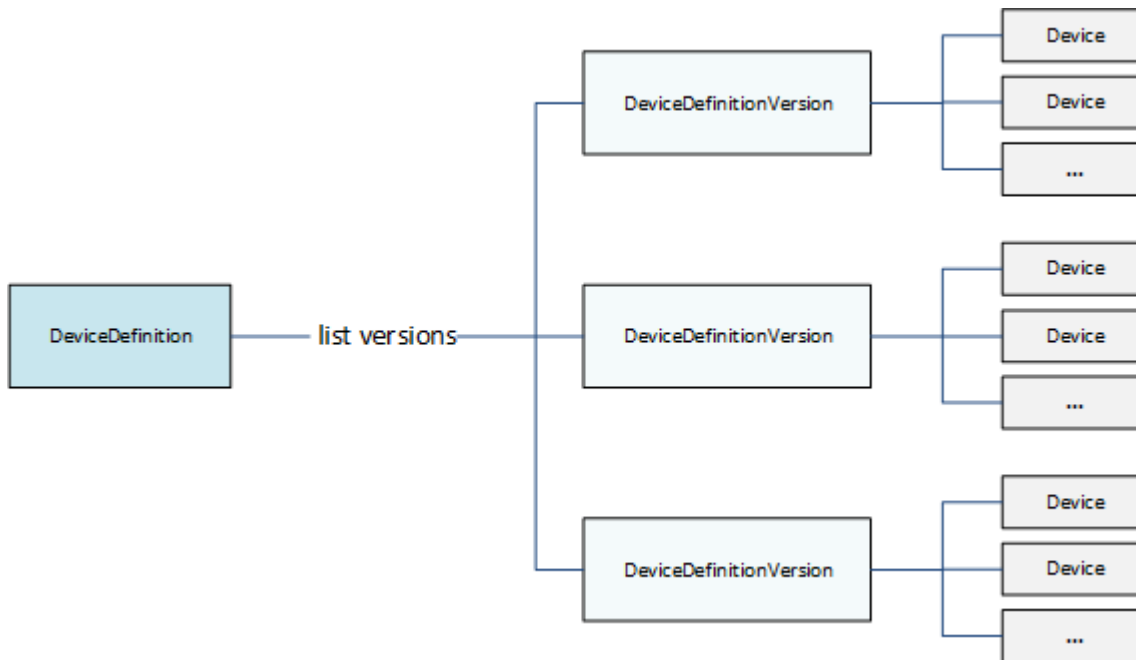
I componenti aggiunti ai gruppi hanno una gerarchia di tre livelli:

- Una definizione che fa riferimento a un elenco di `DefinitionVersion`oggetti di un determinato tipo. Ad esempio, un `DeviceDefinition` fa riferimento a un elenco di oggetti `DeviceDefinitionVersion`.
- Una `DefinitionVersion`che contiene un insieme di entità di un determinato tipo. Ad esempio, un `DeviceDefinitionVersion` contiene un elenco di oggetti `Device`.
- Singole entità che definiscono le proprietà e il comportamento. Ad esempio, a `Device` definisce l'ARN del dispositivo client corrispondente nel AWS IoT registro, l'ARN del certificato del dispositivo e se la sua shadow locale si sincronizza automaticamente con il cloud.

Puoi aggiungere i seguenti tipi di entità a un gruppo:

- [Connector](#)
- [Core](#)
- [Device](#)
- [Funzione](#)
- [Logger](#)
- [Risorsa](#)
- [Subscription](#)

L'esempio seguente `DeviceDefinition` fa riferimento a tre oggetti `DeviceDefinitionVersion` che contengono ciascuno più oggetti `Device`. In un gruppo viene utilizzato solo un oggetto `DeviceDefinitionVersion` alla volta.



Aggiornamento dei gruppi

Nell' AWS IoT Greengrass API, si utilizzano le versioni per aggiornare la configurazione di un gruppo. Le versioni sono immutabili, quindi per aggiungere, rimuovere o modificare i componenti del gruppo, è necessario creare `DefinitionVersion` oggetti che contengano entità nuove o aggiornate.

È possibile associare nuovi `DefinitionVersion` oggetti a oggetti `Definition` nuovi o esistenti. Ad esempio, puoi utilizzare l'operazione `CreateFunctionDefinition` per creare un `FunctionDefinition` che include `FunctionDefinitionVersion` come versione iniziale oppure puoi utilizzare l'operazione `CreateFunctionDefinitionVersion` e fare riferimento a un `FunctionDefinition` esistente.

Dopo aver creato i componenti del gruppo, ne crei uno `GroupVersion` contenente tutti `DefinitionVersion` gli oggetti che desiderate includere nel gruppo. Puoi quindi distribuire `GroupVersion`.

Per distribuire un `GroupVersion`, deve fare riferimento a un `CoreDefinitionVersion` che contiene esattamente uno `Core`. Tutte le entità a cui si fa riferimento devono essere membri del gruppo. Inoltre, un [ruolo di servizio Greengrass](#) deve essere associato all'utente Account AWS nel luogo in Regione AWS cui si sta implementando il `GroupVersion`.

 Note

Le operazioni Update nell'API vengono utilizzate per modificare il nome di un Group o di un oggetto Definition del componente.

Aggiornamento di entità che fanno riferimento a risorse AWS

Le funzioni e le [risorse segrete di Greengrass Lambda definiscono le proprietà specifiche di Greengrass e fanno anche riferimento alle risorse](#) corrispondenti. AWS Per aggiornare queste entità, è possibile apportare modifiche alla AWS risorsa corrispondente anziché agli oggetti Greengrass. Ad esempio, le funzioni Lambda fanno riferimento a una funzione in AWS Lambda e definiscono anche il ciclo di vita e altre proprietà specifiche del gruppo Greengrass.

- Per aggiornare il codice della funzione Lambda o le dipendenze pacchettizzate, apporta le modifiche. AWS Lambda Durante la successiva distribuzione di gruppo, queste modifiche vengono recuperate AWS Lambda e copiate nell'ambiente locale.
- Per aggiornare [le proprietà specifiche di Greengrass](#), devi creare un FunctionDefinitionVersion che contiene le proprietà Function aggiornate.

 Note

Le funzioni Lambda di Greengrass possono fare riferimento a una funzione Lambda tramite alias ARN o versione ARN. Se si fa riferimento all'ARN alias (consigliato), non è necessario aggiornare FunctionDefinitionVersion (o SubscriptionDefinitionVersion) quando si pubblica una nuova versione della funzione in AWS Lambda. Per ulteriori informazioni, consulta [the section called “Riferimento alle funzioni in base all'alias o alla versione”](#).

Consulta anche

- [the section called “Ottenere le notifiche di distribuzione”](#)
- [the section called “Reimpostazione delle distribuzioni”](#)
- [the section called “Creazione di distribuzioni in blocco”](#)
- [Risoluzione dei problemi di distribuzione](#)

- [AWS IoT Greengrass Version 1 Documentazione di riferimento API](#)
- [AWS IoT Greengrass comandi](#) nel AWS CLI Command Reference

Ottenere le notifiche di distribuzione

Le regole EventBridge degli eventi di Amazon ti forniscono notifiche sui cambiamenti di stato per le distribuzioni del tuo gruppo Greengrass. EventBridge fornisce un flusso quasi in tempo reale di eventi di sistema che descrive i cambiamenti nelle risorse. AWS IoT Greengrass invia questi eventi EventBridge a almeno una volta. Ciò significa che AWS IoT Greengrass potrebbe inviare più copie di un determinato evento per garantire la consegna. Inoltre, i listener di eventi potrebbero non ricevere gli eventi nell'ordine in cui si sono verificati.

Note

Amazon EventBridge è un servizio di bus di eventi che puoi utilizzare per connettere le tue applicazioni con dati provenienti da una varietà di fonti, come [i dispositivi principali Greengrass](#) e le notifiche di distribuzione. Per ulteriori informazioni, consulta [What is Amazon EventBridge?](#) nella Amazon EventBridge User Guide.

AWS IoT Greengrass emette un evento quando le distribuzioni di gruppo cambiano stato. È possibile creare una EventBridge regola che venga eseguita per tutte le transizioni di stato o per le transizioni verso gli stati specificati. Quando una distribuzione entra in uno stato che avvia una regola, EventBridge richiama le azioni target definite nella regola. In questo modo è possibile inviare notifiche, acquisire informazioni sugli eventi, intraprendere azioni correttive o avviare altri eventi in risposta a una modifica dello stato. Ad esempio, è possibile creare regole per i seguenti casi d'uso:

- Avvia le operazioni successive all'implementazione, come il download delle risorse e la notifica al personale.
- Inviare notifiche in caso di distribuzione riuscita o non riuscita.
- Pubblicare parametri personalizzati sugli eventi di distribuzione.

AWS IoT Greengrass emette un evento quando una distribuzione entra nei seguenti stati: `Building`, `InProgress` e `Success Failure`

 Note

Il monitoraggio dello stato di un'operazione di [distribuzione di massa](#) non è attualmente supportato. Tuttavia, AWS IoT Greengrass emette eventi di modifica dello stato per le implementazioni di singoli gruppi che fanno parte di una distribuzione di massa.

Evento di modifica dello stato della distribuzione del gruppo

L'[evento](#) per una modifica dello stato della distribuzione utilizza il formato seguente:

```
{
  "version": "0",
  "id": " cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
  "detail-type": "Greengrass Deployment Status Change",
  "source": "aws.greengrass",
  "account": "123456789012",
  "time": "2018-03-22T00:38:11Z",
  "region": "us-west-2",
  "resources": [],
  "detail": {
    "group-id": "284dcd4e-24bc-4c8c-a770-EXAMPLEf03b8",
    "deployment-id": "4f38f1a7-3dd0-42a1-af48-EXAMPLE09681",
    "deployment-type": "NewDeployment|Redeployment|ResetDeployment|
ForceResetDeployment",
    "status": "Building|InProgress|Success|Failure"
  }
}
```

È possibile creare regole che si applicano a uno o più gruppi. È possibile filtrare le regole in base a uno o più dei seguenti tipi di distribuzione e stati di distribuzione:

Tipi di distribuzione

- **NewDeployment.** La prima implementazione di una versione di gruppo.
- **ReDeployment.** Una ridistribuzione di una versione di gruppo.
- **ResetDeployment.** Elimina le informazioni di distribuzione archiviate Cloud AWS internamente e nel AWS IoT Greengrass core. Per ulteriori informazioni, consulta [the section called “Reimpostazione delle distribuzioni”](#).

- **ForceResetDeployment.** Elimina le informazioni di distribuzione memorizzate nel file Cloud AWS e ne segnala l'esito positivo senza attendere che il core risponda. Inoltre, elimina le informazioni di distribuzione archiviate sul core se il core è connesso o quando si connette in seguito.

Stati della distribuzione

- **Building.** AWS IoT Greengrass sta convalidando la configurazione del gruppo e sta creando elementi di distribuzione.
- **InProgress.** La distribuzione è in corso sul AWS IoT Greengrass core.
- **Success.** L'implementazione ha avuto successo.
- **Failure.** La distribuzione non è riuscita.

È possibile che gli eventi vengano duplicati o non funzionino. Per determinare l'ordine degli eventi, utilizza la proprietà `time`.

Note

AWS IoT Greengrass non utilizza la `resources` proprietà, quindi è sempre vuota.

Prerequisiti per la creazione di regole EventBridge

Prima di creare una EventBridge regola per AWS IoT Greengrass, procedi come segue:

- Acquisisci familiarità con eventi, regole e obiettivi in EventBridge
- Crea e configura gli obiettivi richiamati dalle tue regole. EventBridge Le regole possono richiamare molti tipi di target, tra cui:
 - Servizio di notifica semplice Amazon (Amazon Simple Notification Service (Amazon SNS))
 - AWS Lambda funzioni
 - Amazon Kinesis Video Streams
 - Code di Amazon Simple Queue Service (Amazon SQS)

Per ulteriori informazioni, consulta [What is Amazon EventBridge?](#) e [Guida introduttiva ad Amazon EventBridge](#) nella Amazon EventBridge User Guide.

Configurazione delle notifiche di distribuzione (console)

Utilizza i seguenti passaggi per creare una EventBridge regola che pubblichi un argomento di Amazon SNS quando lo stato di distribuzione cambia per un gruppo. Ciò consente a server Web, indirizzi e-mail e altri sottoscrittori di argomenti di rispondere all'evento. Per ulteriori informazioni, consulta [Creazione di una EventBridge regola che si attiva su un evento da una AWS risorsa](#) nella Amazon EventBridge User Guide.

1. Apri la [EventBridgeconsole Amazon](#).
2. Nel pannello di navigazione, scegli Regole.
3. Scegli Create rule (Crea regola).
4. Inserire un nome e una descrizione per la regola.

Una regola non può avere lo stesso nome di un'altra regola nella stessa regione e sullo stesso router di eventi.

5. Per Select event bus (Seleziona bus di eventi), scegli il bus di eventi che desideri associare a questa regola. Se la regola deve cercare eventi corrispondenti provenienti dal tuo account, seleziona Bus di eventi predefiniti di AWS . Quando un AWS servizio del tuo account emette un evento, questo passa sempre al bus eventi predefinito del tuo account.
6. Per Rule type (Tipo di regola), scegli Rule with an event pattern (Regola con un modello di eventi).
7. Scegli Next (Successivo).
8. Per Origine evento, scegli Servizi AWS .
9. Per Event pattern, scegli AWS servizi.
10. Per l'AWS assistenza, scegli Greengrass.
11. Per Tipo evento, scegliere Modifica stato distribuzione Greengrass.

Note

La chiamata AWS API tramite il tipo di CloudTrail evento si basa sull' AWS IoT Greengrass integrazione con AWS CloudTrail. È possibile utilizzare questa opzione per creare regole avviate da chiamate di lettura o scrittura all' AWS IoT Greengrass API. Per ulteriori informazioni, consulta [the section called “Registrazione delle chiamate AWS IoT Greengrass API con AWS CloudTrail”](#).

12. Scegli gli stati di distribuzione che avviano una notifica.

- Per ricevere notifiche per tutti gli eventi di modifica dello stato, scegliere Qualsiasi stato.
 - Per ricevere notifiche solo per alcuni eventi di modifica dello stato, scegliere Stato specifico, e quindi scegliere gli stati di destinazione.
13. Scegli i tipi di distribuzione che avviano una notifica.
- Per ricevere notifiche per tutti i tipi di distribuzione, scegliere Qualsiasi stato.
 - Per ricevere notifiche solo per alcuni tipi di distribuzione, scegliere Stato specifico e quindi scegliere i tipi di distribuzione di destinazione.
14. Scegli Next (Successivo).
15. Per Target types (Tipi di destinazione), scegli AWS service (Servizio).
16. Per Seleziona una destinazione, configura la tua destinazione. Questo esempio utilizza un argomento Amazon SNS, ma puoi configurare altri tipi di destinazione per inviare notifiche.
- a. In Target (Destinazione), scegli SNS topic (Argomento SNS).
 - b. Per Argomento, scegli l'argomento di destinazione.
 - c. Scegli Next (Successivo).
17. In Tag, definisci i tag per la regola o lascia i campi vuoti.
18. Scegli Next (Successivo).
19. Rivedi i dettagli della regola e scegli Create rule (Crea regola).

Configurazione delle notifiche di distribuzione (CLI)

Utilizza i seguenti passaggi per creare una EventBridge regola che pubblichi un argomento di Amazon SNS quando lo stato di distribuzione cambia per un gruppo. Ciò consente a server Web, indirizzi e-mail e altri sottoscrittori di argomenti di rispondere all'evento.

1. Crea la regola.
 - *group-id* Sostituiscilo con l'ID del tuo AWS IoT Greengrass gruppo.

```
aws events put-rule \  
  --name TestRule \  
  --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"group-id\":  
    [\"group-id\"]}}\"
```

Le proprietà omesse dal modello vengono ignorate.

2. Aggiungi l'argomento come destinazione della regola.

- Sostituisci *topic-arn* con l'ARN del tuo argomento Amazon SNS.

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

Note

Per consentire EventBridge ad Amazon di indicare il tuo argomento di riferimento, devi aggiungere una politica basata sulle risorse all'argomento. Per ulteriori informazioni, consulta le [autorizzazioni di Amazon SNS](#) nella Amazon EventBridge User Guide.

Per ulteriori informazioni, consulta la sezione [Eventi e modelli di eventi EventBridge nella](#) Amazon EventBridge User Guide.

Configurazione delle notifiche di distribuzione (AWS CloudFormation)

Utilizza AWS CloudFormation i modelli per creare EventBridge regole che inviano notifiche sui cambiamenti di stato per le implementazioni del tuo gruppo Greengrass. Per ulteriori informazioni, consulta il [riferimento ai tipi di EventBridge risorse Amazon](#) nella Guida AWS CloudFormation per l'utente.

Consulta anche

- [Distribuisci gruppi AWS IoT Greengrass](#)
- [Che cos'è Amazon EventBridge?](#) nella Amazon EventBridge User Guide

Reimpostazione delle distribuzioni

Questa funzionalità è disponibile per AWS IoT Greengrass Core v1.1 e versioni successive.

È possibile reimpostare le distribuzioni di un gruppo su:

- Elimina il gruppo, ad esempio quando desideri spostare il nucleo del gruppo in un altro gruppo o se il nucleo del gruppo è stato ridisegnato. Prima di eliminare un gruppo, è necessario reimpostare le distribuzioni del gruppo per utilizzare il core con un altro gruppo Greengrass.
- Spostare il core del gruppo a un gruppo diverso.
- Ripristinare il gruppo allo stato precedente a eventuali distribuzioni.
- Rimuovere la configurazione di distribuzione dal dispositivo core.
- Eliminare dati sensibili dal dispositivo core o dal cloud.
- Distribuire un nuovo gruppo di configurazione a un core senza dover sostituire il core con un altro nel gruppo corrente.

Note

La funzionalità di reimpostazione delle distribuzioni non è disponibile in AWS IoT Greengrass Core Software v1.0.0. Non è possibile eliminare un gruppo distribuito utilizzando v1.0.0.

L'operazione di ripristino delle distribuzioni elimina prima tutte le informazioni di distribuzione archiviate nel cloud per un determinato gruppo. Quindi ordina al dispositivo principale del gruppo di ripulire anche tutte le informazioni relative alla distribuzione (funzioni Lambda, registri utente, database shadow e certificato del server, ma non i certificati core `config.json` definiti dall'utente o Greengrass). Non è possibile reimpostare le distribuzioni per un gruppo se il gruppo dispone di una distribuzione con stato `In Progress` o `Building`.

Reimposta le distribuzioni dalla console AWS IoT

È possibile ripristinare le distribuzioni di gruppo dalla pagina di configurazione del gruppo nella console. AWS IoT

1. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).
2. Scegliere il gruppo target.
3. Dalla scheda Distribuzioni, scegli Reimposta distribuzioni.
4. Nella finestra di dialogo Reimposta le distribuzioni per questo Greengrass Group, digita **confirm** per concordare e scegli Reimposta distribuzione.

Reimposta le distribuzioni con l'API AWS IoT Greengrass

Puoi utilizzare l'ResetDeploymentsazione nell' AWS IoT Greengrass API o nell' AWS CLI AWS SDK per reimpostare le distribuzioni. Gli esempi in questo argomento utilizzano l'interfaccia a riga di comando.

```
aws greengrass reset-deployments --group-id GroupId [--force]
```

Argomenti del comando CLI **reset-deployments**:

--group-id

L'ID del gruppo. Utilizzare il comando `list-groups` per ottenere questo valore.

--force

Facoltativo. Utilizzare questo parametro se il dispositivo core del gruppo è stato smarrito, rubato o distrutto. Questa opzione determina la procedura di reimpostazione distribuzioni per segnalare la riuscita una volta che tutte le informazioni di distribuzione nel cloud sono state ripulite, senza attendere che un dispositivo core risponda. Tuttavia, se il dispositivo core è o diventa attivo, esegue anche operazioni di pulizia.

L'output del comando dell'interfaccia a riga di comando `reset-deployments` è simile al seguente:

```
{
  "DeploymentId": "4db95ef8-9309-4774-95a4-eea580b6ceef",
  "DeploymentArn": "arn:aws:greengrass:us-west-2:106511594199:/greengrass/groups/
b744ed45-a7df-4227-860a-8d4492caa412/deployments/4db95ef8-9309-4774-95a4-eea580b6ceef"
}
```

È possibile controllare lo stato della reimpostazione distribuzioni con il comando CLI `get-deployment-status`:

```
aws greengrass get-deployment-status --deployment-id DeploymentId --group-id GroupId
```

Argomenti del comando CLI **get-deployment-status**:

--deployment-id

L'ID della distribuzione.

--group-id

L'ID del gruppo.

L'output del comando dell'interfaccia a riga di comando `get-deployment-status` è simile al seguente:

```
{
  "DeploymentStatus": "Success",
  "UpdatedAt": "2017-04-04T00:00:00.000Z"
}
```

`DeploymentStatus` è impostato su `Building` quando la reimpostazione distribuzioni è in fase di preparazione. Quando la distribuzione di ripristino è pronta ma il AWS IoT Greengrass core non ha ripreso la distribuzione ripristinata, lo è. `DeploymentStatus InProgress`

Se l'operazione di ripristino non riesce, le informazioni di errore vengono restituite nella risposta.

Consulta anche

- [Distribuisci gruppi AWS IoT Greengrass](#)
- [ResetDeployments](#) nell'AWS IoT Greengrass Version 1 API Reference
- [GetDeploymentStatus](#) nell'AWS IoT Greengrass Version 1 API Reference

Creazione di distribuzioni in blocco per i gruppi

Puoi utilizzare semplici chiamate API per distribuire un numero elevato di gruppi Greengrass contemporaneamente. Queste distribuzioni vengono attivate a una velocità adattiva con un limite superiore fisso.

Questo tutorial descrive come utilizzare il per AWS CLI creare e monitorare una distribuzione di gruppo in blocco in. AWS IoT Greengrass L'esempio di distribuzione di massa di questo tutorial include più gruppi. Puoi utilizzare l'esempio nella tua implementazione per aggiungere il numero di gruppi desiderato.

Il tutorial include le seguenti fasi di alto livello:

1. [Creazione e caricamento del file di input della distribuzione di massa](#)

2. [Crea e configura un ruolo di esecuzione IAM per distribuzioni di massa](#)
3. [Autorizzazione per l'accesso al bucket S3 da parte del ruolo di esecuzione](#)
4. [Distribuzione dei gruppi](#)
5. [Test della distribuzione](#)

Prerequisiti

Per completare questo tutorial, è necessario quanto segue:

- Uno o più gruppi Greengrass distribuibili. Per ulteriori informazioni sulla creazione di core e gruppi AWS IoT Greengrass, consulta [Guida introduttiva con AWS IoT Greengrass](#).
- AWS CLI Installato e configurato sulla tua macchina. Per ulteriori informazioni, consulta la [Guida per l'utente di AWS CLI](#).
- Un bucket S3 creato nello stesso modo Regione AWS di. AWS IoT Greengrass Per informazioni, consulta [Creazione e configurazione di un bucket S3](#) nella Guida per l'utente di Amazon Simple Storage Service.

Note

Al momento, i bucket abilitati per SSE KMS non sono supportati.

Fase 1: creazione e caricamento del file di input della distribuzione di massa

In questa fase, crei un file di input per la distribuzione e lo carichi nel tuo bucket Amazon S3. Questo file è un file JSON serializzato e delimitato da righe che contiene informazioni su ciascun gruppo della distribuzione di massa. AWS IoT Greengrass utilizza queste informazioni per distribuire ciascun gruppo per tuo conto quando inizi la distribuzione di gruppo in blocco.

1. Eseguire il seguente comando per ottenere il `groupId` per ciascun gruppo da distribuire. Immettere `groupId` nel file di input della distribuzione di massa in modo che AWS IoT Greengrass possa identificare ciascun gruppo da distribuire.

 Note

Puoi trovare questi valori anche nella console. AWS IoT L'ID gruppo viene visualizzato nella pagina Settings (Impostazioni) del gruppo. IDs Le versioni del gruppo vengono visualizzate nella scheda Distribuzioni del gruppo.

```
aws greengrass list-groups
```

La risposta contiene informazioni su ogni gruppo del tuo AWS IoT Greengrass account:

```
{
  "Groups": [
    {
      "Name": "string",
      "Id": "string",
      "Arn": "string",
      "LastUpdatedTimestamp": "string",
      "CreationTimestamp": "string",
      "LatestVersion": "string",
      "LatestVersionArn": "string"
    }
  ],
  "NextToken": "string"
}
```

Eseguire il seguente comando per ottenere il `groupVersionId` di ciascun gruppo da distribuire.

```
list-group-versions --group-id groupId
```

La risposta contiene informazioni su tutte le versioni del gruppo. Prendi nota del `Version` valore della versione di gruppo che desideri utilizzare.

```
{
  "Versions": [
    {
      "Arn": "string",
      "Id": "string",
      "Version": "string",
      "CreationTimestamp": "string"
    }
  ],
  "NextToken": "string"
}
```

2. Nel terminale o nell'editor che preferisci, crea un file *MyBulkDeploymentInputFile*, usando l'esempio seguente. Questo file contiene informazioni su ciascun AWS IoT Greengrass gruppo da includere in una distribuzione di massa. Benché questo esempio definisca più gruppi, per questo tutorial il tuo file potrà contenerne solo uno.

Note

Le dimensioni di questo file devono essere inferiori a 100 MB.

```
{"GroupId": "groupId1", "GroupVersionId": "groupVersionId1",
  "DeploymentType": "NewDeployment"}
{"GroupId": "groupId2", "GroupVersionId": "groupVersionId2",
  "DeploymentType": "NewDeployment"}
{"GroupId": "groupId3", "GroupVersionId": "groupVersionId3",
  "DeploymentType": "NewDeployment"}
...
```

Ciascun record (o riga) contiene un oggetto gruppo. Ogni oggetto gruppo contiene i corrispondenti GroupId e GroupVersionId e un DeploymentType. Attualmente AWS IoT Greengrass supporta solo i tipi di distribuzione in NewDeployment blocco.

Salvare e chiudere il file. Prendere nota del percorso del file.

3. Usa il seguente comando nel tuo terminale per caricare il file di input nel tuo bucket Amazon S3. Specificare il percorso e il nome del file. Per informazioni, consulta [Aggiunta di un oggetto a un bucket](#).

```
aws s3 cp path/MyBulkDeploymentInputFile s3://amzn-s3-demo-bucket/
```

Fase 2: creazione e configurazione di un ruolo di esecuzione IAM

In questo passaggio, utilizzi la console IAM per creare un ruolo di esecuzione autonomo. Quindi stabilisci una relazione di fiducia tra il ruolo AWS IoT Greengrass e ti assicuri che il tuo utente IAM disponga PassRole dei privilegi per il ruolo di esecuzione. Ciò consente di AWS IoT Greengrass assumere il ruolo di esecuzione e creare le distribuzioni per conto dell'utente.

1. Usa la seguente policy per creare un ruolo di esecuzione. Questo documento della policy consente a AWS IoT Greengrass di accedere al file di input della distribuzione di massa al momento della creazione automatica di ciascuna distribuzione.

Per ulteriori informazioni sulla creazione di ruolo IAM e la delega delle autorizzazioni, consulta [Creazione di ruoli IAM](#).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": "greengrass:CreateDeployment",
      "Resource": [
        "arn:aws:greengrass:region:accountId:/greengrass/groups/groupId1",
        "arn:aws:greengrass:region:accountId:/greengrass/groups/groupId2",
        "arn:aws:greengrass:region:accountId:/greengrass/groups/groupId3",
        ...
      ]
    }
  ]
}
```

 Note

Questa policy deve disporre di una risorsa per ciascun gruppo o versione del gruppo nel file di input della distribuzione di massa da distribuire da parte di AWS IoT Greengrass. Per consentire l'accesso a tutti i gruppi di risorse, per Resource, specificare un asterisco:

```
"Resource": ["*"]
```

2. Modificare la relazione di trust per il ruolo di esecuzione in modo da includere AWS IoT Greengrass. Ciò consente a AWS IoT Greengrass di utilizzare il tuo ruolo di esecuzione e le relative autorizzazioni collegate. Per informazioni, consulta [Modifica della relazione di trust per un ruolo esistente](#).

Ti consigliamo di includere anche le chiavi del contesto `aws:SourceArn` e della condizione `aws:SourceAccount` globale nella tua politica di fiducia per evitare il confuso problema della vice sicurezza. Le chiavi di contesto delle condizioni limitano l'accesso per consentire solo le richieste che provengono dall'account specificato e dall'area di lavoro Greengrass. Per ulteriori informazioni sul problema del "confused deputy", consulta [Prevenzione del problema "confused deputy" tra servizi](#).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "greengrass.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "account-id"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
        }
      }
    }
  ]
}
```



```

    }
  }
]
}

```

3. Concedi `PassRole` le autorizzazioni IAM per il tuo ruolo di esecuzione al tuo utente IAM. Questo utente IAM è quello utilizzato per avviare la distribuzione in blocco. `PassRole` le autorizzazioni consentono all'utente IAM di passare il ruolo di esecuzione all'uso. AWS IoT Greengrass Per ulteriori informazioni, consulta [Concedere a un utente le autorizzazioni per passare un ruolo a un servizio](#). AWS

Utilizza l'esempio seguente per aggiornare la policy IAM associata al tuo ruolo di esecuzione. Se necessario, è possibile modificarlo.

```

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

```

Fase 3: autorizzazione per l'accesso al bucket S3 da parte del ruolo di esecuzione

Per avviare la distribuzione in blocco, il ruolo di esecuzione deve essere in grado di leggere il file di input della distribuzione di massa dal bucket Amazon S3. Allega la seguente policy di esempio al tuo bucket Amazon S3 in modo che le sue `GetObject` autorizzazioni siano accessibili al tuo ruolo di esecuzione.

Per ulteriori informazioni, consulta [In che modo aggiungere una policy del bucket S3?](#)

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

Utilizzare il comando seguente nel terminale per verificare la policy del bucket.

```
aws s3api get-bucket-policy --bucket amzn-s3-demo-bucket
```

Note

Puoi modificare direttamente il tuo ruolo di esecuzione per concedergli invece l'autorizzazione alle autorizzazioni del `GetObject` tuo bucket Amazon S3. A tale scopo, collegare la seguente policy di esempio al ruolo di esecuzione.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/objectKey"
    }
  ]
}
```

Fase 4: distribuzione dei gruppi

In questo fase, inizierai un'operazione di distribuzione di massa di tutte le versioni dei gruppi configurate nel file di input della distribuzione di massa. L'operazione di distribuzione per tutte le versioni di gruppo è di tipo `NewDeploymentType`.

Note

Non potrai utilizzare `StartBulkDeployment` mentre è in esecuzione un'altra distribuzione di massa dallo stesso account. La richiesta viene respinta.

1. Per avviare la distribuzione di massa, utilizzare il comando seguente:

Ti consigliamo di includere un token `X-Amzn-Client-Token` in ciascuna richiesta `StartBulkDeployment`. Tali richieste sono idempotenti rispetto al token e ai parametri della richiesta. Il token può essere qualsiasi stringa univoca che preveda una distinzione tra lettere maiuscole e minuscole, composta da un massimo di 64 caratteri ASCII.

```
aws greengrass start-bulk-deployment --cli-input-json "{
  \"InputFileUri\": \"URI of file in S3 bucket\",
  \"ExecutionRoleArn\": \"ARN of execution role\",
  \"AmznClientToken\": \"your Amazon client token\"
}"
```

Il comando deve restituire correttamente il codice 200, oltre alla seguente risposta:

```
{
  "bulkDeploymentId": UUID
}
```

Prendere nota dell'ID della distribuzione di massa. Potrà essere utilizzato per controllare lo stato della distribuzione di massa.

 Note

Sebbene le operazioni di distribuzione in blocco non siano attualmente supportate, puoi creare regole di EventBridge eventi Amazon per ricevere notifiche sulle modifiche allo stato della distribuzione per singoli gruppi. Per ulteriori informazioni, consulta [the section called "Ottenere le notifiche di distribuzione"](#).

2. Utilizzare il seguente comando per controllare lo stato della distribuzione di massa:

```
aws greengrass get-bulk-deployment-status --bulk-deployment-id 1234567
```

Il comando deve restituire il codice di stato 200 oltre a un payload JSON di informazioni:

```
{
  "BulkDeploymentStatus": Running,
  "Statistics": {
    "RecordsProcessed": integer,
    "InvalidInputRecords": integer,
    "RetryAttempts": integer
  },
  "CreatedAt": "string",
  "ErrorMessage": "string",
  "ErrorDetails": [
    {
      "DetailedErrorCode": "string",
      "DetailedErrorMessage": "string"
    }
  ]
}
```

```
}  
]  
}
```

`BulkDeploymentStatus` contiene lo stato corrente dell'esecuzione di massa. L'esecuzione può avere uno di sei diversi stati:

- **Initializing.** La richiesta di distribuzione in blocco è stata ricevuta e l'esecuzione si sta preparando per l'avvio.
- **Running.** L'esecuzione della distribuzione in blocco è iniziata.
- **Completed.** L'esecuzione della distribuzione in blocco ha terminato l'elaborazione di tutti i record.
- **Stopping.** L'esecuzione della distribuzione in blocco ha ricevuto un comando di interruzione e verrà interrotta a breve. Non potrai avviare una nuova distribuzione di massa se lo stato di quella precedente è **Stopping**.
- **Stopped.** L'esecuzione della distribuzione in blocco è stata interrotta manualmente.
- **Failed.** L'esecuzione della distribuzione in blocco ha rilevato un errore ed è terminata. Puoi trovare i dettagli dell'errore nel campo `ErrorDetails`.

Il payload JSON, inoltre, include informazioni statistiche sull'avanzamento della distribuzione di massa. Puoi utilizzare tali informazioni per stabilire quanti gruppi sono stati elaborati e quanti non lo sono stati. Le informazioni statistiche includono:

- **RecordsProcessed:** il numero di gruppi di record che si è tentato di elaborare.
- **InvalidInputRecords:** il numero totale di record che hanno restituito un errore irreversibile. Ad esempio, questo può verificarsi se un gruppo di record del file di input utilizza un formato non valido o specifica una versione di gruppo inesistente oppure se l'esecuzione non concede l'autorizzazione per la distribuzione di un gruppo o di una versione del gruppo.
- **RetryAttempts:** il numero totale di tentativi di distribuzione che hanno restituito un errore irreversibile. Ad esempio, un nuovo tentativo viene attivato se quello di distribuzione di un gruppo precedente restituisce un errore di throttling. Hai a disposizione cinque tentativi di distribuzione del gruppo.

In caso di errore di esecuzione della distribuzione di massa, il payload include anche una sezione `ErrorDetails` utilizzabile per la risoluzione dei problemi. Contiene informazioni sulla causa dell'errore di esecuzione.

Puoi controllare periodicamente lo stato della distribuzione di massa per accertarti che stia procedendo come previsto. Al termine della distribuzione, `RecordsProcessed` dovrebbe essere pari al numero di gruppi di distribuzione del file di input della distribuzione di massa. Questo indica sono stati elaborati tutti i record.

Fase 5: test della distribuzione

Utilizza il comando `ListBulkDeployments` per trovare l'ID della distribuzione di massa.

```
aws greengrass list-bulk-deployments
```

Questo comando restituisce un elenco di tutte le distribuzioni di massa, dalla più recente a quella meno recente, incluso `BulkDeploymentId`.

```
{
  "BulkDeployments": [
    {
      "BulkDeploymentId": 1234567,
      "BulkDeploymentArn": "string",
      "CreatedAt": "string"
    }
  ],
  "NextToken": "string"
}
```

Chiamare ora il comando `ListBulkDeploymentDetailedReports` per raccogliere informazioni dettagliate su ciascuna distribuzione.

```
aws greengrass list-bulk-deployment-detailed-reports --bulk-deployment-id 1234567
```

Il comando deve restituire il codice di stato 200 insieme a un payload JSON di informazioni:

```
{
  "BulkDeploymentResults": [
    {
      "DeploymentId": "string",
      "GroupVersionedArn": "string",
      "CreatedAt": "string",
      "DeploymentStatus": "string",
      "ErrorMessage": "string",
      "ErrorDetails": [
        {
          "DetailedErrorCode": "string",
          "DetailedErrorMessage": "string"
        }
      ]
    }
  ],
  "NextToken": "string"
}
```

Questo payload include in genere un elenco impaginato di ciascuna distribuzione e del relativo stato, dalla più recente alla meno recente. Contiene inoltre altre informazioni in caso di errore di esecuzione della distribuzione di massa. Anche in questo caso, il numero totale di distribuzioni in elenco deve corrispondere al numero di gruppi identificati nel file di input della distribuzione di massa.

Le informazioni restituite possono modificare finché le distribuzioni finché non sono nello stato terminale (esito positivo o negativo). Questo comando può essere richiamato periodicamente fino a quel momento.

Risoluzione dei problemi di distribuzione di massa

Se la distribuzione di massa non viene completata correttamente, puoi provare a eseguire la procedura di risoluzione dei problemi riportata di seguito. Eseguire i comandi nel terminale.

Risoluzione degli errori del file di input

La distribuzione di massa può avere esito negativo in caso di errori di sintassi nel file di input della distribuzione di massa. Questo restituisce lo stato `Failed` della distribuzione di massa, con un messaggio di errore che indica il numero di riga del primo errore di convalida. Possono verificarsi quattro tipi di errore:

- InvalidInputFile: Missing *GroupId* at line number: *line number*

Questo errore indica che la riga del file di input non è in grado di registrare il parametro specificato. I possibili parametri mancanti sono `GroupId` e `GroupVersionId`.

- InvalidInputFile: Invalid deployment type at line number : *line number*. Only valid type is 'NewDeployment'.

Questo errore indica che la riga del file di input elenca un tipo di distribuzione non valido. Al momento, l'unico tipo di distribuzione supportato è `NewDeployment`.

- Line *%s* is too long in S3 File. Valid line is less than 256 chars.

Questo errore indica che la riga del file di input è troppo lunga e deve essere abbreviata.

- Failed to parse input file at line number: *line number*

Questo errore indica che la riga del file di input non è considerata un json valido.

Verifica delle distribuzioni di massa simultanee

Non è possibile avviare una nuova distribuzione di massa mentre un'altra è ancora in esecuzione o non presenta lo stato terminale. Ciò può restituire `Concurrent Deployment Error`. Puoi utilizzare il comando `ListBulkDeployments` per verificare che una distribuzione di massa non sia attualmente in corso. Questo comando elenca le distribuzioni di massa dalla più recente alla meno recente.

```
{
```



```
"BulkDeployments": [  
  {  
    "BulkDeploymentId": BulkDeploymentId,  
    "BulkDeploymentArn": "string",  
    "CreatedAt": "string"  
  }  
],  
"NextToken": "string"  
}
```

Utilizzare `BulkDeploymentId` della prima distribuzione di massa dell'elenco per eseguire il comando `GetBulkDeploymentStatus`. Se la distribuzione di massa più recente è in esecuzione (`Initializing` o `Running`), utilizzare il comando seguente per interromperla.

```
aws greengrass stop-bulk-deployment --bulk-deployment-id BulkDeploymentId
```

Questa operazione restituirà lo stato `Stopping` finché la distribuzione non raggiunge quello `Stopped`. Dopo che la distribuzione ha raggiunto lo stato `Stopped`, potrai avviare una nuova distribuzione di massa.

Controlla ErrorDetails

Eseguire il comando `GetBulkDeploymentStatus` per restituire un payload JSON contenente informazioni dettagliate su qualsiasi errore di esecuzione della distribuzione di massa.

```
"Message": "string",  
"ErrorDetails": [  
  {  
    "DetailedErrorCode": "string",  
    "DetailedErrorMessage": "string"  
  }  
]
```

Quando si verifica un errore, il payload JSON `ErrorDetails` restituito da questa chiamata fornisce ulteriori informazioni sull'errore di esecuzione della distribuzione di massa. Ad esempio, un codice di stato dell'errore della serie 400, indica un errore di input che può essersi verificato nei parametri di input o nelle dipendenze del chiamante.

Controlla il registro AWS IoT Greengrass principale

Puoi risolvere i problemi visualizzando i log AWS IoT Greengrass principali. Utilizza i seguenti comandi per visualizzare `runtime.log`:

```
cd /greengrass/ggc/var/log  
sudo cat system/runtime.log | more
```

Per ulteriori informazioni sulla AWS IoT Greengrass registrazione, vedere. [Monitoraggio con AWS IoT Greengrass registri](#)

Consulta anche

Per ulteriori informazioni, consulta le seguenti risorse:

- [Distribuisci gruppi AWS IoT Greengrass](#)
- [Comandi dell'API Amazon S3 nel Command](#) Reference AWS CLI
- [AWS IoT Greengrass comandi nel AWS CLI Command](#) Reference

Esegui le funzioni Lambda sul core AWS IoT Greengrass

AWS IoT Greengrass fornisce un ambiente di runtime Lambda containerizzato per il codice definito dall'utente in cui si crea. AWS Lambda Funzioni Lambda distribuite in un'esecuzione principale nel AWS IoT Greengrass runtime Lambda locale del core. Le funzioni Lambda locali possono essere attivate da eventi locali, messaggi dal cloud e altre fonti, il che offre funzionalità di elaborazione locale ai dispositivi client. Ad esempio, puoi utilizzare le funzioni Greengrass Lambda per filtrare i dati del dispositivo prima di trasmetterli al cloud.

Per distribuire una funzione Lambda su un core, si aggiunge la funzione a un gruppo Greengrass (facendo riferimento alla funzione Lambda esistente), si configurano le impostazioni specifiche del gruppo per la funzione e quindi si distribuisce il gruppo. Se la funzione accede AWS ai servizi, è inoltre necessario aggiungere le autorizzazioni necessarie al ruolo del gruppo [Greengrass](#).

Puoi configurare parametri che determinano l'esecuzione delle funzioni Lambda, tra cui autorizzazioni, isolamento, limiti di memoria e altro. Per ulteriori informazioni, consulta [the section called "Controllo dell'esecuzione della funzione Greengrass Lambda"](#).

Note

Queste impostazioni consentono anche l'esecuzione AWS IoT Greengrass in un contenitore Docker. Per ulteriori informazioni, consulta [the section called "Esegui AWS IoT Greengrass in un contenitore Docker"](#).

La tabella seguente elenca i [AWS Lambda runtime](#) supportati e le versioni del software AWS IoT Greengrass Core su cui possono essere eseguiti.

Lingua o piattaforma	Versione GGC
Python 3.8	1.11
Python 3.7	1.9 o versioni successive
Python 2.7 *	1.0 o versione successiva
Java 8	1.1 o versione successiva

Lingua o piattaforma	Versione GGC
Node.js 12.x *	1.10 o versione successiva
Node.js 8.10	1.9 o versione successiva
Node.js 6.10 *	1.1 o versione successiva
C, C++	1.6 o versione successiva

* Puoi eseguire funzioni Lambda che utilizzano questi runtime su versioni supportate di AWS IoT Greengrass, ma non puoi crearle in. AWS Lambda Se il runtime sul tuo dispositivo è diverso dal runtime AWS Lambda specificato per quella funzione, puoi scegliere il tuo runtime utilizzando `FunctionRuntimeOverride` in `FunctionDefinitionVersion`. Per ulteriori informazioni, consulta [CreateFunctionDefinition](#). Per ulteriori informazioni sui runtime supportati, consulta la [politica di supporto di Runtime](#) nella AWS Lambda Developer Guide.

SDKs per le funzioni Greengrass Lambda

AWS ne fornisce tre SDKs che possono essere utilizzate dalle funzioni Greengrass Lambda in esecuzione su un core. AWS IoT Greengrass Queste SDKs sono contenute in diversi pacchetti, quindi le funzioni possono usarle contemporaneamente. Per utilizzare un SDK in una funzione Greengrass Lambda, includilo nel pacchetto di distribuzione della funzione Lambda su cui carichi. AWS Lambda

AWS IoT Greengrass SDK principale

Consente alle funzioni Lambda locali di interagire con il core per:

- Scambia messaggi MQTT con. AWS IoT Core
- Scambia messaggi MQTT con connettori, dispositivi client e altre funzioni Lambda nel gruppo Greengrass.
- Interagire con il servizio shadow locale.
- Richiama altre funzioni Lambda locali.
- Accesso a [risorse segrete](#).
- Interagire con [stream manager](#).

AWS IoT Greengrass fornisce il AWS IoT Greengrass Core SDK nelle seguenti lingue e piattaforme su GitHub

- [AWS IoT Greengrass SDK principale per Java](#)
- [AWS IoT Greengrass Core SDK per Node.js](#)
- [AWS IoT Greengrass SDK di base per Python](#)
- [AWS IoT Greengrass Core SDK per C](#)

Per includere la dipendenza AWS IoT Greengrass Core SDK nel pacchetto di distribuzione della funzione Lambda:

1. Scarica la lingua o la piattaforma del pacchetto AWS IoT Greengrass Core SDK che corrisponde al runtime della tua funzione Lambda.
2. Decomprimere il pacchetto scaricato per ottenere l'SDK. Il kit SDK è la cartella `greengrasssdk`.
3. Includi `greengrasssdk` nel pacchetto di distribuzione della funzione Lambda che contiene il codice della funzione. Questo è il pacchetto su cui carichi AWS Lambda quando crei la funzione Lambda.

StreamManagerClient

Solo il seguente AWS IoT Greengrass Core SDKs può essere utilizzato per le operazioni di [gestione degli stream](#):

- Java SDK (v1.4.0 o successivo)
- Python SDK (v1.5.0 o successivo)
- Node.js SDK (v1.6.0 o successivo)

Per utilizzare AWS IoT Greengrass Core SDK for Python per interagire con lo stream manager, devi installare Python 3.7 o versione successiva. È inoltre necessario installare le dipendenze da includere nei pacchetti di distribuzione delle funzioni Python Lambda:

1. Passare alla directory SDK che contiene il file `requirements.txt`. Questo file elenca le dipendenze.
2. Installare le dipendenze SDK. Ad esempio, eseguire il comando `pip` seguente per installarle nella directory corrente:

```
pip install --target . -r requirements.txt
```

Installa AWS IoT Greengrass Core SDK per Python sul dispositivo principale

Se stai eseguendo funzioni Python Lambda, puoi anche usarle [pip](#) per installare Core AWS IoT Greengrass SDK per Python sul dispositivo principale. Quindi puoi distribuire le tue funzioni senza includere l'SDK nel pacchetto di distribuzione delle funzioni Lambda. Per ulteriori informazioni, consulta [greengrasssdk](#).

Questo supporto è destinato ai core con vincoli di dimensione. Ti consigliamo di includere l'SDK nei pacchetti di distribuzione delle funzioni Lambda, quando possibile.

AWS IoT Greengrass SDK per il Machine Learning

Consente alle funzioni Lambda locali di utilizzare modelli di machine learning (ML) distribuiti nel core di Greengrass come risorse ML. Le funzioni Lambda possono utilizzare l'SDK per richiamare e interagire con un servizio di inferenza locale distribuito nel core come connettore. Le funzioni Lambda e i connettori ML possono anche utilizzare l'SDK per inviare dati al connettore ML Feedback per il caricamento e la pubblicazione. Per ulteriori informazioni, inclusi esempi di codice che utilizzano l'SDK, consulta [the section called “Classificazione delle immagini ML”](#), [the section called “Rilevamento di oggetti ML”](#) e [the section called “Feedback ML”](#).

La tabella seguente elenca le lingue o le piattaforme supportate per le versioni SDK e le versioni del software AWS IoT Greengrass Core su cui possono essere eseguite.

Versione SDK	Lingua o piattaforma	Versione GGC richiesta	Changelog
1.1.0	Python 3.7 o 2.7	1.9.3 o versioni successive	Aggiunto supporto Python 3.7 e nuovo cliente feedback.

Versione SDK	Lingua o piattaforma	Versione GGC richiesta	Changelog
1.0.0	Python 2.7	1.7 o versione successiva	Versione iniziale.

Per informazioni di download, consulta [the section called “AWS IoT Greengrass Software ML SDK”](#).

AWS SDKs

Consente alle funzioni Lambda locali di effettuare chiamate dirette a AWS servizi come Amazon S3, DynamoDB AWS IoT e. AWS IoT Greengrass Per utilizzare un AWS SDK in una funzione Greengrass Lambda, è necessario includerlo nel pacchetto di distribuzione. Quando utilizzi l' AWS SDK nello stesso pacchetto dell'SDK AWS IoT Greengrass Core, assicurati che le funzioni Lambda utilizzino i namespace corretti. Le funzioni Greengrass Lambda non possono comunicare con i servizi cloud quando il core è offline.

Scaricala AWS SDKs dal [Getting Started Resource Center](#).

Per ulteriori informazioni sulla creazione di un pacchetto di distribuzione, consulta [the section called “Creare e impacchettare una funzione Lambda”](#) il tutorial Getting Started o [Creating a deployment package](#) nella AWS Lambda Developer Guide.

Migrazione delle funzioni Lambda basate sul cloud

Il AWS IoT Greengrass Core SDK segue il modello di programmazione AWS SDK, che semplifica il trasferimento delle funzioni Lambda sviluppate per il cloud in funzioni Lambda eseguite su un core. AWS IoT Greengrass

Ad esempio, la seguente funzione Python Lambda utilizza AWS SDK per Python (Boto3) per pubblicare un messaggio sull'argomento `some/topic` nel cloud:

```
import boto3

iot_client = boto3.client("iot-data")
```

```
response = iot_client.publish(  
    topic="some/topic", qos=0, payload="Some payload".encode()  
)
```

Per eseguire il porting della funzione per un AWS IoT Greengrass core, nell'importazione e nell'inizializzazione, modificate il nome del boto3 modulo in `greengrasssdk`, come mostrato nell'esempio seguente:

```
import greengrasssdk  
  
iot_client = greengrasssdk.client("iot-data")  
iot_client.publish(topic="some/topic", qos=0, payload="Some payload".encode())
```

Note

Il AWS IoT Greengrass Core SDK supporta solo l'invio di messaggi MQTT con QoS = 0. Per ulteriori informazioni, consulta [the section called “Messaggio di qualità del servizio”](#).

La somiglianza tra i modelli di programmazione consente inoltre di sviluppare le funzioni Lambda nel cloud e quindi AWS IoT Greengrass migrarle con il minimo sforzo. [Gli eseguibili Lambda](#) non vengono eseguiti nel cloud, quindi non puoi utilizzare l'AWS SDK per svilupparli nel cloud prima della distribuzione.

Fai riferimento alle funzioni Lambda per alias o versione

I gruppi Greengrass possono fare riferimento a una funzione Lambda tramite alias (consigliato) o per versione. L'utilizzo di un alias semplifica la gestione degli aggiornamenti del codice perché non è necessario modificare la tabella di sottoscrizione o la definizione del gruppo quando il codice della funzione viene aggiornato. È sufficiente invece indirizzare l'alias alla nuova versione della funzione. Gli alias si trasformano in numeri di versione durante la distribuzione di gruppo. Quando si utilizzano gli alias, la versione risolta viene aggiornata alla versione a cui punta l'alias al momento della distribuzione.

AWS IoT Greengrass non supporta gli alias Lambda per le versioni `$LATEST`. Le versioni `$LATEST` non sono legate a versioni di funzioni pubblicate e immutabili e possono essere modificate in qualsiasi momento, il che è contrario al principio dell'immutabilità della versione. AWS IoT Greengrass

Una pratica comune per mantenere aggiornate le funzioni di Greengrass Lambda con le modifiche al codice consiste nell'utilizzare un alias denominato nel gruppo Greengrass e negli **PRODUCTION** abbonamenti. Quando promuovi nuove versioni della tua funzione Lambda in produzione, indirizzi l'alias all'ultima versione stabile e quindi ridistribuisce il gruppo. Puoi anche scegliere di utilizzare questo metodo per eseguire il rollback a una versione precedente.

Controllo dell'esecuzione delle funzioni Greengrass Lambda utilizzando la configurazione specifica del gruppo

AWS IoT Greengrass fornisce la gestione basata sul cloud delle funzioni di Greengrass Lambda. Sebbene il codice e le dipendenze di una funzione Lambda siano gestiti utilizzando AWS Lambda, è possibile configurare il comportamento della funzione Lambda quando viene eseguita in un gruppo Greengrass.

Impostazioni di configurazione specifiche del gruppo

AWS IoT Greengrass fornisce le seguenti impostazioni di configurazione specifiche del gruppo per le funzioni Greengrass Lambda.

Utente e gruppo di sistema

L'identità di accesso utilizzata per eseguire una funzione Lambda. Per impostazione predefinita, le funzioni Lambda vengono eseguite come identità di [accesso predefinita](#) del gruppo. Di solito, questo è l'account di sistema AWS IoT Greengrass standard (ggc_user e ggc_group). Puoi modificare l'impostazione e scegliere l'ID utente e l'ID di gruppo che dispongono delle autorizzazioni necessarie per eseguire la funzione Lambda. Puoi sostituire sia l'UID che il GID o soltanto uno di essi se lasci vuoti l'altro campo. Questa impostazione offre un controllo più granulare sull'accesso alle risorse del dispositivo. Ti consigliamo di configurare l'hardware Greengrass con limiti di risorse, autorizzazioni di file e quote disco appropriati per gli utenti e i gruppi le cui autorizzazioni vengono utilizzate per eseguire le funzioni Lambda.

Questa funzionalità è disponibile per AWS IoT Greengrass Core v1.7 e versioni successive.

Important

Ti consigliamo di evitare di eseguire le funzioni Lambda come root a meno che non sia assolutamente necessario. L'esecuzione come root aumenta i seguenti rischi:

- Il rischio di modifiche non intenzionali, come l'eliminazione accidentale di un file importante.
- Il rischio per i dati e il dispositivo causato da individui malintenzionati.
- Il rischio che si verifichi un container sfugge quando i container Docker vengono utilizzati con `--net=host` e `UID=EUID=0`

Se è necessario eseguire l'esecuzione come root, è necessario aggiornare la AWS IoT Greengrass configurazione per abilitarla. Per ulteriori informazioni, consulta [the section called "Esecuzione di una funzione Lambda come utente root"](#).

ID utente del sistema (numero)

L'ID utente dell'utente che dispone delle autorizzazioni necessarie per eseguire la funzione Lambda. Questa impostazione è disponibile solo se scegli di eseguirla come altro ID utente/ID di gruppo. Puoi usare il `getent passwd` comando sul tuo dispositivo AWS IoT Greengrass principale per cercare l'ID utente che desideri utilizzare per eseguire la funzione Lambda.

Se si utilizza lo stesso UID per eseguire i processi e la funzione Lambda su un dispositivo principale Greengrass, il ruolo del gruppo Greengrass può concedere ai processi credenziali temporanee. I processi possono utilizzare le credenziali temporanee nelle implementazioni principali di Greengrass.

ID del gruppo di sistema (numero)

L'ID di gruppo per il gruppo che dispone delle autorizzazioni necessarie per eseguire la funzione Lambda. Questa impostazione è disponibile solo se scegli di eseguirla come un altro ID utente/ID di gruppo. Puoi usare il `getent group` comando sul tuo dispositivo AWS IoT Greengrass principale per cercare l'ID del gruppo che desideri utilizzare per eseguire la funzione Lambda.

Containerizzazione della funzione Lambda

Scegli se la funzione Lambda viene eseguita con la containerizzazione predefinita per il gruppo o specifica la containerizzazione che deve essere sempre utilizzata per questa funzione Lambda.

La modalità di containerizzazione di una funzione Lambda ne determina il livello di isolamento.

- Le funzioni Lambda containerizzate vengono eseguite in modalità contenitore Greengrass. La funzione Lambda viene eseguita in un ambiente di runtime isolato (o namespace) all'interno del contenitore. AWS IoT Greengrass

- Le funzioni Lambda non containerizzate vengono eseguite in modalità No container. Le funzioni Lambda vengono eseguite come un normale processo Linux senza alcun isolamento.

Questa funzionalità è disponibile per AWS IoT Greengrass Core v1.7 e versioni successive.

Ti consigliamo di eseguire le funzioni Lambda in un contenitore Greengrass a meno che il tuo caso d'uso non richieda che vengano eseguite senza containerizzazione. Quando le funzioni Lambda vengono eseguite in un contenitore Greengrass, è possibile utilizzare le risorse locali e del dispositivo collegate e ottenere i vantaggi dell'isolamento e di una maggiore sicurezza. Prima di modificare l'impostazione di containerizzazione, consulta [the section called “Considerazioni sulla scelta della containerizzazione delle funzioni Lambda”](#).

Note

Per funzionare senza abilitare lo spazio dei nomi del kernel e il cgroup del dispositivo, tutte le funzioni Lambda devono essere eseguite senza containerizzazione. Puoi eseguire facilmente questa operazione impostando la containerizzazione predefinita per il gruppo. Per informazioni, consultare [the section called “Impostazione della containerizzazione predefinita per le funzioni Lambda in un gruppo”](#).

Memory limit (Limite memoria)

L'allocazione di memoria per la funzione. Il valore predefinito è 16 MB.

Note

L'impostazione del limite di memoria non è disponibile quando si modifica la funzione Lambda in modo che venga eseguita senza containerizzazione. Le funzioni Lambda eseguite senza containerizzazione non hanno limiti di memoria. L'impostazione del limite di memoria viene eliminata quando si modifica l'impostazione di containerizzazione predefinita della funzione Lambda o del gruppo in modo che venga eseguita senza containerizzazione.

Timeout

La quantità di tempo prima che la funzione o la richiesta venga terminata. Il valore predefinito è 3 secondi.

Bloccato

Il ciclo di vita di una funzione Lambda può essere su richiesta o di lunga durata. Il valore predefinito è on demand.

Una funzione Lambda su richiesta viene avviata in un contenitore nuovo o riutilizzato quando viene richiamata. Le richieste per la funzione potrebbero essere elaborate da qualsiasi container disponibile. Una funzione Lambda di lunga durata, o bloccata, si avvia automaticamente dopo AWS IoT Greengrass l'avvio e continua a funzionare nel proprio contenitore (o sandbox). Tutte le richieste per la funzione vengono elaborate dallo stesso container. Per ulteriori informazioni, consulta [the section called “Configurazione del ciclo di vita”](#).

Read access to /sys directory (Accesso in lettura alla directory /sys)

Se la funzione può accedere all'host'. s /sys folder. Use this when the function must read device information from /sys Il valore predefinito è false.

Note

Questa impostazione non è disponibile quando si esegue una funzione Lambda senza containerizzazione. Il valore di questa impostazione viene scartato quando si modifica la funzione Lambda in modo che venga eseguita senza containerizzazione.

Tipo di codifica

Il tipo di codifica prevista del payload di input per la funzione, in formato JSON o binario. Il valore predefinito è JSON.

Il supporto per il tipo di codifica binaria è disponibile a partire da AWS IoT Greengrass Core Software v1.5.0 e AWS IoT Greengrass Core SDK v1.1.0. L'accettazione dei dati di input binari può essere utile per le funzioni che interagiscono con i dati del dispositivo poiché le limitate funzionalità hardware dei dispositivi rendono spesso difficile o impossibile la creazione di un tipo di dati JSON.

Note

[Gli eseguibili Lambda](#) supportano solo il tipo di codifica binaria, non JSON.

Argomenti del processo

Gli argomenti della riga di comando vengono passati alla funzione Lambda durante l'esecuzione.

Variabili di ambiente

Le coppie chiave-valore che possono trasferire in modo dinamico le impostazioni al codice e alle librerie delle funzioni. Le variabili di ambiente locali funzionano nello stesso modo delle [variabili di ambiente delle funzioni AWS Lambda](#), ma sono disponibili nell'ambiente core.

Policy di accesso alle risorse

Un elenco di un massimo di 10 [risorse locali](#), [risorse segrete](#) e [risorse di apprendimento automatico](#) a cui la funzione Lambda può accedere e l'autorizzazione read-only o read-write corrispondente. Nella console, queste risorse affiliate sono elencate nella pagina di configurazione del gruppo nella scheda Risorse.

La [modalità di containerizzazione influisce sul modo in](#) cui le funzioni Lambda possono accedere alle risorse locali di dispositivi e volumi e alle risorse di machine learning.

- Le funzioni Lambda non containerizzate devono accedere alle risorse locali del dispositivo e del volume direttamente tramite il file system sul dispositivo principale.
- Per consentire alle funzioni Lambda non containerizzate di accedere alle risorse di machine learning nel gruppo Greengrass, è necessario impostare le proprietà del proprietario della risorsa e delle autorizzazioni di accesso sulla risorsa di machine learning. Per ulteriori informazioni, consulta [the section called “Accesso alle risorse di Machine Learning”](#).

Per informazioni sull'utilizzo dell' AWS IoT Greengrass API per impostare impostazioni di configurazione specifiche del gruppo per le funzioni Lambda definite dall'utente, [CreateFunctionDefinition](#) vedere nel riferimento AWS IoT Greengrass Version 1 all'API [create-function-definition](#) nel riferimento ai comandi. AWS CLI [Per distribuire le funzioni Lambda su un core di Greengrass, crea una versione di definizione della funzione che contenga le tue funzioni, crea una versione di gruppo che faccia riferimento alla versione della definizione della funzione e ad altri componenti del gruppo, quindi distribuisce il gruppo.](#)

Esecuzione di una funzione Lambda come utente root

Questa funzionalità è disponibile per AWS IoT Greengrass Core v1.7 e versioni successive.

Prima di poter eseguire una o più funzioni Lambda come root, devi prima aggiornare la AWS IoT Greengrass configurazione per abilitare il supporto. Il supporto per l'esecuzione delle funzioni

Lambda come root è disattivato per impostazione predefinita. La distribuzione non riesce se si tenta di distribuire una funzione Lambda e di eseguirla come root (UID e GID pari a 0) e non si è aggiornata la configurazione. AWS IoT Greengrass Nel log di runtime (/log) *greengrass_root* viene visualizzato un errore come il seguente: ggc/var/log/system/runtime

```
lambda(s)
[list of function arns] are configured to run as root while Greengrass is not
configured to run lambdas with root permissions
```

Important

Ti consigliamo di evitare di eseguire le funzioni Lambda come root a meno che non sia assolutamente necessario. L'esecuzione come root aumenta i seguenti rischi:

- Il rischio di modifiche non intenzionali, come l'eliminazione accidentale di un file importante.
- Il rischio per i dati e il dispositivo causato da individui malintenzionati.
- Il rischio che si verifichi un container sfugge quando i container Docker vengono utilizzati con `--net=host` e `UID=EUID=0`

Per consentire l'esecuzione delle funzioni Lambda come root

1. Sul tuo AWS IoT Greengrass dispositivo, accedi alla cartella *greengrass-root* /config.

Note

Per impostazione predefinita, *greengrass-root* è la directory /greengrass.

2. Modificare il file config.json per aggiungere "allowFunctionsToRunAsRoot" : "yes" al campo runtime. Per esempio:

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

```
...  
}
```

3. Usa i seguenti comandi per riavviare: AWS IoT Greengrass

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

Ora puoi impostare l'ID utente e l'ID di gruppo (UID/GID) delle funzioni Lambda su 0 per eseguire quella funzione Lambda come root.

Puoi modificare il valore di "allowFunctionsToRunAsRoot" to "no" e restart AWS IoT Greengrass se desideri impedire l'esecuzione delle funzioni Lambda come root.

Considerazioni sulla scelta della containerizzazione delle funzioni Lambda

Questa funzionalità è disponibile per AWS IoT Greengrass Core v1.7 e versioni successive.

Per impostazione predefinita, le funzioni Lambda vengono eseguite all'interno di un AWS IoT Greengrass contenitore. Questo container isola le funzioni dall'host, garantendo una maggiore sicurezza sia per l'host sia per le funzioni nel container.

Ti consigliamo di eseguire le funzioni Lambda in un contenitore Greengrass a meno che il tuo caso d'uso non richieda che vengano eseguite senza containerizzazione. Eseguendo le funzioni Lambda in un contenitore Greengrass, hai un maggiore controllo sulla limitazione dell'accesso alle risorse.

Di seguito sono riportati alcuni esempi di casi d'uso per l'esecuzione senza containerizzazione:

- Vuoi eseguirlo AWS IoT Greengrass su un dispositivo che non supporta la modalità contenitore (ad esempio, perché stai usando una distribuzione Linux speciale o hai una versione del kernel troppo vecchia).
- Vuoi eseguire la tua funzione Lambda in un altro ambiente contenitore con il proprio OverlayFS, ma riscontri conflitti OverlayFS quando esegui in un contenitore Greengrass.
- È necessario accedere alle risorse locali con percorsi che non possono essere determinati al momento della distribuzione o che possono variare dopo la distribuzione, come nel caso dei dispositivi integrabili.
- Hai un'applicazione legacy che è stata scritta come processo e hai riscontrato problemi durante l'esecuzione come funzione Lambda containerizzata.

Differenze nella containerizzazione

Containerizzazione	Note
Container Greengrass	• Tutte le AWS IoT Greengrass funzionalità sono disponibili quando si esegue una funzione Lambda in un contenitore Greengrass. • Le funzioni Lambda eseguite in un contenitore Greengrass non hanno accesso al codice distribuito di altre funzioni Lambda, anche se vengono eseguite con lo stesso ID di gruppo. In altre parole, le funzioni Lambda vengono eseguite con un maggiore isolamento l'una dall'altra. • Poiché le funzioni Lambda eseguite in un AWS IoT Greengrass contenitore hanno tutti i processi figlio eseguiti nello stesso contenitore della funzione Lambda, i processi figlio vengono terminati quando viene terminata la funzione Lambda.
Nessun container	• Le seguenti funzionalità non sono disponibili per le funzioni Lambda non containerizzate: • Limiti di memoria della funzione Lambda. • Risorse volume e dispositivo locale. È necessario accedere direttamente a queste risorse sul dispositivo core anziché accedervi come membri del gruppo Greengrass. • Se la funzione Lambda non containerizzata accede a una risorsa di machine learning, devi identificare il proprietario della risorsa e impostare le autorizzazioni di accesso sulla risorsa, non sulla funzione Lambda. Ciò richiede il software Core v1.10 o successivo.

Containerizzazione	Note
	AWS IoT Greengrass Per ulteriori informazioni, consulta the section called “Accesso alle risorse di Machine Learning”. • La funzione Lambda ha accesso in sola lettura al codice distribuito di altre funzioni Lambda in esecuzione con lo stesso ID di gruppo. • Le funzioni Lambda che generano processi secondari in una sessione di processo diversa o con un gestore SIGHUP (signal hangup) sovrascritto, ad esempio con l'utilità nohup, non vengono terminate automaticamente quando viene terminata la funzione Lambda principale. AWS IoT Greengrass

 Note

L'impostazione predefinita della containerizzazione per il gruppo Greengrass non si applica ai [connettori](#).

La modifica della containerizzazione per una funzione Lambda può causare problemi durante la distribuzione. Se alla funzione Lambda sono state assegnate risorse locali che non sono più disponibili con le nuove impostazioni di containerizzazione, la distribuzione non riesce.

- Quando si modifica una funzione Lambda dall'esecuzione in un contenitore Greengrass all'esecuzione senza containerizzazione, i limiti di memoria per la funzione vengono eliminati. È necessario accedere direttamente al file system anziché utilizzare le risorse locali collegate. Prima della distribuzione, è necessario rimuovere le risorse collegate.
- Quando si modifica una funzione Lambda dall'esecuzione senza containerizzazione all'esecuzione in un contenitore, la funzione Lambda perde l'accesso diretto al file system. È necessario definire un limite di memoria per ogni funzione oppure accettare il valore predefinito di 16 MB. Puoi configurare queste impostazioni per ogni funzione Lambda prima della distribuzione.

Per modificare le impostazioni di containerizzazione per una funzione Lambda

1. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).
2. Scegliete il gruppo che contiene la funzione Lambda di cui desiderate modificare le impostazioni.
3. Scegli la scheda Funzioni Lambda.
4. Sulla funzione Lambda che desideri modificare, scegli i puntini di sospensione (...), quindi scegli Modifica configurazione.
5. Modifica le impostazioni di containerizzazione. Se configuri la funzione Lambda per l'esecuzione in un contenitore Greengrass, devi anche impostare Limite di memoria e accesso in lettura alla directory /sys.
6. Scegli Salva e quindi Conferma per salvare le modifiche alla tua funzione Lambda.

Le modifiche diventano effettive quando il gruppo viene distribuito.

Puoi anche usare il comando [CreateFunctionDefinition](#) e [CreateFunctionDefinitionVersion](#) nell'AWS IoT Greengrass API Reference. Se stai modificando l'impostazione di containerizzazione, assicurati di aggiornare anche gli altri parametri. Ad esempio, se state passando dall'esecuzione di una funzione Lambda in un contenitore Greengrass all'esecuzione senza containerizzazione, assicuratevi di cancellare il parametro. `MemorySize`

Determinare le modalità di isolamento supportate dal dispositivo Greengrass in uso

Puoi utilizzare il correttore delle AWS IoT Greengrass dipendenze per determinare quali modalità di isolamento (Greengrass container/no container) sono supportate dal tuo dispositivo Greengrass.

Per eseguire il controllo delle dipendenze AWS IoT Greengrass

1. [Scarica ed esegui il controllo delle AWS IoT Greengrass dipendenze dal repository. GitHub](#)

```
wget https://github.com/aws-samples/aws-greengrass-samples/raw/master/greengrass-  
dependency-checker-GGCv1.11.x.zip  
unzip greengrass-dependency-checker-GGCv1.11.x.zip  
cd greengrass-dependency-checker-GGCv1.11.x  
sudo modprobe configs  
sudo ./check_ggc_dependencies | more
```

2. Nel punto in cui viene visualizzato `more`, premi il tasto Spacebar per visualizzare un'altra pagina di testo.

Per informazioni sul comando `modprobe`, esegui `man modprobe` nel terminale.

Impostazione dell'identità di accesso predefinita per le funzioni Lambda in un gruppo

Questa funzionalità è disponibile per AWS IoT Greengrass Core v1.8 e versioni successive.

Per un maggiore controllo sull'accesso alle risorse del dispositivo, puoi configurare l'identità di accesso predefinita utilizzata per eseguire le funzioni Lambda nel gruppo. Questa impostazione determina le autorizzazioni predefinite concesse alle funzioni Lambda quando vengono eseguite sul dispositivo principale. Per sovrascrivere l'impostazione per singole funzioni nel gruppo, puoi utilizzare la proprietà `Run as` (Esegui come) della funzione. Per ulteriori informazioni, consulta [Run as \(Esegui come\)](#).

Questa impostazione a livello di gruppo viene utilizzata anche per eseguire il software Core sottostante. AWS IoT Greengrass Si tratta di funzioni Lambda di sistema che gestiscono le operazioni, come il routing dei messaggi, la sincronizzazione degli shadow locali e il rilevamento automatico degli indirizzi IP.

L'identità di accesso predefinita può essere configurata per essere eseguita come account di AWS IoT Greengrass sistema standard (`ggc_user` e `ggc_group`) o utilizzare le autorizzazioni di un altro utente o gruppo. Si consiglia di configurare l'hardware Greengrass con limiti di risorse, autorizzazioni di file e quote disco appropriati per tutti gli utenti e i gruppi le cui autorizzazioni vengono utilizzate per eseguire funzioni Lambda definite dall'utente o di sistema.

Per modificare l'identità di accesso predefinita per il gruppo AWS IoT Greengrass

1. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).
2. Scegli il gruppo di cui desideri modificare le impostazioni.
3. Scegli la scheda Funzioni Lambda e, nella sezione Ambiente di runtime della funzione Lambda predefinito, scegli Modifica.
4. Nella pagina Modifica ambiente di runtime della funzione Lambda predefinito, in Utente e gruppo di sistema predefiniti, scegli Un altro ID utente/ID gruppo.

Quando scegli questa opzione, vengono visualizzati i campi ID utente di sistema (numero) e ID del gruppo di sistema (numero).

5. Inserire un ID utente, un ID gruppo o entrambi. Se si lascia un campo vuoto, viene utilizzato il rispettivo account di sistema Greengrass (ggc_user o ggc_group).
 - Per ID utente di sistema (numero), inserisci l'ID utente dell'utente che dispone delle autorizzazioni che desideri utilizzare di default per eseguire le funzioni Lambda nel gruppo. È possibile utilizzare il comando `getent passwd` sul dispositivo AWS IoT Greengrass per cercare l'ID utente.
 - Per ID gruppo di sistema (numero), inserisci l'ID del gruppo che dispone delle autorizzazioni che desideri utilizzare di default per eseguire le funzioni Lambda nel gruppo. È possibile utilizzare il comando `getent group` sul dispositivo AWS IoT Greengrass per cercare l'ID gruppo.

 Important

L'esecuzione come utente radice aumenta rischi per i dati e il dispositivo. Non eseguire come radice (UID/GID = 0) a meno che non sia richiesto dal business case. Per ulteriori informazioni, consulta [the section called “Esecuzione di una funzione Lambda come utente root”](#).

Le modifiche diventano effettive quando il gruppo viene distribuito.

Impostazione della containerizzazione predefinita per le funzioni Lambda in un gruppo

Questa funzionalità è disponibile per AWS IoT Greengrass Core v1.7 e versioni successive.

L'impostazione di containerizzazione per un gruppo Greengrass determina la containerizzazione predefinita per le funzioni Lambda nel gruppo.

- In modalità contenitore Greengrass, per impostazione predefinita, le funzioni Lambda vengono eseguite in un ambiente di runtime isolato all'interno del AWS IoT Greengrass contenitore.
- In modalità No container, le funzioni Lambda vengono eseguite come normali processi Linux per impostazione predefinita.

È possibile modificare le impostazioni del gruppo per specificare la containerizzazione predefinita per le funzioni Lambda nel gruppo. Puoi sovrascrivere questa impostazione per una o più funzioni Lambda nel gruppo se desideri che le funzioni Lambda vengano eseguite con una containerizzazione

diversa da quella predefinita del gruppo. Prima di modificare le impostazioni di containerizzazione, consulta [the section called “Considerazioni sulla scelta della containerizzazione delle funzioni Lambda”](#).

 Important

Se desideri modificare la containerizzazione predefinita per il gruppo, ma disponi di una o più funzioni che utilizzano una containerizzazione diversa, modifica le impostazioni per le funzioni Lambda prima di modificare l'impostazione del gruppo. Se modifichi prima l'impostazione di containerizzazione del gruppo, i valori delle impostazioni Memory Limit (Limite memoria) e Read access to /sys directory (Accesso in lettura alla directory /sys) vengono ignorati.

Per modificare le impostazioni di containerizzazione per il tuo gruppo AWS IoT Greengrass

1. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).
2. Scegli il gruppo di cui desideri modificare le impostazioni.
3. Scegli la scheda Funzioni Lambda.
4. In Ambiente di runtime della funzione Lambda predefinito, scegli Modifica.
5. Nella pagina Modifica ambiente di runtime della funzione Lambda predefinito, in Containerizzazione predefinita della funzione Lambda, modificare l'impostazione di containerizzazione.
6. Seleziona Salva.

Le modifiche diventano effettive quando il gruppo viene distribuito.

Flussi di comunicazione per le funzioni Greengrass Lambda

Le funzioni di Greengrass Lambda supportano diversi metodi di comunicazione con altri membri del AWS IoT Greengrass gruppo, servizi locali e servizi cloud (inclusi i servizi). AWS

Comunicazione tramite messaggi MQTT

Le funzioni Lambda possono inviare e ricevere messaggi MQTT utilizzando un pattern di pubblicazione e sottoscrizione controllato dagli abbonamenti.

Questo flusso di comunicazione consente alle funzioni Lambda di scambiare messaggi con le seguenti entità:

- Dispositivi client del gruppo.
- Connettori nel gruppo.
- Altre funzioni Lambda del gruppo.
- AWS IoT.
- Servizio Device Shadow locale

Una sottoscrizione definisce un'origine del messaggio, una destinazione del messaggio e un argomento (o oggetto) utilizzato per instradare i messaggi dall'origine alla destinazione. I messaggi pubblicati su una funzione Lambda vengono passati al gestore registrato della funzione. Le sottoscrizioni garantiscono maggiore sicurezza e consentono interazioni prevedibili. Per ulteriori informazioni, consulta [the section called “Sottoscrizioni gestite nel flusso di lavoro di messaggistica MQTT”](#).

Note

Quando il core è offline, le funzioni di Greengrass Lambda possono scambiare messaggi con dispositivi client, connettori, altre funzioni e shadow locali, ma i messaggi vengono messi in coda. AWS IoT Per ulteriori informazioni, consulta [the section called “Coda di messaggi MQTT”](#).

Altri flussi di comunicazione

- Per interagire con le risorse locali di dispositivi e volumi e i modelli di machine learning su un dispositivo principale, le funzioni Greengrass Lambda utilizzano interfacce di sistema operativo specifiche della piattaforma. Ad esempio, puoi utilizzare il metodo `open` nel modulo `os` nelle funzioni Python. Per poter accedere a una risorsa, una funzione deve essere affiliata alla risorsa e deve disporre dell'autorizzazione `read-only` o `read-write`. Per ulteriori informazioni, inclusa la disponibilità della versione AWS IoT Greengrass base, vedere e. [Accesso alle risorse locali the section called “Accesso alle risorse di machine learning dal codice della funzione Lambda”](#)

Note

Se esegui la funzione Lambda senza containerizzazione, non puoi utilizzare le risorse locali e di volume collegate e devi accedere direttamente a tali risorse.

- Le funzioni Lambda possono utilizzare il Lambda client nell'SDK AWS IoT Greengrass Core per richiamare altre funzioni Lambda nel gruppo Greengrass.
- Le funzioni Lambda possono utilizzare l' AWS SDK per comunicare con i servizi. AWS [Per ulteriori informazioni, consulta AWS SDK.](#)
- Le funzioni Lambda possono utilizzare interfacce di terze parti per comunicare con servizi cloud esterni, in modo simile alle funzioni Lambda basate sul cloud.

Note

Le funzioni di Greengrass Lambda non possono comunicare con AWS altri servizi cloud quando il core è offline.

Recupero dell'argomento MQTT di input (o oggetto)

AWS IoT Greengrass utilizza gli abbonamenti per controllare lo scambio di messaggi MQTT tra dispositivi client, funzioni Lambda e connettori in un gruppo e con AWS IoT o il servizio shadow locale. Le sottoscrizioni definiscono un'origine messaggio, destinazione messaggio e un argomento MQTT utilizzati per instradare i messaggi. Quando la destinazione è una funzione Lambda, il gestore della funzione viene richiamato quando l'origine pubblica un messaggio. Per ulteriori informazioni, consulta [the section called “Comunicazione tramite messaggi MQTT”](#).

L'esempio seguente mostra come una funzione Lambda può ottenere l'argomento di input da context quello passato al gestore. A tale scopo, esegue l'accesso alla chiave subject dalla gerarchia del contesto (`context.client_context.custom[ 'subject' ]`). Nell'esempio viene inoltre analizzato il messaggio JSON di input e quindi pubblicati l'argomento e il messaggio analizzati.

Note

Nell' AWS IoT Greengrass API, l'argomento di un [abbonamento](#) è rappresentato dalla subject proprietà.

```
import greengrasssdk
import logging

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

OUTPUT_TOPIC = 'test/topic_results'

def get_input_topic(context):
    try:
        topic = context.client_context.custom['subject']
    except Exception as e:
        logging.error('Topic could not be parsed. ' + repr(e))
    return topic

def get_input_message(event):
    try:
        message = event['test-key']
    except Exception as e:
        logging.error('Message could not be parsed. ' + repr(e))
    return message

def function_handler(event, context):
    try:
        input_topic = get_input_topic(context)
        input_message = get_input_message(event)
        response = 'Invoked on topic "%s" with message "%s"' % (input_topic,
input_message)
        logging.info(response)
    except Exception as e:
        logging.error(e)

    client.publish(topic=OUTPUT_TOPIC, payload=response)

    return
```

Per testare la funzione, aggiungerlo al gruppo utilizzando le impostazioni di configurazione predefinite. Quindi, aggiungere le seguenti sottoscrizioni e distribuire il gruppo. Per istruzioni, consulta [the section called “Modulo 3 \(parte 1\): funzioni Lambda su AWS IoT Greengrass”](#).

di
argomenti

t_message

c_results

Al termine della distribuzione, invoca la funzione.

1. Nella AWS IoT console, apri la pagina del client di test MQTT.
2. Effettuate la sottoscrizione all'`test/topic_results` argomento selezionando la scheda Sottoscrivi a un argomento.
3. Pubblica un messaggio sull'`test/input_message` argomento selezionando la scheda Pubblica su un argomento. Per questo esempio, devi includere la proprietà `test-key` nel messaggio JSON.

```
{
  "test-key": "Some string value"
}
```

In caso di esito positivo, la funzione pubblica l'argomento di input e la stringa di messaggio nell'argomento `test/topic_results`.

Configurazione del ciclo di vita per le funzioni Greengrass Lambda

Il ciclo di vita della funzione Greengrass Lambda determina quando una funzione viene avviata e come crea e utilizza i contenitori. Inoltre, il ciclo di vita determina il modo in cui vengono conservate le variabili e la logica di preelaborazione che sono al di fuori dell'handler della funzione.

AWS IoT Greengrass supporta i cicli di vita on-demand (impostazione predefinita) o di lunga durata:

- Le funzioni on demand vengono avviate quando vengono richiamate e terminano quando non vi sono più attività da eseguire. Il richiamo di una funzione crea un container (o sandbox) separato per elaborare i richiami, a meno che un container esistente non sia disponibile per essere riutilizzato. I dati inviati alla funzione possono essere stati estratti da uno qualsiasi dei container.

È possibile eseguire in parallelo più richiami di una funzione on demand.

Le variabili e la logica di preelaborazione definite al di fuori dell'handler della funzione non vengono conservate quando vengono creati nuovi container.

- Le funzioni di lunga durata (o bloccate) si avviano automaticamente all'avvio del AWS IoT Greengrass core e vengono eseguite in un unico contenitore. Tutti i dati inviati alla funzione vengono estratti dallo stesso container.

Numerosi richiami vengono messi in coda finché non vengono eseguiti i richiami precedenti.

Le variabili e la logica di preelaborazione definite al di fuori dell'handler della funzione vengono conservate per ogni richiamo dell'handler.

Le funzioni Lambda di lunga durata sono utili quando è necessario iniziare a lavorare senza alcun input iniziale. Ad esempio, una funzione di lunga durata è in grado di caricare e avviare l'elaborazione di un modello ML in modo che sia pronto quando la funzione inizia a ricevere i dati del dispositivo.

Note

Ricorda che le funzioni di lunga durata dispongono di timeout associati ai richiami dell'handler. Per eseguire il codice in esecuzione a tempo indeterminato, è necessario avviarlo al di fuori dell'handler. Assicurati che al di fuori dell'handler non vi sia alcun codice di blocco che possa impedire il completamento dell'inizializzazione della funzione. Queste funzioni vengono eseguite a meno che il core non si arresti (ad esempio, durante una distribuzione di gruppo o il riavvio di un dispositivo) o che la funzione non entri in uno stato di errore (come un timeout del gestore, un'eccezione non rilevata o quando supera i limiti di memoria).

Per ulteriori informazioni sul riutilizzo dei contenitori, consulta [Understanding Container Reuse nel blog di Compute](#). AWS Lambda AWS

Eseguibili Lambda

Questa funzionalità è disponibile per AWS IoT Greengrass Core v1.6 e versioni successive.

Un eseguibile Lambda è un tipo di funzione Greengrass Lambda che è possibile utilizzare per eseguire codice binario nell'ambiente principale. Consente di eseguire funzionalità specifiche del dispositivo in modo nativo e di sfruttare l'ingombro ridotto del codice compilato. Gli eseguibili Lambda possono essere richiamati da eventi, richiamare altre funzioni e accedere a risorse locali.

Gli eseguibili Lambda supportano solo il tipo di codifica binaria (non JSON), ma per il resto puoi gestirli nel tuo gruppo Greengrass e distribuirli come le altre funzioni di Greengrass Lambda. Tuttavia, il processo di creazione degli eseguibili Lambda è diverso dalla creazione delle funzioni Lambda Python, Java e Node.js:

- Non puoi usare la AWS Lambda console per creare (o gestire) un eseguibile Lambda. Puoi creare un eseguibile Lambda solo utilizzando l' AWS Lambda API.
- Carichi il codice della funzione AWS Lambda come eseguibile compilato che include [AWS IoT Greengrass Core SDK per C](#).
- È necessario specificare il nome dell'eseguibile come handler della funzione.

Gli eseguibili Lambda devono implementare determinate chiamate e modelli di programmazione nel codice della funzione. Ad esempio, il metodo `main` deve effettuare le seguenti operazioni:

- Eseguire una chiamata a `gg_global_init` per inizializzare le variabili globali interne Greengrass. Questa funzione deve essere chiamata prima di creare qualsiasi thread e prima di chiamare qualsiasi altra funzione AWS IoT Greengrass Core SDK.
- Chiama `gg_runtime_start` per registrare il gestore di funzioni con il runtime Greengrass Lambda. Questa funzione deve essere chiamata durante l'inizializzazione. La chiamata di questa funzione fa sì che il thread corrente venga utilizzato dal runtime. Il parametro facoltativo `GG_RT_OPT_ASYNC` indica a questa funzione di non bloccare, ma di creare un nuovo thread per il runtime. Questa funzione utilizza un handler `SIGTERM`.

[Il frammento seguente è il `main` metodo tratto dall'esempio di codice `simple\_handler.c` in `poi`.](#) GitHub

```
int main() {
    gg_error err = GGE_SUCCESS;

    err = gg_global_init(0);
    if(err) {
        gg_log(GG_LOG_ERROR, "gg_global_init failed %d", err);
        goto cleanup;
    }

    gg_runtime_start(handler, 0);

cleanup:
    return -1;
}
```

[Per ulteriori informazioni su requisiti, vincoli e altri dettagli di implementazione, consulta Core SDK for C.AWS IoT Greengrass](#)

Creare un file eseguibile Lambda

Dopo aver compilato il codice insieme all'SDK, usa l' AWS Lambda API per creare una funzione Lambda e caricare l'eseguibile compilato.

Note

La funzione devono essere compilata utilizzando un compilatore compatibile con C89.

L'esempio seguente utilizza il comando [CLI create-function](#) per creare un eseguibile Lambda. Il comando specifica:

- Il nome dell'eseguibile per l'handler. Deve corrispondere al nome esatto dell'eseguibile compilato.
- Il percorso del file .zip contenente l'eseguibile compilato.
- `arn:aws:greengrass:::runtime/function/executable` per il runtime. Questo è il runtime per tutti gli eseguibili Lambda.

 Note

Infattirole, puoi specificare l'ARN di qualsiasi ruolo di esecuzione Lambda. AWS IoT Greengrass non utilizza questo ruolo, ma il parametro è necessario per creare la funzione. Per ulteriori informazioni sui ruoli di esecuzione Lambda, consulta il [modello di AWS Lambda autorizzazioni](#) nella Guida per gli AWS Lambda sviluppatori.

```
aws lambda create-function \  
--region aws-region \  
--function-name function-name \  
--handler executable-name \  
--role role-arn \  
--zip-file fileb://file-name.zip \  
--runtime arn:aws:greengrass::runtime/function/executable
```

Successivamente, utilizza l' AWS Lambda API per pubblicare una versione e creare un alias.

- Utilizza [publish-version](#) per pubblicare una versione della funzione.

```
aws lambda publish-version \  
--function-name function-name \  
--region aws-region
```

- Utilizza [create-alias](#) per creare un alias che punti alla versione appena pubblicata. Ti consigliamo di fare riferimento alle funzioni Lambda tramite alias quando le aggiungi a un gruppo Greengrass.

```
aws lambda create-alias \  
--function-name function-name \  
--name alias-name \  
--function-version version-number \  
--region aws-region
```

 Note

La AWS Lambda console non visualizza gli eseguibili Lambda. Per aggiornare il codice della funzione, è necessario utilizzare l' AWS Lambda API.

Quindi, aggiungi l'eseguibile Lambda a un gruppo Greengrass, configuralo per accettare dati di input binari nelle impostazioni specifiche del gruppo e distribuisci il gruppo. Puoi farlo nella AWS IoT Greengrass console o utilizzando l'API. AWS IoT Greengrass

Esecuzione AWS IoT Greengrass in un contenitore Docker

AWS IoT Greengrass [può essere configurato per l'esecuzione in un contenitore Docker](#).

Puoi scaricare un Dockerfile tramite [Amazon su CloudFront](#) cui sono installati il software AWS IoT Greengrass Core e le dipendenze. Per modificare l'immagine Docker per l'esecuzione su architetture della piattaforma diverse o ridurre la dimensione dell'immagine Docker, consulta il file README nel download del pacchetto Docker.

Per aiutarti a iniziare a sperimentare AWS IoT Greengrass, fornisce AWS anche immagini Docker predefinite su cui sono installati il AWS IoT Greengrass software Core e le dipendenze. Puoi scaricare un'immagine da [Docker Hub](#) o [Amazon Elastic Container Registry](#) (Amazon ECR). Queste immagini predefinite utilizzano immagini di base Amazon Linux 2 (x86_64) e Alpine Linux (x86_64, ARMv7L o). AArch64

Important

Il 30 giugno 2022, AWS IoT Greengrass è terminata la manutenzione delle immagini Docker del software AWS IoT Greengrass Core v1.x pubblicate su Amazon Elastic Container Registry (Amazon ECR) e Docker Hub. Puoi continuare a scaricare queste immagini Docker da Amazon ECR e Docker Hub fino al 30 giugno 2023, ovvero 1 anno dopo la fine della manutenzione. Tuttavia, le immagini Docker del software AWS IoT Greengrass Core v1.x non ricevono più patch di sicurezza o correzioni di bug dopo la fine della manutenzione il 30 giugno 2022. Se esegui un carico di lavoro di produzione che dipende da queste immagini Docker, ti consigliamo di creare le tue immagini Docker utilizzando i Dockerfile forniti. AWS IoT Greengrass Per ulteriori informazioni, consulta [AWS IoT Greengrass Software Docker](#).

Questo argomento descrive come scaricare l'immagine AWS IoT Greengrass Docker da Amazon ECR ed eseguirla su una piattaforma Windows, macOS o Linux (x86_64). L'argomento include le seguenti fasi:

1. [Ottieni l'immagine del AWS IoT Greengrass contenitore da Amazon ECR](#)
2. [Creazione e configurazione del gruppo e del core Greengrass](#)

3. [Esegui AWS IoT Greengrass localmente](#)
4. [Configurazione della containerizzazione "No container" per il gruppo](#)
5. [Distribuisci le funzioni Lambda nel contenitore Docker](#)
6. [\(Facoltativo\) Implementa i dispositivi client che interagiscono con Greengrass nel contenitore Docker](#)

Le seguenti funzionalità non sono supportate quando vengono eseguite in un contenitore Docker: AWS IoT Greengrass

- [Connettori](#) eseguiti in modalità container Greengrass. Per eseguire un connettore in un container Docker, il connettore deve essere eseguito in modalità Nessun container. Per trovare i connettori che supportano la modalità Nessun container consulta [the section called "AWS-connettori Greengrass forniti"](#). Alcuni di questi connettori dispongono di un parametro della modalità di isolamento che devi impostare su No container (Nessun contenitore).
- [Risorse volume e dispositivo locale](#). Le funzioni Lambda definite dall'utente che vengono eseguite nel contenitore Docker devono accedere direttamente ai dispositivi e ai volumi sul core.

Queste funzionalità non sono supportate quando l'ambiente di runtime Lambda per il gruppo Greengrass è impostato su [Nessun contenitore](#), necessario per l'esecuzione AWS IoT Greengrass in un contenitore Docker.

Prerequisiti

Prima di iniziare questo tutorial, devi fare quanto segue.

- È necessario installare il software e le versioni seguenti sul computer host in base alla versione AWS Command Line Interface (AWS CLI) scelta.

AWS CLI version 2

- [Docker](#) versione 18.09 o successiva. Anche le versioni precedenti potrebbero funzionare, ma consigliamo 18.09 o successive.
- AWS CLI versione 2.0.0 o successiva.
 - Per installare la AWS CLI versione 2, vedere [Installazione della AWS CLI versione 2](#).
 - Per configurare AWS CLI, vedere [Configurazione di AWS CLI](#).

 Note

Per eseguire l'aggiornamento a una AWS CLI versione 2 successiva su un computer Windows, è necessario ripetere la procedura di [installazione MSI](#).

AWS CLI version 1

- [Docker](#) versione 18.09 o successiva. Anche le versioni precedenti potrebbero funzionare, ma consigliamo 18.09 o successive.
- [Python](#) versione 3.6 o successiva.
- [pip](#) versione 18.1 o successiva.
- AWS CLI versione 1.17.10 o successiva
 - Per installare la AWS CLI versione 1, vedi [Installazione della AWS CLI versione 1](#).
 - Per configurare AWS CLI, vedere [Configurazione di AWS CLI](#).
 - Per eseguire l'aggiornamento alla versione più recente della AWS CLI versione 1, esegui il comando seguente.

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

 Note

Se utilizzi l'[installazione MSI](#) della AWS CLI versione 1 su Windows, tieni presente quanto segue:

- Se l'installazione della AWS CLI versione 1 non riesce a installare botocore, prova a utilizzare l'installazione di [Python](#) e pip.
 - Per eseguire l'aggiornamento a una AWS CLI versione 1 successiva, è necessario ripetere il processo di installazione MSI.
- Per accedere alle risorse di Amazon Elastic Container Registry (Amazon ECR), devi concedere la seguente autorizzazione.
 - Amazon ECR richiede agli utenti di concedere l'`ecr:GetAuthorizationToken` autorizzazione tramite una policy AWS Identity and Access Management (IAM) prima di poter autenticarsi in un registro e inviare o estrarre immagini da un repository Amazon ECR. Per ulteriori informazioni, consulta [Amazon ECR Repository Policy Examples e Accessing One Amazon ECR Repository nella Amazon Elastic Container Registry User Guide](#).

Fase 1: ottenere l'immagine del AWS IoT Greengrass contenitore da Amazon ECR

AWS fornisce immagini Docker su cui è installato il software AWS IoT Greengrass Core.

Warning

A partire dalla v1.11.6 del software AWS IoT Greengrass Core, le immagini Greengrass Docker non includono più Python 2.7, perché Python 2.7 end-of-life è arrivato nel 2020 e non riceve più aggiornamenti di sicurezza. Se scegli di eseguire l'aggiornamento a queste immagini Docker, ti consigliamo di verificare che le tue applicazioni funzionino con le nuove immagini Docker prima di distribuire gli aggiornamenti sui dispositivi di produzione. Se hai bisogno di Python 2.7 per la tua applicazione che utilizza un'immagine Greengrass Docker, puoi modificare il Dockerfile Greengrass per includere Python 2.7 per la tua applicazione.

Per i passaggi che mostrano come estrarre l'latest immagine da Amazon ECR, scegli il tuo sistema operativo:

Estrazione dell'immagine del container (Linux)

Esegui i seguenti comandi nel terminale del computer.

1. Accedi al AWS IoT Greengrass registro in Amazon ECR.

```
aws ecr get-login-password --region us-west-2 | docker login --username AWS --password-stdin https://216483018798.dkr.ecr.us-west-2.amazonaws.com
```

In caso di esito positivo, l'output stampa `Login Succeeded`.

2. Recupera l'immagine del AWS IoT Greengrass contenitore.

```
docker pull 216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

Note

L'latest immagine contiene l'ultima versione stabile del software AWS IoT Greengrass Core installato su un'immagine base di Amazon Linux 2. È anche possibile estrarre altre

versioni dall'archivio. Per trovare tutte le immagini disponibili, controlla la pagina Tag in [Docker Hub](#) o usa il comando `aws ecr list-images`. Per esempio:

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --  
repository-name aws-iot-greengrass
```

3. Abilita la protezione symlink e hardlink. Se stai sperimentando l'esecuzione AWS IoT Greengrass in un container, puoi abilitare le impostazioni solo per l'avvio corrente.

Note

Potresti dover utilizzare `sudo` per eseguire questi comandi.

- Per abilitare le impostazioni solo per l'avvio corrente:

```
echo 1 > /proc/sys/fs/protected_hardlinks  
echo 1 > /proc/sys/fs/protected_symlinks
```

- Per rendere le impostazioni persistenti in caso di riavvio:

```
echo '# AWS IoT Greengrass' >> /etc/sysctl.conf  
echo 'fs.protected_hardlinks = 1' >> /etc/sysctl.conf  
echo 'fs.protected_symlinks = 1' >> /etc/sysctl.conf  
  
sysctl -p
```

4. Abilita l'inoltro di IPv4 rete, necessario per l'implementazione AWS IoT Greengrass cloud e le comunicazioni MQTT per funzionare su Linux. Nel file `/etc/sysctl.conf`, imposta `net.ipv4.ip_forward` su 1, quindi ricarica `sysctls`.

```
sudo nano /etc/sysctl.conf  
# set this net.ipv4.ip_forward = 1  
sudo sysctl -p
```

Note

Puoi utilizzare l'editor che desideri al posto di `nano`.

Estrazione dell'immagine del container (macOS)

Esegui i seguenti comandi nel terminale del computer.

1. Accedi al AWS IoT Greengrass registro in Amazon ECR.

```
aws ecr get-login-password --region us-west-2 | docker login --username AWS --password-stdin https://216483018798.dkr.ecr.us-west-2.amazonaws.com
```

In caso di esito positivo, l'output stampa `Login Succeeded`.

2. Recupera l'immagine del AWS IoT Greengrass contenitore.

```
docker pull 216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

Note

L'latest immagine contiene l'ultima versione stabile del software AWS IoT Greengrass Core installato su un'immagine base di Amazon Linux 2. È anche possibile estrarre altre versioni dall'archivio. Per trovare tutte le immagini disponibili, controlla la pagina Tag in [Docker Hub](#) o usa il comando `aws ecr list-images`. Per esempio:

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --repository-name aws-iot-greengrass
```

Estrazione dell'immagine del container (Windows)

Esegui i comandi seguenti in un prompt di comandi. Prima di utilizzare i comandi di Docker su Windows, Docker Desktop deve essere in esecuzione.

1. Accedi al AWS IoT Greengrass registro in Amazon ECR.

```
aws ecr get-login-password --region us-west-2 | docker login --username AWS --password-stdin https://216483018798.dkr.ecr.us-west-2.amazonaws.com
```

In caso di esito positivo, l'output stampa `Login Succeeded`.

2. Recupera l'immagine del AWS IoT Greengrass contenitore.

```
docker pull 216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

Note

L'latest immagine contiene l'ultima versione stabile del software AWS IoT Greengrass Core installato su un'immagine base di Amazon Linux 2. È anche possibile estrarre altre versioni dall'archivio. Per trovare tutte le immagini disponibili, controlla la pagina Tag in [Docker Hub](#) o usa il comando `aws ecr list-images`. Per esempio:

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --  
repository-name aws-iot-greengrass
```

Fase 2: creazione e configurazione del gruppo e del core Greengrass

Nell'immagine Docker è installato il software AWS IoT Greengrass Core, ma è necessario creare un gruppo e un core Greengrass. Questa operazione include il download dei certificati e del file di configurazione del core.

- Seguire la procedura riportata in [the section called “Modulo 2: Installazione del software AWS IoT Greengrass Core”](#). Salta i passaggi in cui scarichi ed esegui il AWS IoT Greengrass software Core. Il software e le dipendenze di runtime sono già impostati nell'immagine Docker.

Passaggio 3: Esegui localmente AWS IoT Greengrass

Una volta configurato il gruppo, è possibile configurare e avviare il core. Per le fasi che mostrano come eseguire questa operazione, scegli il sistema operativo:

Esecuzione di Greengrass in locale (Linux)

Esegui i seguenti comandi nel terminale del computer.

1. Crea una cartella per le risorse di sicurezza del dispositivo e sposta il certificato e le chiavi in quella cartella. Esegui i comandi seguenti. Sostituiscila *path-to-security-files* con il percorso delle risorse di sicurezza e *certificateId* sostituiscila con l'ID del certificato nei nomi dei file.

```
mkdir /tmp/certs
mv path-to-security-files/certificateId-certificate.pem.crt /tmp/certs
mv path-to-security-files/certificateId-public.pem.key /tmp/certs
mv path-to-security-files/certificateId-private.pem.key /tmp/certs
mv path-to-security-files/AmazonRootCA1.pem /tmp/certs
```

2. Crea una cartella per la configurazione del dispositivo e sposta il file di configurazione AWS IoT Greengrass Core in quella cartella. Esegui i comandi seguenti. Sostituiscilo *path-to-config-file* con il percorso del file di configurazione.

```
mkdir /tmp/config
mv path-to-config-file/config.json /tmp/config
```

3. Avvia AWS IoT Greengrass e installa i certificati e il file di configurazione nel contenitore Docker.

Sostituisci /tmp con il percorso in cui hai decompresso i certificati e il file di configurazione.

```
docker run --rm --init -it --name aws-iot-greengrass \
--entrypoint /greengrass-entrypoint.sh \
-v /tmp/certs:/greengrass/certs \
-v /tmp/config:/greengrass/config \
-p 8883:8883 \
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

L'output dovrebbe essere simile al seguente:

```
Setting up greengrass daemon
Validating hardlink/softlink protection
Waiting for up to 30s for Daemon to start

Greengrass successfully started with PID: 10
```

Esecuzione di Greengrass in locale (macOS)

Esegui i seguenti comandi nel terminale del computer.

1. Crea una cartella per le risorse di sicurezza del dispositivo e sposta il certificato e le chiavi in quella cartella. Esegui i comandi seguenti. Sostituiscila *path-to-security-files* con il

percorso delle risorse di sicurezza e *certificateId* sostituiscila con l'ID del certificato nei nomi dei file.

```
mkdir /tmp/certs
mv path-to-security-files/certificateId-certificate.pem.crt /tmp/certs
mv path-to-security-files/certificateId-public.pem.key /tmp/certs
mv path-to-security-files/certificateId-private.pem.key /tmp/certs
mv path-to-security-files/AmazonRootCA1.pem /tmp/certs
```

2. Crea una cartella per la configurazione del dispositivo e sposta il file di configurazione AWS IoT Greengrass Core in quella cartella. Esegui i comandi seguenti. Sostituiscilo *path-to-config-file* con il percorso del file di configurazione.

```
mkdir /tmp/config
mv path-to-config-file/config.json /tmp/config
```

3. Avvia AWS IoT Greengrass e installa i certificati e il file di configurazione nel contenitore Docker.

Sostituisci */tmp* con il percorso in cui hai decompresso i certificati e il file di configurazione.

```
docker run --rm --init -it --name aws-iot-greengrass \
--entrypoint /greengrass-entrypoint.sh \
-v /tmp/certs:/greengrass/certs \
-v /tmp/config:/greengrass/config \
-p 8883:8883 \
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

L'output dovrebbe essere simile al seguente:

```
Setting up greengrass daemon
Validating hardlink/softlink protection
Waiting for up to 30s for Daemon to start

Greengrass successfully started with PID: 10
```

Esecuzione di Greengrass in locale (Windows)

1. Crea una cartella per le risorse di sicurezza del dispositivo e sposta il certificato e le chiavi in quella cartella. Esegui i comandi seguenti in un prompt di comandi. Sostituiscila *path-to-*

security-files con il percorso delle risorse di sicurezza e *certificateId* sostituiscila con l'ID del certificato nei nomi dei file.

```
mkdir C:\Users\%USERNAME%\Downloads\certs
move path-to-security-files\certificateId-certificate.pem.crt C:\Users\%USERNAME%\Downloads\certs
move path-to-security-files\certificateId-public.pem.key C:\Users\%USERNAME%\Downloads\certs
move path-to-security-files\certificateId-private.pem.key C:\Users\%USERNAME%\Downloads\certs
move path-to-security-files\AmazonRootCA1.pem C:\Users\%USERNAME%\Downloads\certs
```

2. Crea una cartella per la configurazione del dispositivo e sposta il file di configurazione AWS IoT Greengrass Core in quella cartella. Esegui i comandi seguenti in un prompt di comandi. Sostituiscilo *path-to-config-file* con il percorso del file di configurazione.

```
mkdir C:\Users\%USERNAME%\Downloads\config
move path-to-config-file\config.json C:\Users\%USERNAME%\Downloads\config
```

3. Avvia AWS IoT Greengrass e installa i certificati e il file di configurazione nel contenitore Docker. Esegui i comandi seguenti nel prompt di comandi.

```
docker run --rm --init -it --name aws-iot-greengrass --entrypoint /greengrass-entrypoint.sh -v c:/Users/%USERNAME%/Downloads/certs:/greengrass/certs
-v c:/Users/%USERNAME%/Downloads/config:/greengrass/config -p 8883:8883
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

Quando Docker ti richiede di condividere l'unità C:\ con il daemon Docker, consenti di eseguire il montaggio vincolato della directory C:\ all'interno del container Docker. Per ulteriori informazioni, consulta la sezione relativa alle [unità condivise](#) nella documentazione Docker.

L'output dovrebbe essere simile al seguente:

```
Setting up greengrass daemon
Validating hardlink/softlink protection
Waiting for up to 30s for Daemon to start

Greengrass successfully started with PID: 10
```

 Note

Se il container non apre la shell e si chiude immediatamente, puoi eseguire il debug del problema mediante il montaggio vincolato dei log di runtime di Greengrass all'avvio dell'immagine. Per ulteriori informazioni, consulta [the section called “Per rendere persistenti i log di runtime Greengrass all'interno del container Docker”](#).

Fase 4: configurazione della containerizzazione "No container" per il gruppo Greengrass

Quando si esegue AWS IoT Greengrass in un contenitore Docker, tutte le funzioni Lambda devono essere eseguite senza containerizzazione. In questa fase, assicurati di aver impostato la containerizzazione predefinita per il gruppo su No container (Nessun container). È necessario eseguire questa operazione prima di distribuire il gruppo per la prima volta.

1. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).
2. Scegli il gruppo di cui desideri modificare le impostazioni.
3. Scegli la scheda Funzioni Lambda.
4. In Ambiente di runtime della funzione Lambda predefinito, scegli Modifica.
5. Nell'ambiente di runtime Modifica funzione Lambda predefinita, in Containerizzazione predefinita della funzione Lambda, modifica le impostazioni di containerizzazione.
6. Seleziona Salva.

Le modifiche diventano effettive quando il gruppo viene distribuito.

Per ulteriori informazioni, consulta [the section called “Impostazione della containerizzazione predefinita per le funzioni Lambda in un gruppo”](#).

 Note

Per impostazione predefinita, le funzioni Lambda utilizzano l'impostazione di containerizzazione del gruppo. Se sovrascrivi l'impostazione Nessun contenitore per qualsiasi funzione Lambda AWS IoT Greengrass quando è in esecuzione in un contenitore Docker, la distribuzione non riesce.

Fase 5: Implementazione delle funzioni Lambda nel contenitore Docker AWS IoT Greengrass

Puoi distribuire funzioni Lambda di lunga durata nel contenitore Greengrass Docker.

- Segui i passaggi [the section called “Modulo 3 \(parte 1\): funzioni Lambda su AWS IoT Greengrass”](#) per distribuire una funzione Hello World Lambda di lunga durata nel contenitore.

Passaggio 6: (Facoltativo) Implementa i dispositivi client che interagiscono con Greengrass in esecuzione nel contenitore Docker

Puoi anche distribuire dispositivi client con cui interagiscono AWS IoT Greengrass quando è in esecuzione in un contenitore Docker.

- Segui i passaggi indicati [the section called “Modulo 4: Interazione con i dispositivi client in un gruppo AWS IoT Greengrass”](#) per distribuire dispositivi client che si connettono al core e inviano messaggi MQTT.

Arresto del contenitore Docker AWS IoT Greengrass

Per fermare il contenitore AWS IoT Greengrass Docker, premi Ctrl+C nel terminale o nel prompt dei comandi. Questa azione invia SIGTERM al processo daemon Greengrass il compito di eliminare il processo daemon Greengrass e tutti i processi Lambda che sono stati avviati dal processo daemon. Il container Docker viene inizializzato con il processo /dev/init come PID 1 per facilitare la rimozione di eventuali processi zombie rimanenti. Per ulteriori informazioni, consulta la [pagina di riferimento per l'esecuzione del Docker](#).

AWS IoT Greengrass Risoluzione dei problemi in un contenitore Docker

Utilizza le seguenti informazioni per risolvere i problemi relativi all'esecuzione AWS IoT Greengrass in un contenitore Docker.

Errore: impossibile eseguire un accesso interattivo da un dispositivo non TTY.

Soluzione: questo errore può verificarsi quando si esegue il comando `aws ecr get-login-password`. Assicurati di aver installato la versione 2 o la AWS CLI versione 1 più recente. Ti consigliamo di utilizzare la AWS CLI versione 2. Per ulteriori informazioni, consulta [Installazione dell'AWS CLI](#) nella Guida per l'utente dell'AWS Command Line Interface .

Errore: opzioni sconosciute: -no-include-email.

Soluzione: questo errore può verificarsi quando si esegue il comando `aws ecr get-login`. Assicurati di avere installata la AWS CLI versione più recente (ad esempio, esegui: `pip install awscli --upgrade --user`). Se utilizzi Windows e hai installato l'interfaccia a riga di comando utilizzando il programma di installazione di MSI, è necessario ripetere il processo di installazione. Per ulteriori informazioni, vedere [Installazione di AWS Command Line Interface su Microsoft Windows](#) nella Guida per l'AWS Command Line Interface utente.

Attenzione: IPv4 è disattivata. Networking non funzionerà.

Soluzione: è possibile che venga visualizzato questo avviso o un messaggio simile quando si esegue AWS IoT Greengrass su un computer Linux. [Abilita l'inoltro di IPv4 rete come descritto in questo passaggio](#). AWS IoT Greengrass l'implementazione nel cloud e le comunicazioni MQTT non funzionano quando l'IPv4 inoltro non è abilitato. Per ulteriori informazioni, consulta [Configurazione di parametri kernel associati a uno spazio dei nomi \(sysctls\) durante il runtime](#) nella documentazione Docker.

Errore: un firewall sta bloccando la condivisione di file tra finestre e contenitori.

Soluzione: è possibile ricevere questo errore o un messaggio `Firewall Detected` quando viene eseguito Docker su un computer Windows. Questo errore può inoltre verificarsi se sei connesso a una rete privata virtuale (VPN, Virtual Private Network) e le impostazioni di rete impediscono il montaggio dell'unità condivisa. In tal caso, disattivare la rete VPN e riavviare il container Docker.

Errore: si è verificato un errore (`AccessDeniedException`) durante la chiamata dell' `GetAuthorizationToken` operazione: `User: arn:aws:iam: :user/ <account-id><user-name>` non è autorizzato a eseguire: `ecr: on resource: * GetAuthorizationToken`

Potresti ricevere questo errore durante l'esecuzione del `aws ecr get-login-password` comando se non disponi di autorizzazioni sufficienti per accedere a un repository Amazon ECR. Per ulteriori informazioni, consulta [Esempi di policy per i repository di Amazon ECR](#) e [Accesso a un repository Amazon ECR nella](#) Amazon ECR User Guide.

Per informazioni generali AWS IoT Greengrass sulla risoluzione dei problemi, consulta. [Risoluzione dei problemi](#)

Esecuzione del debug AWS IoT Greengrass in un contenitore Docker

Per il debug dei problemi relativi a un container Docker, puoi rendere persistenti i log di runtime Greengrass o collegare una shell interattiva al container Docker.

Per rendere persistenti i log di runtime Greengrass all'interno del container Docker

È possibile eseguire il contenitore AWS IoT Greengrass Docker dopo aver montato in bind la directory. /greengrass/ggc/var/log I log persistono anche dopo la chiusura o la rimozione del container.

Su Linux o macOS

[Interrompi qualsiasi container Docker Greengrass](#) in esecuzione sull'host, quindi esegui il comando seguente in un terminale. In questo modo viene eseguito il montaggio vincolato della directory log di Greengrass e viene avviata l'immagine Docker.

Sostituisci /tmp con il percorso in cui hai decompresso i certificati e il file di configurazione.

```
docker run --rm --init -it --name aws-iot-greengrass \
  --entrypoint /greengrass-entrypoint.sh \
  -v /tmp/certs:/greengrass/certs \
  -v /tmp/config:/greengrass/config \
  -v /tmp/log:/greengrass/ggc/var/log \
  -p 8883:8883 \
  216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

In seguito, puoi esaminare i log in /tmp/log sull'host per scoprire cosa è successo mentre Greengrass era in esecuzione all'interno del container Docker.

Su Windows

[Interrompi qualsiasi container Docker Greengrass](#) in esecuzione sull'host, quindi esegui il comando seguente in un prompt dei comandi. In questo modo viene eseguito il montaggio vincolato della directory log di Greengrass e viene avviata l'immagine Docker.

```
cd C:\Users\%USERNAME%\Downloads
mkdir log
docker run --rm --init -it --name aws-iot-greengrass --entrypoint /greengrass-
entrypoint.sh -v c:/Users/%USERNAME%/Downloads/certs:/greengrass/certs -v c:/
Users/%USERNAME%/Downloads/config:/greengrass/config -v c:/Users/%USERNAME%/
```

```
Downloads/log:/greengrass/ggc/var/log -p 8883:8883 216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

In seguito, puoi esaminare i log in `C:/Users/%USERNAME%/Downloads/log` sull'host per scoprire cosa è successo mentre Greengrass era in esecuzione all'interno del container Docker.

Per collegare una shell interattiva al container Docker

È possibile collegare una shell interattiva a un contenitore Docker in esecuzione. AWS IoT Greengrass Questo può essere utile per esaminare lo stato del container Docker Greengrass.

Su Linux o macOS

Mentre il container Docker Greengrass è in esecuzione esegui il comando seguente in un terminale separato.

```
docker exec -it $(docker ps -a -q -f "name=aws-iot-greengrass") /bin/bash
```

Su Windows

Mentre il container Docker Greengrass è in esecuzione esegui i comandi seguenti in un prompt dei comandi separato.

```
docker ps -a -q -f "name=aws-iot-greengrass"
```

Sostituisci *gg-container-id* con il `container_id` risultato del comando precedente.

```
docker exec -it gg-container-id /bin/bash
```

Accedi alle risorse locali con funzioni e connettori Lambda

Questa funzionalità è disponibile per AWS IoT Greengrass Core v1.3 e versioni successive.

Con AWS IoT Greengrass, puoi creare AWS Lambda funzioni e configurare [connettori](#) nel cloud e distribuirli sui dispositivi principali per l'esecuzione locale. Sui core Greengrass che eseguono Linux, queste funzioni e connettori Lambda distribuiti localmente possono accedere alle risorse locali fisicamente presenti sul dispositivo principale Greengrass. Ad esempio, per comunicare con dispositivi collegati tramite Modbus oppure CANbus, è possibile abilitare la funzione Lambda per accedere alla porta seriale del dispositivo principale. Per configurare l'accesso sicuro alle risorse locali, è necessario garantire la sicurezza dell'hardware fisico e del sistema operativo del dispositivo core Greengrass.

Per iniziare l'accesso alle risorse locali, consulta i seguenti tutorial:

- [Come configurare l'accesso alle risorse locali utilizzando l'interfaccia AWS a riga di comando](#)
- [Come configurare l'accesso alle risorse locali utilizzando AWS Management Console](#)

Tipi di risorse supportati

È possibile accedere a due tipi di risorse locali: risorse di volume e risorse di dispositivo.

Risorse volume

I file o le directory per il file system radice (fatta eccezione in `/sys`, `/dev` o `/var`). Ciò include:

- Cartelle o file utilizzati per leggere o scrivere informazioni tra le funzioni di Greengrass Lambda (ad esempio, `/usr/lib/python2.x/site-packages/local`)
- Cartelle o file sotto il file system `/proc` dell'host (ad esempio, `/proc/net` o `/proc/stat`). Supportato nella versione 1.6 o successiva. Per i requisiti aggiuntivi, consultare [the section called "Risorse di volume nella directory `/proc`"](#).

Tip

Per configurare le directory `/var`, `/var/run` e `/var/lib` come risorse di volume, monta prima la directory in una cartella diversa e poi configura la cartella come risorsa di volume.

Quando configuri le risorse di volume, specifichi un percorso origine e un percorso di destinazione. Il percorso di origine è il percorso assoluto della risorsa nell'host. Il percorso di destinazione è il percorso assoluto della risorsa all'interno dell'ambiente dello spazio dei nomi Lambda. Questo è il contenitore in cui viene eseguita una funzione o un connettore Greengrass Lambda. Le eventuali modifiche apportate al percorso di destinazione vengono applicate al percorso di origine nel file system host.

 Note

I file nel percorso di destinazione sono visibili solo nello spazio dei nomi Lambda. Non è possibile visualizzarli in un normale spazio dei nomi Linux.

Risorse di dispositivo

I file in `/dev`. Solo i dispositivi a caratteri o a blocchi in `/dev` sono consentiti per le risorse del dispositivo. Ciò include:

- Porte seriali utilizzate per comunicare con i dispositivi connessi tramite le porte seriali (ad esempio `/dev/ttyS0`, `/dev/ttyS1`).
- USB utilizzato per connettere periferiche USB (ad esempio, `/dev/ttyUSB0` o `/dev/bus/usb`).
- GPIOs utilizzato per sensori e attuatori tramite GPIO (ad esempio, `/dev/gpiomem`).
- GPU utilizzato per accelerare l'apprendimento automatico utilizzando sistemi di bordo GPU (ad esempio, `/dev/nvidia0`).
- Fotocamere utilizzate per acquisire immagini e video (ad esempio, `/dev/video0`).

 Note

`/dev/shm` è un'eccezione. Può essere configurato esclusivamente come risorsa di volume. Alle risorse in `/dev/shm` deve essere concessa l'autorizzazione `rw`.

AWS IoT Greengrass supporta anche i tipi di risorse utilizzati per eseguire l'inferenza dell'apprendimento automatico. Per ulteriori informazioni, consulta [Esecuzione dell'inferenza di Machine Learning](#).

Requisiti

I seguenti requisiti si applicano alla configurazione dell'accesso sicuro alle risorse locali:

- È necessario utilizzare AWS IoT Greengrass Core Software v1.3 o versione successiva. Per creare risorse per la directory /proc dell'host, è necessario utilizzare la versione 1.6 o successiva.
- La risorsa locale (inclusi i driver e le librerie necessarie) deve essere installata correttamente sul dispositivo core di Greengrass ed essere disponibile in modo coerente durante l'uso.
- L'operazione desiderata della risorsa e l'accesso alla risorsa non devono richiedere privilegi root.
- Sono disponibili solo le autorizzazioni `read` o `read and write`. Le funzioni Lambda non sono in grado di eseguire operazioni privilegiate sulle risorse.
- È necessario fornire il percorso completo della risorsa locale sul sistema operativo del dispositivo core di Greengrass.
- Un nome o ID di risorsa deve avere un massimo di 128 caratteri e deve utilizzare il modello `[a-zA-Z0-9: _- ]+`.

Risorse di volume nella directory /proc

Le seguenti considerazioni si applicano alle risorse di volume che si trovano nella directory /proc dell'host.

- È necessario utilizzare AWS IoT Greengrass Core Software v1.6 o versione successiva.
- È possibile consentire l'accesso in sola lettura per le funzioni Lambda, ma non l'accesso in lettura/scrittura. Questo livello di accesso è gestito da AWS IoT Greengrass.
- Potrebbe anche essere necessario concedere autorizzazioni di gruppo del sistema operativo per abilitare l'accesso in lettura nel file system. Ad esempio, supponiamo che la directory o il file sorgente abbiano un'autorizzazione per i file 660, indicando che solo il proprietario o un utente del gruppo hanno accesso in lettura (e scrittura). In questo caso, è necessario aggiungere le autorizzazioni del proprietario del gruppo di sistema operativo alla risorsa. Per ulteriori informazioni, consulta [the section called "Autorizzazione di accesso ai file dell'owner del gruppo"](#).
- L'ambiente host e lo spazio dei nomi Lambda contengono entrambi a /proc directory, so be sure to avoid naming conflicts when you specify the destination path. For example, if /proc is the source path, you can specify /host -proc come percorso di destinazione (o qualsiasi nome di percorso diverso da "/proc «).

Autorizzazione di accesso ai file dell'owner del gruppo

Un processo funzionale AWS IoT Greengrass Lambda normalmente viene eseguito come `ggc_user`. Tuttavia, puoi concedere autorizzazioni di accesso ai file aggiuntive al processo della funzione Lambda nella definizione della risorsa locale, come segue:

- Per aggiungere le autorizzazioni del gruppo Linux proprietario della risorsa, utilizzate il `GroupOwnerSetting#AutoAddGroupOwner` parametro o l'opzione **Aggiungi automaticamente** le autorizzazioni del file system del gruppo di sistema proprietario dell'opzione **Resource Console**.
- Per aggiungere le autorizzazioni di un altro gruppo Linux, usa il `GroupOwnerSetting#GroupOwner` parametro o l'opzione **Specificare un altro gruppo di sistema** per aggiungere le autorizzazioni del file system. Il valore `GroupOwner` viene ignorato se `GroupOwnerSetting#AutoAddGroupOwner` è `"true"`.

Un processo funzionale AWS IoT Greengrass Lambda eredita tutte le autorizzazioni del file system di `ggc_user` gruppo Linux (se aggiunto). Affinché la funzione Lambda possa accedere a una risorsa, il processo della funzione Lambda deve disporre delle autorizzazioni richieste per la risorsa. È possibile utilizzare il comando `chmod(1)` per modificare l'autorizzazione della risorsa, se necessario.

Consulta anche

- [Quote di servizio](#) per le risorse in Riferimenti generali di Amazon Web Services

Come configurare l'accesso alle risorse locali utilizzando l'interfaccia AWS a riga di comando

Questa funzionalità è disponibile per AWS IoT Greengrass Core v1.3 e versioni successive.

Per utilizzare una risorsa locale, è necessario aggiungere una definizione di risorsa alla definizione di gruppo che viene distribuita sul dispositivo core di Greengrass. La definizione di gruppo deve contenere anche una definizione di funzione Lambda in cui si concedono le autorizzazioni di accesso per le risorse locali alle funzioni Lambda. Per ulteriori informazioni, inclusi i requisiti e i vincoli, consulta [Accedi alle risorse locali con funzioni e connettori Lambda](#).

Questo tutorial descrive il processo per creare una risorsa locale e configurare l'accesso ad essa utilizzando AWS Command Line Interface (CLI). Per seguire i passaggi del tutorial, devi aver già creato un gruppo Greengrass come descritto in [Guida introduttiva con AWS IoT Greengrass](#).

Per un tutorial che utilizza il AWS Management Console, vedi. [Come configurare l'accesso alle risorse locali utilizzando AWS Management Console](#)

Creazione di risorse locali

In primo luogo, puoi usare il comando [CreateResourceDefinition](#) per creare una definizione di risorsa che specifica le risorse a cui accedere. In questo esempio, creiamo due risorse, `TestDirectory` e `TestCamera`:

```
aws greengrass create-resource-definition --cli-input-json '{
  "Name": "MyLocalVolumeResource",
  "InitialVersion": {
    "Resources": [
      {
        "Id": "data-volume",
        "Name": "TestDirectory",
        "ResourceDataContainer": {
          "LocalVolumeResourceData": {
            "SourcePath": "/src/LRAtest",
            "DestinationPath": "/dest/LRAtest",
            "GroupOwnerSetting": {
              "AutoAddGroupOwner": true,
              "GroupOwner": ""
            }
          }
        }
      },
      {
        "Id": "data-device",
        "Name": "TestCamera",
        "ResourceDataContainer": {
          "LocalDeviceResourceData": {
            "SourcePath": "/dev/video0",
            "GroupOwnerSetting": {
              "AutoAddGroupOwner": true,
              "GroupOwner": ""
            }
          }
        }
      }
    ]
  }
}
```

```

    }
  }
]
}'

```

Resources: un elenco di oggetti `Resource` nel gruppo Greengrass. Un gruppo Greengrass può avere fino a 50 risorse.

Resource#Id: identificatore univoco della risorsa. L'ID è usato per riferirsi a una risorsa nella configurazione della funzione Lambda. Limitazioni della lunghezza: 128 caratteri. Modello: `[a-zA-Z0-9:_-]+`.

Resource#Name: il nome della risorsa. Il nome della risorsa viene visualizzato nella console Greengrass. Limitazioni della lunghezza: 128 caratteri. Modello: `[a-zA-Z0-9:_-]+`.

LocalDeviceResourceData# SourcePath: Il percorso assoluto locale della risorsa del dispositivo. Il percorso di origine di una risorsa di dispositivo può riferirsi solo a un dispositivo a caratteri o a blocchi in `/dev`.

LocalVolumeResourceData# SourcePath: Il percorso assoluto locale della risorsa di volume sul dispositivo principale Greengrass. Questa posizione è esterna al [container](#) in cui viene eseguita la funzione. Il percorso di origine per un tipo di risorsa di volume non può iniziare con `/sys`.

LocalVolumeResourceData# DestinationPath: il percorso assoluto della risorsa di volume all'interno dell'ambiente Lambda. Questa posizione è interna al container in cui viene eseguita la funzione.

GroupOwnerSetting: consente di configurare privilegi di gruppo aggiuntivi per il processo Lambda. Questo campo è facoltativo. Per ulteriori informazioni, consulta [Autorizzazione di accesso ai file dell'owner del gruppo](#).

GroupOwnerSetting# AutoAddGroupOwner: Se vero, Greengrass aggiunge automaticamente il proprietario del gruppo del sistema operativo Linux specificato della risorsa ai privilegi del processo Lambda. In questo modo il processo Lambda dispone delle autorizzazioni di accesso ai file del gruppo Linux aggiunto.

GroupOwnerSetting# GroupOwner: specifica il nome del gruppo di sistemi operativi Linux i cui privilegi vengono aggiunti al processo Lambda. Questo campo è facoltativo.

Un ARN di versione della definizione di risorsa viene restituito da [CreateResourceDefinition](#). L'ARN deve essere utilizzato quando si aggiorna una definizione di gruppo.

```
{
  "LatestVersionArn": "arn:aws:greengrass:us-west-2:012345678901:/greengrass/
definition/resources/ab14d0b5-116e-4951-a322-9cde24a30373/versions/a4d9b882-
d025-4760-9cfe-9d4fada5390d",
  "Name": "MyLocalVolumeResource",
  "LastUpdatedTimestamp": "2017-11-15T01:18:42.153Z",
  "LatestVersion": "a4d9b882-d025-4760-9cfe-9d4fada5390d",
  "CreationTimestamp": "2017-11-15T01:18:42.153Z",
  "Id": "ab14d0b5-116e-4951-a322-9cde24a30373",
  "Arn": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/resources/
ab14d0b5-116e-4951-a322-9cde24a30373"
}
```

Creazione della funzione Greengrass

Dopo che le risorse sono state create, utilizzare il comando [CreateFunctionDefinition](#) per creare la funzione Greengrass e concedere alla funzione l'accesso alla risorsa:

```
aws greengrass create-function-definition --cli-input-json '{
  "Name": "MyFunctionDefinition",
  "InitialVersion": {
    "Functions": [
      {
        "Id": "greengrassLraTest",
        "FunctionArn": "arn:aws:lambda:us-
west-2:012345678901:function:lraTest:1",
        "FunctionConfiguration": {
          "Pinned": false,
          "MemorySize": 16384,
          "Timeout": 30,
          "Environment": {
            "ResourceAccessPolicies": [
              {
                "ResourceId": "data-volume",
                "Permission": "rw"
              },
              {
                "ResourceId": "data-device",
                "Permission": "ro"
              }
            ]
          },
          "AccessSysfs": true
        }
      }
    ]
  }
}
```

```

    }
  }
}
]
}
}'

```

ResourceAccessPolicies: contiene gli `resourceId` e `permission` che garantiscono alla funzione Lambda l'accesso alla risorsa. Una funzione Lambda può accedere a un massimo di 20 risorse.

ResourceAccessPolicy#Permission: specifica quali autorizzazioni ha la funzione Lambda sulla risorsa. Le opzioni disponibili sono `rw` (lettura/scrittura) o `ro` (solo lettura).

AccessSysfs: Se impostato su `true`, il processo Lambda può avere accesso in lettura alla `/sys` cartella sul dispositivo principale Greengrass. Viene utilizzato nei casi in cui la funzione Greengrass Lambda deve leggere le informazioni sul dispositivo. `/sys`

Come in precedenza, [CreateFunctionDefinition](#) restituisce un ARN di versione della definizione della funzione. L'ARN deve essere usato nella versione di definizione del gruppo.

```

{
  "LatestVersionArn": "arn:aws:greengrass:us-west-2:012345678901:/greengrass/
definition/functions/3c9b1685-634f-4592-8dfd-7ae1183c28ad/versions/37f0d50e-ef50-4faf-
b125-ade8ed12336e",
  "Name": "MyFunctionDefinition",
  "LastUpdatedTimestamp": "2017-11-22T02:28:02.325Z",
  "LatestVersion": "37f0d50e-ef50-4faf-b125-ade8ed12336e",
  "CreationTimestamp": "2017-11-22T02:28:02.325Z",
  "Id": "3c9b1685-634f-4592-8dfd-7ae1183c28ad",
  "Arn": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/
functions/3c9b1685-634f-4592-8dfd-7ae1183c28ad"
}

```

Aggiungere la funzione Lambda al gruppo

Infine, utilizzare [CreateGroupVersion](#) per aggiungere la funzione al gruppo. Per esempio:

```

aws greengrass create-group-version --group-id "b36a3aeb-3243-47ff-9fa4-7e8d98cd3cf5" \
--resource-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/
greengrass/definition/resources/db6bf40b-29d3-4c4e-9574-21ab7d74316c/versions/31d0010f-
e19a-4c4c-8098-68b79906fb87" \

```

```
--core-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/cores/adbf3475-f6f3-48e1-84d6-502f02729067/versions/297c419a-9deb-46dd-8ccc-341fc670138b" \
--function-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/functions/d1123830-da38-4c4c-a4b7-e92eec7b6d3e/versions/a2e90400-caae-4ffd-b23a-db1892a33c78" \
--subscription-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/subscriptions/7a8ef3d8-1de3-426c-9554-5b55a32fbc6/versions/470c858c-7eb3-4abd-9d48-230236bfbf6a"
```

Note

Per informazioni su come ottenere l'ID del gruppo da utilizzare con questo comando, consulta [the section called “Ottenere l'ID del gruppo”](#).

Viene restituita una nuova versione del gruppo:

```
{
  "Arn": "arn:aws:greengrass:us-west-2:012345678901:/greengrass/groups/b36a3aeb-3243-47ff-9fa4-7e8d98cd3cf5/versions/291917fb-ec54-4895-823e-27b52da25481",
  "Version": "291917fb-ec54-4895-823e-27b52da25481",
  "CreationTimestamp": "2017-11-22T01:47:22.487Z",
  "Id": "b36a3aeb-3243-47ff-9fa4-7e8d98cd3cf5"
}
```

Il tuo gruppo Greengrass ora contiene la funzione LRAtest Lambda che ha accesso a due risorse: e. TestDirectory TestCamera

In questo esempio, la funzione Lambda, `lraTest.py`, scritta in Python, scrive sulla risorsa di volume locale:

```
# Demonstrates a simple use case of local resource access.
# This Lambda function writes a file test to a volume mounted inside
# the Lambda environment under destLRAtest. Then it reads the file and
# publishes the content to the AWS IoT LRAtest topic.

import sys
import greengrasssdk
import platform
import os
```

```
import logging

# Setup logging to stdout
logger = logging.getLogger(__name__)
logging.basicConfig(stream=sys.stdout, level=logging.DEBUG)

# Create a Greengrass Core SDK client.
client = greengrasssdk.client('iot-data')
volumePath = '/dest/LRAtest'

def function_handler(event, context):
    try:
        client.publish(topic='LRA/test', payload='Sent from AWS IoT Greengrass Core.')
        volumeInfo = os.stat(volumePath)
        client.publish(topic='LRA/test', payload=str(volumeInfo))
        with open(volumePath + '/test', 'a') as output:
            output.write('Successfully write to a file.')
        with open(volumePath + '/test', 'r') as myfile:
            data = myfile.read()
        client.publish(topic='LRA/test', payload=data)
    except Exception as e:
        logger.error('Failed to publish message: ' + repr(e))
    return
```

Questi comandi sono forniti dall'API Greengrass per creare e gestire le definizioni delle risorse e le versioni di definizione delle risorse:

- [CreateResourceDefinition](#)
- [CreateResourceDefinitionVersion](#)
- [DeleteResourceDefinition](#)
- [GetResourceDefinition](#)
- [GetResourceDefinitionVersion](#)
- [ListResourceDefinitions](#)
- [ListResourceDefinitionVersions](#)
- [UpdateResourceDefinition](#)

Risoluzione dei problemi

- D: Perché la distribuzione del mio gruppo Greengrass non riesce con un errore simile:

```
group config is invalid:
  ggc_user or [ggc_group root tty] don't have ro permission on the file: /dev/tty0
```

R: Questo errore indica che il processo Lambda non ha l'autorizzazione per accedere alla risorsa specificata. La soluzione è quella di modificare l'autorizzazione dei file della risorsa in modo che Lambda possa accedervi. (Vedi [Autorizzazione di accesso ai file dell'owner del gruppo](#) per informazioni dettagliate).

- D: Quando configuro `/var/run` come risorsa di volume, perché la funzione Lambda non si avvia con un messaggio di errore nel runtime.log:

```
[ERROR]-container_process.go:39,Runtime execution error: unable to start lambda
container.
container_linux.go:259: starting container process caused "process_linux.go:345:
container init caused \"rootfs_linux.go:62: mounting \"/var/run\" to rootfs \"/
greengrass/ggc/packages/1.3.0/rootfs_sys\" at \"/greengrass/ggc/packages/1.3.0/
rootfs_sys/run\"
caused \"invalid argument\""
```

R: AWS IoT Greengrass core attualmente non supporta la configurazione di `/var/lib` come risorse di `/var` `/var/run` volume. Una soluzione è quella di montare prima `/var`, `/var/run` o `/var/lib` in una cartella diversa e quindi configurare la cartella come risorsa di volume.

- D: Quando configuro `/dev/shm` come risorsa di volume con autorizzazioni di sola lettura, perché la funzione Lambda non si avvia con un errore nel runtime.log:

```
[ERROR]-container_process.go:39,Runtime execution error: unable to start lambda
container.
container_linux.go:259: starting container process caused "process_linux.go:345:
container init caused \"rootfs_linux.go:62: mounting \"/dev/shm\" to rootfs \"/
greengrass/ggc/packages/1.3.0/rootfs_sys\" at \"/greengrass/ggc/packages/1.3.0/
rootfs_sys/dev/shm\"
caused \"operation not permitted\""
```

R: `/dev/shm` può essere configurato solo in lettura/scrittura Cambia l'autorizzazione a livello di risorsa per `rw` per risolvere il problema.

Come configurare l'accesso alle risorse locali utilizzando AWS Management Console

Questa funzionalità è disponibile per AWS IoT Greengrass Core v1.3 e versioni successive.

È possibile configurare le funzioni Lambda per accedere in modo sicuro alle risorse locali sul dispositivo principale Greengrass host. Per risorse locali si intendono bus e periferiche che sono fisicamente presenti nell'host o volumi di file system nel sistema operativo dell'host. Per ulteriori informazioni, inclusi i requisiti e i vincoli, consulta [Accedi alle risorse locali con funzioni e connettori Lambda](#).

Questo tutorial descrive come utilizzare per configurare l' AWS Management Console accesso alle risorse locali presenti su un AWS IoT Greengrass dispositivo principale. Include le seguenti fasi di alto livello:

1. [Creare un pacchetto di distribuzione della funzione Lambda](#)
2. [Creare e pubblicare una funzione Lambda](#)
3. [Aggiungere la funzione Lambda al gruppo](#)
4. [Aggiungere una risorsa locale al gruppo](#)
5. [Aggiunta di sottoscrizioni al gruppo](#)
6. [Distribuzione del gruppo.](#)

Per un tutorial che utilizza il AWS Command Line Interface, vedi [Come configurare l'accesso alle risorse locali utilizzando l'interfaccia AWS a riga di comando](#).

Prerequisiti

Per completare questo tutorial, è necessario quanto segue:

- Un gruppo Greengrass e un core Greengrass (v1.3 o successivo). Per creare un gruppo o un core Greengrass, consulta [Guida introduttiva con AWS IoT Greengrass](#).
- Le seguenti directory sul dispositivo core Greengrass:
 - /src/ LRAtest
 - /dest/ LRAtest

Il gruppo proprietario di queste directory deve disporre dell'accesso in lettura e scrittura alle directory. Per concedere l'accesso potresti utilizzare il seguente comando:


```
sudo chmod 0775 /src/LRAtest
```

Fase 1: Creare un pacchetto di distribuzione della funzione Lambda

In questo passaggio, crei un pacchetto di distribuzione della funzione Lambda, che è un file ZIP che contiene il codice e le dipendenze della funzione. Inoltre, scarichi il AWS IoT Greengrass Core SDK da includere nel pacchetto come dipendenza.

1. Sul computer, copia il seguente script Python in un file locale denominato `lraTest.py`. Questa è la logica dell'app per la funzione Lambda.

```
# Demonstrates a simple use case of local resource access.
# This Lambda function writes a file test to a volume mounted inside
# the Lambda environment under destLRAtest. Then it reads the file and
# publishes the content to the AWS IoT LRAtest topic.

import sys
import greengrasssdk
import platform
import os
import logging

# Setup logging to stdout
logger = logging.getLogger(__name__)
logging.basicConfig(stream=sys.stdout, level=logging.DEBUG)

# Create a Greengrass Core SDK client.
client = greengrasssdk.client('iot-data')
volumePath = '/dest/LRAtest'

def function_handler(event, context):
    try:
        client.publish(topic='LRA/test', payload='Sent from AWS IoT Greengrass
Core.')
        volumeInfo = os.stat(volumePath)
        client.publish(topic='LRA/test', payload=str(volumeInfo))
        with open(volumePath + '/test', 'a') as output:
            output.write('Successfully write to a file.')
        with open(volumePath + '/test', 'r') as myfile:
            data = myfile.read()
```

```
client.publish(topic='LRA/test', payload=data)
except Exception as e:
    logger.error('Failed to publish message: ' + repr(e))
return
```

2. Dalla pagina dei download di [AWS IoT Greengrass Core SDK](#), scarica AWS IoT Greengrass Core SDK per Python sul tuo computer.
3. Decomprimere il pacchetto scaricato per ottenere l'SDK. Il kit SDK è la cartella greengrasssdk.
4. Comprimi i seguenti elementi in un file denominato `lraTestLambda.zip`:
 - `lraTest.py`. Logica delle app.
 - `greengrasssdk`. Libreria richiesta per tutte le funzioni Python Lambda.

Il `lraTestLambda.zip` file è il pacchetto di distribuzione della funzione Lambda. Ora sei pronto per creare una funzione Lambda e caricare il pacchetto di distribuzione.

Fase 2: Creare e pubblicare una funzione Lambda

In questo passaggio, si utilizza la AWS Lambda console per creare una funzione Lambda e configurarla per utilizzare il pacchetto di distribuzione. In seguito, pubblicherai una versione della funzione e creerai un alias.

Innanzitutto, crea la funzione Lambda.

1. In AWS Management Console, scegli Servizi e apri la AWS Lambda console.
2. Scegli Funzioni.
3. Scegli Crea funzione e poi scegli Autore da zero.
4. Nella sezione Basic information (Informazioni di base), specifica i seguenti valori:
 - a. Nel campo Function name (Nome funzione), immettere **TestLRA**.
 - b. In Runtime, scegliere Python 3.7.
 - c. Per le autorizzazioni, mantieni l'impostazione predefinita. Questo crea un ruolo di esecuzione che concede le autorizzazioni Lambda di base. Questo ruolo non viene utilizzato da. AWS IoT Greengrass
5. Scegli Crea funzione.

Basic information

Function name
Enter a name that describes the purpose of your function.

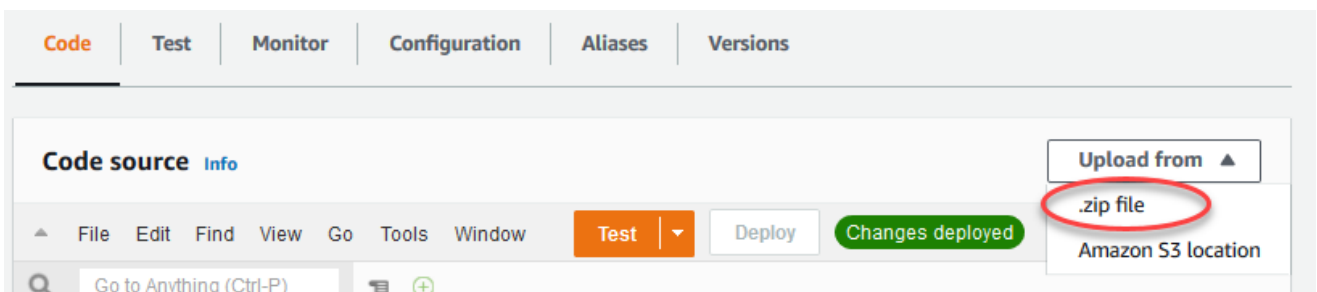
Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime [Info](#)
Choose the language to use to write your function.

Permissions [Info](#)
Lambda will create an execution role with permission to upload logs to Amazon CloudWatch Logs. You can configure and modify permissions further when you add triggers.
► [Choose or create an execution role](#)

[Cancel](#) [Create function](#)

6. Carica il pacchetto di distribuzione della funzione Lambda e registra il gestore.
 - a. Nella scheda Codice, in Codice sorgente, scegli Carica da. Dal menu a discesa, scegli un file.zip.



- b. Scegli Carica, quindi scegli il pacchetto di `lraTestLambda.zip` distribuzione. Quindi, scegliere Save (Salva).
 - c. Nella scheda Codice della funzione, in Impostazioni di runtime, scegliete Modifica, quindi immettete i seguenti valori.
 - In Runtime, scegliere Python 3.7.
 - Per Handler, immettere `lraTest.function_handler`.
 - d. Seleziona Salva.

Note

Il pulsante Test sulla AWS Lambda console non funziona con questa funzione. Il AWS IoT Greengrass Core SDK non contiene moduli necessari per eseguire le

funzioni Greengrass Lambda in modo indipendente nella console. AWS Lambda Questi moduli (ad esempio, `greengrass_common`) vengono forniti alle funzioni dopo essere stati distribuiti nel core Greengrass.

Successivamente, pubblica la prima versione della tua funzione Lambda. Quindi, creare un [alias per la versione](#).

I gruppi Greengrass possono fare riferimento a una funzione Lambda tramite alias (consigliato) o per versione. L'utilizzo di un alias semplifica la gestione degli aggiornamenti del codice perché non è necessario modificare la tabella di sottoscrizione o la definizione del gruppo quando il codice della funzione viene aggiornato. Invece, è sufficiente indirizzare l'alias alla nuova versione della funzione.

7. Da Actions (Operazioni), seleziona Publish new version (Pubblica nuova versione).
8. Per Version description (Descrizione versione), immettere **First version**, quindi scegliere Publish (Pubblica).
9. Nella pagina di configurazione TestLRA: 1, in Actions (Operazioni), scegli Create alias (Crea alias).
10. Nella pagina Crea alias, per Nome, inserisci. **test** Per Version (Versione), immetti 1.

 Note

AWS IoT Greengrass non supporta gli alias Lambda per le versioni \$LATEST.

11. Scegli Create (Crea) .

An alias is a pointer to one or two versions. Choose each version that you want the alias to point to.

Name*

Description

Version*

You can shift traffic between two versions, based on weights (%) that you assign. Click [here](#) to learn more.

Additional version

Cancel

Create

Ora puoi aggiungere la funzione Lambda al tuo gruppo Greengrass.

Fase 3: Aggiungere la funzione Lambda al gruppo Greengrass

In questa fase, aggiungerai la funzione al gruppo e configurerai il ciclo di vita della funzione.

Innanzitutto, aggiungi la funzione Lambda al tuo gruppo Greengrass.

1. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).
2. Scegliete il gruppo Greengrass in cui desiderate aggiungere la funzione Lambda.
3. Nella pagina di configurazione del gruppo, scegli la scheda Funzioni Lambda.
4. Nella sezione Le mie funzioni Lambda, scegli Aggiungi.
5. Nella pagina Aggiungi funzione Lambda, scegli la funzione Lambda. Seleziona **TestLRA**.
6. Scegli la versione della funzione Lambda.
7. Nella sezione Configurazione della funzione Lambda, seleziona Utente e gruppo di sistema e Containerizzazione della funzione Lambda.

Quindi, configura il ciclo di vita della funzione Lambda.

8. Per Timeout, scegli 30 seconds (30 secondi).

 Important

Le funzioni Lambda che utilizzano risorse locali (come descritto in questa procedura) devono essere eseguite in un contenitore Greengrass. In caso contrario, il tentativo di distribuire la funzione avrà esito negativo. Per ulteriori informazioni, consulta [Containerizzazione](#).

9. Nella parte inferiore della pagina, scegli Aggiungi funzione Lambda.

Fase 4: aggiungere una risorsa locale al gruppo Greengrass

In questa fase, aggiungerai una risorsa di volume locale al gruppo Greengrass e concederai alla funzione l'accesso in lettura e scrittura alla risorsa. Una risorsa locale dispone di un ambito a livello di gruppo. Puoi concedere le autorizzazioni per qualsiasi funzione Lambda del gruppo per accedere alla risorsa.

1. Nella pagina di configurazione del gruppo, scegli la scheda Risorse.
2. Nella sezione Risorse locali, scegli Aggiungi.
3. Nella pagina Aggiungi una risorsa locale, utilizza i seguenti valori.
 - a. Per Resource Name (Nome risorsa) immetti **testDirectory**.
 - b. Per Resource type (Tipo di risorsa), scegli Volume.
 - c. Per Percorso del dispositivo locale, immettere **/src/LRAtest**. Questo percorso deve esistere sul sistema operativo dell'host.

Il percorso del dispositivo locale è il percorso assoluto locale della risorsa sul file system del dispositivo principale. Questa posizione è esterna al [container](#) in cui viene eseguita la funzione. Il percorso non può iniziare con `/sys`.

- d. Per Destination path (Percorso di destinazione), immetti **/dest/LRAtest**. Questo percorso deve esistere sul sistema operativo dell'host.

Il percorso di destinazione è il percorso assoluto della risorsa nello spazio dei nomi Lambda. Questa posizione è interna al container in cui viene eseguita la funzione.

- e. In Proprietario del gruppo di sistema e autorizzazione di accesso ai file, seleziona Aggiungi automaticamente le autorizzazioni del file system del gruppo di sistema proprietario della risorsa.

L'opzione System group owner and file access permission (proprietario del gruppo di sistema e autorizzazione all'accesso ai file) consente di concedere ulteriori autorizzazioni di accesso ai file al processo Lambda. Per ulteriori informazioni, consulta [Autorizzazione di accesso ai file dell'owner del gruppo](#).

4. Scegliere Add resource (Aggiungi risorsa). Nella pagina Resources (Risorse) viene visualizzata la nuova risorsa testDirectory.

Fase 5: aggiungere sottoscrizioni al gruppo Greengrass

In questa fase, aggiungerai due sottoscrizioni al gruppo Greengrass. Questi abbonamenti consentono la comunicazione bidirezionale tra la funzione Lambda e AWS IoT

Innanzitutto, crea un abbonamento per la funzione Lambda a cui inviare messaggi. AWS IoT

1. Nella pagina di configurazione del gruppo, scegli la scheda Abbonamenti.
2. Scegli Aggiungi.
3. Nella pagina Crea un abbonamento, configura l'origine e la destinazione come segue:
 - a. Per Tipo di sorgente, scegli la funzione Lambda, quindi scegli TestLRA.
 - b. Per il tipo di Target, scegli Servizio, quindi scegli IoT Cloud.
 - c. Per il filtro per argomento **LRA/test**, inserisci, quindi scegli Crea abbonamento.
4. Nella pagina Subscriptions (Sottoscrizioni) viene visualizzata la nuova sottoscrizione.

Quindi, configura un abbonamento che richiami la funzione da AWS IoT

5. Nella pagina Abbonamenti, scegli Aggiungi abbonamento.
6. Nella pagina Select your source and target (Seleziona origine e destinazione), configura l'origine e la destinazione come indicato di seguito:
 - a. Per Tipo di sorgente, scegli la funzione Lambda, quindi scegli IoT Cloud.
 - b. Per Tipo di destinazione, scegli Servizio, quindi scegli TestLRA.

- c. Scegli Next (Successivo).
7. Nella pagina Filter your data with a topic (Filtra i dati con un argomento), in Topic filter (Filtro argomento), immetti **invoke/LRAFunction**, quindi scegliere Next (Avanti).
8. Scegli Fine. Nella pagina Subscriptions (Sottoscrizioni) vengono visualizzate le sottoscrizioni.

Passaggio 6: Distribuire il gruppo AWS IoT Greengrass

In questa fase, distribuirai la versione corrente della definizione del gruppo.

1. Assicurati che il AWS IoT Greengrass core sia in funzione. Esegui i seguenti comandi nel terminale di Raspberry Pi in base alle esigenze.
 - a. Per controllare se il daemon è in esecuzione:

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

Se l'output contiene una voce root per /greengrass/ggc/packages/1.11.6/bin/daemon, allora il daemon è in esecuzione.

Note

La versione indicata nel percorso dipende dalla versione del software AWS IoT Greengrass Core installata sul dispositivo principale.

- b. Per avviare il demone:

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

2. Nella pagina di configurazione del gruppo, scegli Deploy.

Note

La distribuzione fallisce se si esegue la funzione Lambda senza containerizzazione e si tenta di accedere alle risorse locali collegate.

3. Se richiesto, nella scheda Funzione Lambda, in Funzioni Lambda di sistema, selezionare Rilevatore IP, quindi Modifica e quindi Rileva automaticamente.

Questo consente ai dispositivi di acquisire automaticamente informazioni di base sulla connettività, come, ad esempio indirizzo IP, DNS e numero della porta. Il rilevamento automatico è consigliato, ma supporta AWS IoT Greengrass anche gli endpoint specificati manualmente. Ti viene chiesto il metodo di individuazione solo la prima volta che il gruppo viene distribuito.

Note

Se richiesto, concedi l'autorizzazione a creare il ruolo di [servizio Greengrass](#) e associarlo al Account AWS tuo ruolo attuale. Regione AWS Questo ruolo consente di accedere AWS IoT Greengrass alle tue risorse nei AWS servizi.

Nella pagina Deployments (Distribuzioni) vengono visualizzati il timestamp della distribuzione, l'ID versione e lo stato. Una volta completata, lo stato della distribuzione è Completato.

Per la risoluzione dei problemi, consultare [Risoluzione dei problemi](#).

Test dell'accesso alle risorse locali

Ora puoi verificare se l'accesso alla risorsa locale è configurato correttamente. Per eseguire il test, devi abbonarti all'argomento LRA/test e pubblicare nell'argomento invoke/LRAFunction. Il test ha esito positivo se la funzione Lambda invia il payload previsto a. AWS IoT

1. Dal menu di navigazione della AWS IoT console, in Test, scegliete MQTT test client.
2. In Sottoscrivi a un argomento, per Filtro argomento, inserisci **LRA/test**.
3. In Informazioni aggiuntive, per la visualizzazione del payload MQTT, seleziona Visualizza i payload come stringhe.
4. Scegliere Subscribe (Effettua sottoscrizione). La tua funzione Lambda viene pubblicata nell'argomento LRA/Test.

Subscribe to a topic | **Publish to a topic**

Topic filter [Info](#)
The topic filter describes the topic(s) to which you want to subscribe. The topic filter can include MQTT wildcard characters.

▼ **Additional configuration**

Number of messages to keep
The MQTT test client keeps this many of the most recent messages published to a topic that matches this topic filter.

Quality of service
When subscribing to a topic, quality of service 0 will be chosen by default.

☒ Quality of Service 0 - Message will be delivered at most once

☐ Quality of Service 1 - Message will be delivered at least once

MQTT payload display

☐ Auto-format JSON payloads (improves readability)

☒ Display payloads as strings (more accurate)

☐ Display raw payloads (displays binary data as hexadecimal values)

Subscribe

5. In **Pubblica su un argomento**, inserisci il nome dell'argomento **invoke/LRAFunction**, quindi scegli **Pubblica** per richiamare la funzione Lambda. Il test ha esisto positivo se nella pagina vengono visualizzati tre payload del messaggio della funzione.

Subscribe to a topic

Publish to a topic

Topic name

The topic name identifies the message. The message payload will be published to this topic with a Quality of Service (QoS) of 0.

Q

invoke/LRAFunction

X

Message payload

```
{  
  "message": "Hello from AWS IoT console"  
}
```

► Additional configuration

Publish

Subscriptions

lra/test

▼

X

lra/test

▼

May 03, 2021, 12:09:18 (UTC-0400)

Successfully write to a file.

lra/test

▼

May 03, 2021, 12:09:06 (UTC-0400)

```
posix.stat_result(st_mode=16893, st_ino=171142L, st_dev=45831L, st_nlink=2, st_uid=0, st  
_gid=119, st_size=4096L, st_atime=1620054520, st_mtime=1620058120, st_ctime=1620058120)
```

lra/test

▼

May 03, 2021, 12:09:04 (UTC-0400)

Sent from Greengrass Core.

Pause

Clear

Export

Edit

Il file di test creato dalla funzione Lambda si trova nella `/src/LRAtest` directory del dispositivo principale Greengrass. Sebbene la funzione Lambda scriva su un file nella `/dest/LRAtest` directory, tale file è visibile solo nello spazio dei nomi Lambda. Non è possibile visualizzarlo in un normale spazio dei nomi Linux. Le eventuali modifiche apportate al percorso di destinazione vengono applicate al percorso di origine nel file system.

Per la risoluzione dei problemi, consultare [Risoluzione dei problemi](#).

Esecuzione dell'inferenza di Machine Learning

Questa funzionalità è disponibile per AWS IoT Greengrass Core v1.6 o versioni successive.

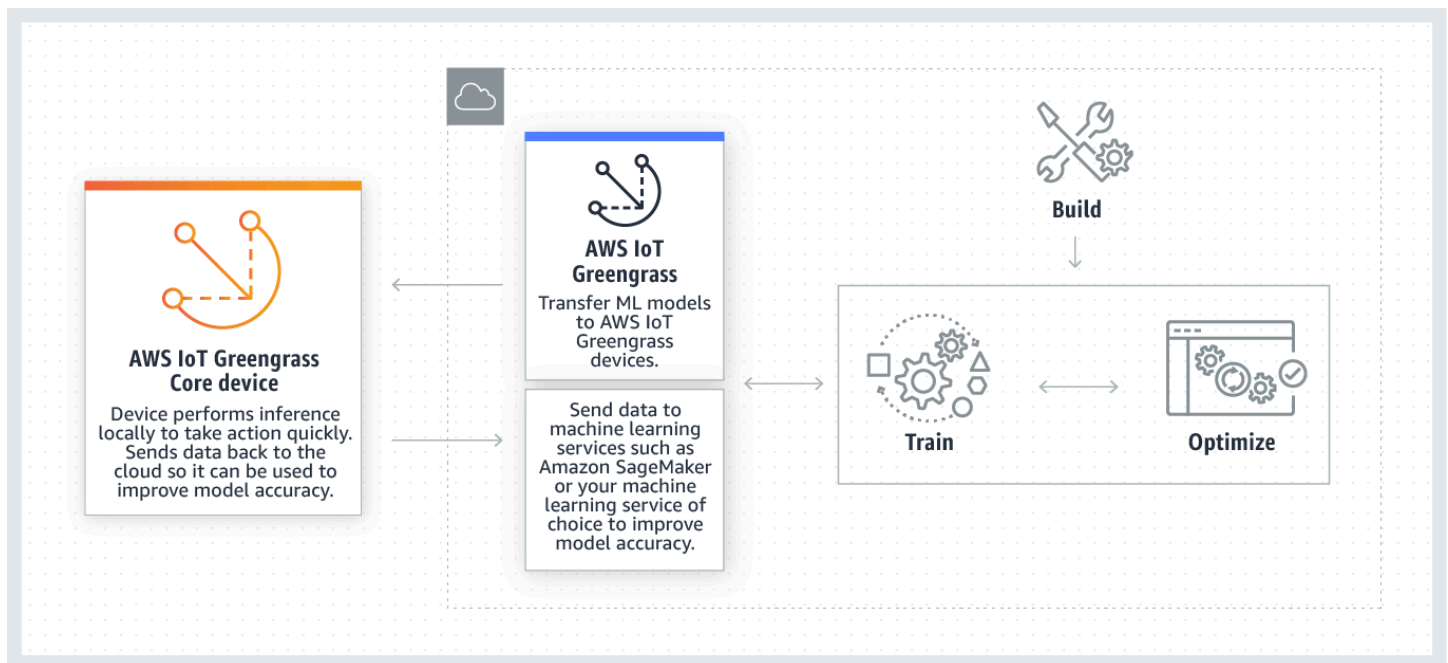
Con AWS IoT Greengrass, puoi eseguire inferenze di machine learning (ML) all'edge su dati generati localmente utilizzando modelli addestrati sul cloud. In questo modo si beneficia della bassa latenza e dei risparmi sui costi dell'inferenza locale e si sfrutta la potenza del cloud computing per modelli di formazione ed elaborazioni complesse.

Per iniziare eseguendo un'inferenza locale, consulta [the section called “Come configurare l'inferenza di Machine Learning”](#).

Come funziona AWS IoT Greengrass l'inferenza ML

Puoi addestrare i tuoi modelli di inferenza ovunque, distribuirli localmente come risorse di machine learning in un gruppo Greengrass e quindi accedervi dalle funzioni di Greengrass Lambda. Ad esempio, puoi creare e addestrare modelli di deep learning nell'[SageMaker intelligenza artificiale](#) e distribuirli nel tuo core Greengrass. Quindi, le funzioni Lambda possono utilizzare i modelli locali per eseguire inferenze sui dispositivi connessi e inviare nuovi dati di addestramento al cloud.

Il diagramma seguente mostra il flusso di lavoro di inferenza AWS IoT Greengrass ML.



AWS IoT Greengrass L'inferenza ML semplifica ogni fase del flusso di lavoro ML, tra cui:

- Sviluppo e distribuzione di prototipi di framework di Machine Learning.
- Accesso a modelli qualificati per il cloud e distribuzione sui dispositivi core Greengrass.
- [Creazione di app di inferenza in grado di accedere agli acceleratori hardware \(come GPUs e FPGAs\) come risorse locali.](#)

Risorse di Machine Learning

Le risorse di machine learning rappresentano modelli di inferenza addestrati sul cloud che vengono implementati su un core. AWS IoT Greengrass Per distribuire risorse di machine learning, aggiungete innanzitutto le risorse a un gruppo Greengrass, quindi definite in che modo le funzioni Lambda del gruppo possono accedervi. Durante la distribuzione di gruppo, AWS IoT Greengrass recupera i pacchetti del modello sorgente dal cloud e li estrae nelle directory all'interno dello spazio dei nomi di runtime Lambda. Quindi, le funzioni Greengrass Lambda utilizzano i modelli distribuiti localmente per eseguire l'inferenza.

Per aggiornare un modello distribuito localmente, aggiornare prima il modello di origine (nel cloud) che corrisponde alla risorsa di Machine Learning, quindi distribuire il gruppo. Durante la distribuzione, AWS IoT Greengrass controlla l'origine delle modifiche. Se vengono rilevate modifiche, AWS IoT Greengrass aggiorna il modello locale.

Origini di modello supportate

AWS IoT Greengrass supporta sorgenti di modelli SageMaker AI e Amazon S3 per risorse di apprendimento automatico.

I seguenti requisiti si applicano alle origini di modello:

- I bucket S3 che memorizzano le sorgenti del modello SageMaker AI e Amazon S3 non devono essere crittografati utilizzando SSE-C. Per i bucket che utilizzano la crittografia lato server, l'inferenza AWS IoT Greengrass ML attualmente supporta solo le opzioni di crittografia SSE-S3 o SSE-KMS. Per ulteriori informazioni sulle opzioni di crittografia lato server, consulta [Protezione dei dati utilizzando la crittografia lato server nella Guida per l'utente di Amazon Simple Storage Service](#).
- I nomi dei bucket S3 che memorizzano le fonti dei modelli SageMaker AI e Amazon S3 non devono includere periodi (). . Per ulteriori informazioni, consulta la regola sull'utilizzo di bucket in stile host virtuale con SSL nelle [Regole per la denominazione dei bucket nella Amazon Simple Storage Service User Guide](#).

- [Il Regione AWS supporto a livello di servizio deve essere disponibile sia per l'IA che per l'intelligenza artificiale. AWS IoT Greengrass SageMaker](#) Attualmente, AWS IoT Greengrass supporta modelli di SageMaker intelligenza artificiale nelle seguenti regioni:
 - Stati Uniti orientali (Ohio)
 - Stati Uniti orientali (Virginia settentrionale)
 - US West (Oregon)
 - Asia Pacifico (Mumbai)
 - Asia Pacifico (Seoul)
 - Asia Pacifico (Singapore)
 - Asia Pacifico (Sydney)
 - Asia Pacifico (Tokyo)
 - Europa (Francoforte)
 - Europa (Irlanda)
 - Europa (Londra)
- AWS IoT Greengrass deve avere l'readautorizzazione alla fonte del modello, come descritto nelle sezioni seguenti.

SageMaker INTELLIGENZA ARTIFICIALE

AWS IoT Greengrass supporta modelli che vengono salvati come lavori di formazione sull' SageMaker intelligenza artificiale. SageMaker L'intelligenza artificiale è un servizio di machine learning completamente gestito che puoi utilizzare per creare e addestrare modelli utilizzando algoritmi integrati o personalizzati. Per ulteriori informazioni, consulta [Cos'è l' SageMaker IA?](#) nella Guida per sviluppatori di SageMaker intelligenza artificiale.

Se hai configurato il tuo ambiente di SageMaker intelligenza artificiale [creando un bucket](#) il cui nome contiene `sagemaker`, allora AWS IoT Greengrass disponi delle autorizzazioni sufficienti per accedere ai tuoi lavori di formazione sull' SageMaker intelligenza artificiale. La policy gestita `AWSGreengrassResourceAccessRolePolicy` consente di accedere al bucket il cui nome contiene la stringa `sagemaker`. Questa policy è collegata al [ruolo di servizio Greengrass](#).

Altrimenti, devi concedere l' AWS IoT Greengrass readautorizzazione al bucket in cui è archiviato il tuo lavoro di formazione. Per effettuare questa operazione, inserisci la seguente policy inline nel ruolo di servizio. Puoi elencare più ARNs bucket.

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

Amazon S3

AWS IoT Greengrass supporta modelli archiviati in Amazon S3 come file `tar.gz` o `zip`.

Per consentire l'accesso AWS IoT Greengrass ai modelli archiviati nei bucket Amazon S3, devi concedere l'AWS IoT Greengrass read autorizzazione ad accedere ai bucket effettuando una delle seguenti operazioni:

- Archivia il modello in un bucket il cui nome contiene `greengrass`.

La policy gestita `AWSGreengrassResourceAccessRolePolicy` consente di accedere al bucket il cui nome contiene la stringa `greengrass`. Questa policy è collegata al [ruolo di servizio Greengrass](#).

- Incorpora una policy inline nel ruolo di servizio Greengrass.

Se il nome del bucket non contiene `greengrass`, aggiungi le seguenti policy inline al ruolo di servizio. Puoi elencare più bucket. ARNs

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ]
    }
  ]
}
```



```
    ],  
    "Resource": [  
        "arn:aws:s3:::amzn-s3-demo-bucket"  
    ]  
  }  
]  
}
```

Per ulteriori informazioni, consulta [Incorporare le politiche in linea](#) nella Guida per l'utente IAM.

Requisiti

I seguenti requisiti si applicano alla creazione e all'utilizzo di risorse di Machine Learning:

- È necessario utilizzare AWS IoT Greengrass Core v1.6 o versione successiva.
- Le funzioni Lambda definite dall'utente possono `read` eseguire `read` and `write` operazioni sulla risorsa. Le autorizzazioni per altre operazioni non sono disponibili. La modalità di containerizzazione delle funzioni Lambda affiliate determina il modo in cui si impostano le autorizzazioni di accesso. Per ulteriori informazioni, consulta [the section called “Accesso alle risorse di Machine Learning”](#).
- È necessario fornire il percorso completo della risorsa sul sistema operativo del dispositivo core.
- Un nome o ID di risorsa deve avere un massimo di 128 caratteri e deve utilizzare il modello `[a-zA-Z0-9:_-]+`.

Runtime e librerie per inferenza ML

È possibile utilizzare i seguenti runtime e librerie ML con AWS IoT Greengrass

- [Runtime di apprendimento approfondito di Amazon SageMaker Neo](#)
- Apache MXNet
- TensorFlow

Questi runtime e librerie possono essere installati su piattaforme NVIDIA Jetson TX2, Intel Atom e Raspberry Pi. Per informazioni di download, consulta [the section called “Runtime e librerie di Machine Learning supportati”](#). Puoi installarli direttamente sul dispositivo principale.

Assicurati di leggere le seguenti informazioni sulla compatibilità e sulle limitazioni.

SageMaker Runtime di deep learning AI Neo

Puoi utilizzare il runtime di deep learning SageMaker AI Neo per eseguire inferenze con modelli di machine learning ottimizzati sui tuoi AWS IoT Greengrass dispositivi. Questi modelli sono ottimizzati utilizzando il compilatore di deep learning SageMaker AI Neo per migliorare la velocità di previsione dell'inferenza dell'apprendimento automatico. Per ulteriori informazioni sull'ottimizzazione dei modelli nell' SageMaker intelligenza artificiale, consulta la documentazione di [SageMaker AI Neo](#).

Note

Attualmente, puoi ottimizzare i modelli di machine learning utilizzando il compilatore di deep learning Neo solo in specifiche regioni di Amazon Web Services. Tuttavia, puoi utilizzare il runtime di deep learning Neo con modelli ottimizzati in ognuna delle Regione AWS quali è supportato il AWS IoT Greengrass core. Per informazioni, consulta [Come configurare l'inferenza Machine Learning ottimizzata](#).

MXNet controllo delle versioni

Attualmente Apache MXNet non garantisce la compatibilità con le versioni future, quindi i modelli addestrati utilizzando versioni successive del framework potrebbero non funzionare correttamente nelle versioni precedenti del framework. Per evitare conflitti tra le fasi di formazione del modello e di gestione del modello e per fornire un' end-to-end esperienza coerente, utilizzate la stessa versione del MXNet framework in entrambe le fasi.

MXNet su Raspberry Pi

Le funzioni Greengrass Lambda che accedono ai MXNet modelli locali devono impostare la seguente variabile di ambiente:

```
MXNET_ENGINE_TYPE=NativeEngine
```

È possibile impostare la variabile d'ambiente nel codice funzione o aggiungerla alla configurazione specifica del gruppo della funzione. Per un esempio in cui viene aggiunta come impostazione di configurazione, vedi questa [fase](#).

 Note

Per un uso generale del MXNet framework, come l'esecuzione di un esempio di codice di terze parti, la variabile di ambiente deve essere configurata su Raspberry Pi.

TensorFlow limitazioni relative al servizio dei modelli su Raspberry Pi

I seguenti consigli per migliorare i risultati di inferenza si basano sui nostri test con le librerie Arm TensorFlow a 32 bit sulla piattaforma Raspberry Pi. Queste raccomandazioni sono destinate a utenti esperti solo per riferimento, senza garanzie di alcun tipo.

- Modelli che sono addestrati utilizzando il formato [Checkpoint](#) devono essere "congelati" nel formato del buffer di protocollo prima di essere messi a disposizione. Per un esempio, consulta la libreria di modelli di classificazione delle [TensorFlowimmagini -Slim](#).
- Non utilizzare le librerie TF-Estimator e TF-Slim in codice di training o di inferenza. Utilizzare invece il pattern di caricamento del modello di file .pb che viene mostrato nell'esempio seguente.

```
graph = tf.Graph()
graph_def = tf.GraphDef()
graph_def.ParseFromString(pb_file.read())
with graph.as_default():
    tf.import_graph_def(graph_def)
```

 Note

Per ulteriori informazioni sulle piattaforme supportate per TensorFlow, vedete [Installazione TensorFlow](#) nella TensorFlow documentazione.

Accedi alle risorse di machine learning dalle funzioni Lambda

Le funzioni Lambda definite dall'utente possono accedere alle risorse di machine learning per eseguire inferenze locali sul core. AWS IoT Greengrass Una risorsa di machine learning è costituita dal modello con training e da altri artefatti che vengono scaricati nel dispositivo core.

Per consentire a una funzione Lambda di accedere a una risorsa di machine learning centrale, è necessario collegare la risorsa alla funzione Lambda e definire le autorizzazioni di accesso. La [modalità di containerizzazione](#) della funzione Lambda affiliata (o allegata) determina come eseguire questa operazione.

Autorizzazioni di accesso per risorse di Machine Learning

A partire da AWS IoT Greengrass Core v1.10.0, puoi definire il proprietario della risorsa per una risorsa di machine learning. Il proprietario della risorsa rappresenta il gruppo del sistema operativo e le autorizzazioni AWS IoT Greengrass utilizzati per scaricare gli artefatti delle risorse. Se il proprietario di una risorsa non è definito, gli artefatti della risorsa scaricati sono accessibili solo al root.

- Se le funzioni Lambda non containerizzate accedono a una risorsa di machine learning, è necessario definire un proprietario della risorsa perché non esiste alcun controllo delle autorizzazioni da parte del contenitore. Le funzioni Lambda non containerizzate possono ereditare le autorizzazioni del proprietario della risorsa e utilizzarle per accedere alla risorsa.
- Se solo le funzioni Lambda containerizzate accedono alla risorsa, ti consigliamo di utilizzare le autorizzazioni a livello di funzione invece di definire un proprietario della risorsa.

Proprietà del proprietario delle risorse

Un proprietario delle risorse specifica un proprietario del gruppo e le autorizzazioni del proprietario del gruppo.

Proprietario del gruppo. L'ID del gruppo (GID) di un gruppo OS Linux esistente sul dispositivo core. Le autorizzazioni del gruppo vengono aggiunte al processo Lambda. In particolare, il GID viene aggiunto al gruppo supplementare IDs della funzione Lambda.

Se una funzione Lambda del gruppo Greengrass è configurata per essere [eseguita come](#) lo stesso gruppo di sistemi operativi del proprietario della risorsa per una risorsa di machine learning, la risorsa deve essere collegata alla funzione Lambda. In caso contrario, la distribuzione non riesce perché questa configurazione fornisce autorizzazioni implicite che la funzione Lambda può utilizzare per accedere alla risorsa senza autorizzazione. AWS IoT Greengrass Il controllo

di convalida della distribuzione viene saltato se la funzione Lambda viene eseguita come root (UID=0).

Ti consigliamo di utilizzare un gruppo di sistemi operativi non utilizzato da altre risorse, funzioni Lambda o file sul core Greengrass. L'utilizzo di un gruppo di sistemi operativi condiviso offre alle funzioni Lambda collegate più autorizzazioni di accesso di quelle necessarie. Se si utilizza un gruppo di sistemi operativi condiviso, è necessario allegare anche una funzione Lambda associata a tutte le risorse di machine learning che utilizzano il gruppo di sistemi operativi condiviso. In caso contrario, la distribuzione non riesce.

Autorizzazioni del proprietario del gruppo. L'autorizzazione di sola lettura o lettura e scrittura da aggiungere al processo Lambda.

Le funzioni Lambda non containerizzate devono ereditare queste autorizzazioni di accesso alla risorsa. Le funzioni Lambda containerizzate possono ereditare queste autorizzazioni a livello di risorsa o definire autorizzazioni a livello di funzione. Se definiscono autorizzazioni a livello di funzione, le autorizzazioni devono essere identiche o più restrittive rispetto alle autorizzazioni a livello di risorsa.

Nella tabella seguente vengono illustrate le configurazioni di autorizzazione di accesso supportate.

GGC v1.10 or later

Proprietà	Se solo le funzioni Lambda containerizzate accedono alla risorsa	Se qualche funzione Lambda non containerizzata accede alla risorsa
Proprietà a livello di funzione		
Autorizzazioni (lettura/scrittura)	Obbligatorie a meno che la risorsa non definisca un proprietario delle risorse. Se viene definito un proprietario delle risorse, le autorizzazioni a livello di funzione devono essere identiche o più restrittive rispetto alle autorizzazioni del proprietario delle risorse.	Funzioni Lambda non containerizzate: Non supportato. Le funzioni Lambda non containerizzate devono ereditare le autorizzazioni a livello di risorsa.

Proprietà	Se solo le funzioni Lambda containerizzate accedono alla risorsa	Se qualche funzione Lambda non containerizzata accede alla risorsa
	Se solo le funzioni Lambda containerizzate accedono alla risorsa, ti consigliamo di non definire un proprietario della risorsa.	Funzioni Lambda containerizzate: Facoltativo, ma devono essere identiche o più restrittive rispetto alle autorizzazioni a livello di risorse.

Proprietà a livello di risorse

Proprietario delle risorse	Facoltativo (non consigliato).	Obbligatorio.
Autorizzazioni (lettura/scrittura)	Facoltativo (non consigliato).	Obbligatorio.

GGC v1.9 or earlier

Proprietà	Se solo le funzioni Lambda containerizzate accedono alla risorsa	Se qualche funzione Lambda non containerizzata accede alla risorsa
Proprietà a livello di funzione		
Autorizzazioni (lettura/scrittura)	Obbligatorio.	Non supportato.
Proprietà a livello di risorse		
Proprietario delle risorse	Non supportato.	Non supportato.
Autorizzazioni (lettura/scrittura)	Non supportato.	Non supportato.

 Note

Quando si utilizza l' AWS IoT Greengrass API per configurare le funzioni e le risorse Lambda, è richiesta anche la `ResourceId` proprietà a livello di funzione. La `ResourceId` proprietà collega la risorsa di machine learning alla funzione Lambda.

Definizione delle autorizzazioni di accesso per le funzioni Lambda (console)

Nella AWS IoT console, definisci le autorizzazioni di accesso quando configuri una risorsa di machine learning o ne alleggi una a una funzione Lambda.

Funzioni Lambda containerizzate

Se alla risorsa di machine learning sono collegate solo funzioni Lambda containerizzate:

- Scegli Nessun gruppo di sistema come proprietario della risorsa di machine learning. Questa è l'impostazione consigliata quando solo le funzioni Lambda containerizzate accedono alla risorsa di machine learning. Altrimenti, potresti concedere alle funzioni Lambda allegate più autorizzazioni di accesso di quelle necessarie.

Funzioni Lambda non containerizzate (richiede GGC v1.10 o versione successiva)

Se alla risorsa di machine learning sono collegate funzioni Lambda non containerizzate:

- Specificate il System group ID (GID) da utilizzare come proprietario della risorsa di machine learning. Scegli Specificare il gruppo di sistema e le autorizzazioni e inserisci il GID. È possibile utilizzare il `getent group` comando sul dispositivo principale per cercare l'ID di un gruppo di sistema.
- Scegli Accesso in sola lettura o Accesso in lettura e scrittura per le autorizzazioni del gruppo di sistema.

Definizione delle autorizzazioni di accesso per le funzioni Lambda (API)

Nell' AWS IoT Greengrass API, definisci le autorizzazioni per le risorse di machine learning nella ResourceAccessPolicy proprietà della funzione Lambda o nella proprietà OwnerSetting della risorsa.

Funzioni Lambda containerizzate

Se alla risorsa di machine learning sono collegate solo funzioni Lambda containerizzate:

- Per le funzioni Lambda containerizzate, definisci le autorizzazioni di accesso nella proprietà della Permission proprietà. ResourceAccessPolicies Per esempio:

```
"Functions": [  
  {  
    "Id": "my-containerized-function",  
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:function-name:alias-or-version",  
    "FunctionConfiguration": {  
      "Environment": {  
        "ResourceAccessPolicies": [  
          {  
            "ResourceId": "my-resource-id",  
            "Permission": "ro-or-rw"  
          }  
        ]  
      },  
      "MemorySize": 512,  
      "Pinned": true,  
      "Timeout": 5  
    }  
  }  
]
```

- Per le risorse di machine learning, ometti la proprietà OwnerSetting. Per esempio:

```
"Resources": [  
  {  
    "Id": "my-resource-id",  
    "Name": "my-resource-name",  
    "ResourceDataContainer": {  
      "S3MachineLearningModelResourceData": {  
        "DestinationPath": "/local-destination-path",  

```



```

        "S3Uri": "s3://uri-to-resource-package"
      }
    }
  }
]

```

Questa è la configurazione consigliata quando solo le funzioni Lambda containerizzate accedono alla risorsa di machine learning. Altrimenti, potresti concedere alle funzioni Lambda allegate più autorizzazioni di accesso di quelle necessarie.

Funzioni Lambda non containerizzate (richiede GGC v1.10 o versione successiva)

Se alla risorsa di machine learning sono collegate funzioni Lambda non containerizzate:

- Per le funzioni Lambda non containerizzate, ometti la proprietà `in.PermissionResourceAccessPolicies`. Questa configurazione è obbligatoria e consente alla funzione di ereditare l'autorizzazione a livello di risorsa. Per esempio:

```

"Functions": [
  {
    "Id": "my-non-containerized-function",
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:function-
name:alias-or-version",
    "FunctionConfiguration": {
      "Environment": {
        "Execution": {
          "IsolationMode": "NoContainer",
        },
        "ResourceAccessPolicies": [
          {
            "ResourceId": "my-resource-id"
          }
        ]
      },
      "Pinned": true,
      "Timeout": 5
    }
  }
]

```

- Per le funzioni Lambda containerizzate che accedono anche alla risorsa di machine learning, ometti la `Permission` proprietà `ResourceAccessPolicies` o definisci un'autorizzazione uguale o più restrittiva dell'autorizzazione a livello di risorsa. Per esempio:

```
"Functions": [
  {
    "Id": "my-containerized-function",
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:function-name:alias-or-version",
    "FunctionConfiguration": {
      "Environment": {
        "ResourceAccessPolicies": [
          {
            "ResourceId": "my-resource-id",
            "Permission": "ro-or-rw" // Optional, but cannot exceed
the GroupPermission defined for the resource.
          }
        ]
      },
      "MemorySize": 512,
      "Pinned": true,
      "Timeout": 5
    }
  }
]
```

- Per le risorse di machine learning, definisci la proprietà `OwnerSetting`, inclusi il figlio `GroupOwner` e le proprietà `GroupPermission`. Per esempio:

```
"Resources": [
  {
    "Id": "my-resource-id",
    "Name": "my-resource-name",
    "ResourceDataContainer": {
      "S3MachineLearningModelResourceData": {
        "DestinationPath": "/local-destination-path",
        "S3Uri": "s3://uri-to-resource-package",
        "OwnerSetting": {
          "GroupOwner": "os-group-id",
          "GroupPermission": "ro-or-rw"
        }
      }
    }
  }
]
```

```
}  
]
```

Accesso alle risorse di machine learning dal codice della funzione Lambda

Le funzioni Lambda definite dall'utente utilizzano interfacce del sistema operativo specifiche della piattaforma per accedere alle risorse di machine learning su un dispositivo principale.

GGC v1.10 or later

Per le funzioni Lambda containerizzate, la risorsa è montata all'interno del contenitore Greengrass e disponibile nel percorso di destinazione locale definito per la risorsa. Per le funzioni Lambda non containerizzate, la risorsa è collegata simbolicamente a una directory di lavoro specifica di Lambda e passata alla variabile di ambiente nel processo Lambda. `AWS_GG_RESOURCE_PREFIX`

Per ottenere il percorso degli artefatti scaricati di una risorsa di machine learning, le funzioni Lambda aggiungono la variabile di `AWS_GG_RESOURCE_PREFIX` ambiente al percorso di destinazione locale definito per la risorsa. Per le funzioni Lambda containerizzate, il valore restituito è una singola barra (`.`). /

```
resourcePath = os.getenv("AWS_GG_RESOURCE_PREFIX") + "/destination-path"  
with open(resourcePath, 'r') as f:  
    # load_model(f)
```

GGC v1.9 or earlier

Gli artefatti scaricati di una risorsa di machine learning si trovano nel percorso di destinazione locale definito per la risorsa. Solo le funzioni Lambda containerizzate possono accedere alle risorse di machine learning in AWS IoT Greengrass Core v1.9 e versioni precedenti.

```
resourcePath = "/local-destination-path"  
with open(resourcePath, 'r') as f:  
    # load_model(f)
```

L'implementazione del caricamento del modello dipende dalla libreria ML.

Risoluzione dei problemi

Utilizza le informazioni seguenti per risolvere problemi relativi all'accesso alle risorse di machine learning.

Argomenti

- [MLModelProprietario non valido: GroupOwnerSetting è fornito nella risorsa del modello ML, ma non è presente o non è presente GroupOwner GroupPermission](#)
- [NoContainer la funzione non può configurare l'autorizzazione quando si collegano risorse di Machine Learning. <function-arn>si riferisce alla risorsa Machine Learning <resource-id>con autorizzazione <ro/rw> nella politica di accesso alle risorse.](#)
- [La funzione <function-arn>si riferisce alla risorsa di Machine Learning <resource-id>con autorizzazione mancante in entrambe ResourceAccessPolicy le risorse OwnerSetting.](#)
- [La funzione <function-arn>si riferisce alla risorsa Machine Learning <resource-id>con autorizzazione\ "rw\», mentre l'impostazione del proprietario della risorsa consente GroupPermission solo\ "ro\».](#)
- [NoContainer La funzione <function-arn>si riferisce alle risorse del percorso di destinazione annidato.](#)
- [Lambda <function-arn> ottiene l'accesso alla risorsa <resource-id> condividendo lo stesso ID del proprietario del gruppo](#)

MLModelProprietario non valido: GroupOwnerSetting è fornito nella risorsa del modello ML, ma non è presente o non è presente GroupOwner GroupPermission

Soluzione: viene visualizzato questo errore se una risorsa di machine learning contiene l'[ResourceDownloadOwnerSetting](#) oggetto ma il requisito GroupOwner o la GroupPermission proprietà non sono definiti. Per risolvere questo problema, definisci la proprietà mancante.

NoContainer la funzione non può configurare l'autorizzazione quando si collegano risorse di Machine Learning. <function-arn>si riferisce alla risorsa Machine Learning <resource-id>con autorizzazione <ro/rw> nella politica di accesso alle risorse.

Soluzione: viene visualizzato questo errore se una funzione Lambda non containerizzata specifica autorizzazioni a livello di funzione per una risorsa di machine learning. Le funzioni non

containerizzate devono ereditare le autorizzazioni dalle autorizzazioni del proprietario della risorsa definite nella risorsa di machine learning. Per risolvere questo problema, scegli di [ereditare le autorizzazioni del proprietario della risorsa](#) (console) o [rimuovere le autorizzazioni dalla politica di accesso alle risorse \(API\) della funzione Lambda](#).

La funzione <function-arn> si riferisce alla risorsa di Machine Learning <resource-id> con autorizzazione mancante in entrambe ResourceAccessPolicy le risorse OwnerSetting.

Soluzione: viene visualizzato questo errore se le autorizzazioni per la risorsa di machine learning non sono configurate per la funzione Lambda o la risorsa allegata. Per risolvere questo problema, configura le autorizzazioni nella [ResourceAccessPolicy](#) proprietà per la funzione Lambda o nella proprietà per [OwnerSetting](#) la risorsa.

La funzione <function-arn> si riferisce alla risorsa Machine Learning <resource-id> con autorizzazione \ "rw\», mentre l'impostazione del proprietario della risorsa consente GroupPermission solo \ "ro\».

Soluzione: viene visualizzato questo errore se le autorizzazioni di accesso definite per la funzione Lambda allegata superano le autorizzazioni del proprietario della risorsa definite per la risorsa di machine learning. Per risolvere questo problema, imposta autorizzazioni più restrittive per la funzione Lambda o autorizzazioni meno restrittive per il proprietario della risorsa.

NoContainer La funzione <function-arn> si riferisce alle risorse del percorso di destinazione annidato.

Soluzione: viene visualizzato questo errore se più risorse di machine learning collegate a una funzione Lambda non containerizzata utilizzano lo stesso percorso di destinazione o un percorso di destinazione annidato. Per risolvere questo problema, specifica percorsi di destinazione separati per le risorse.

Lambda <function-arn> ottiene l'accesso alla risorsa <resource-id> condividendo lo stesso ID del proprietario del gruppo

Soluzione: viene visualizzato questo errore `runtime.log` se viene specificato lo stesso gruppo di sistema operativo come identità [Esegui come](#) identità della funzione Lambda e [proprietario della risorsa](#) per una risorsa di machine learning, ma la risorsa non è associata alla funzione Lambda. Questa configurazione fornisce alla funzione Lambda autorizzazioni implicite che può utilizzare per accedere alla risorsa senza autorizzazione. AWS IoT Greengrass

Per risolvere questo problema, usa un gruppo di sistemi operativi diverso per una delle proprietà o collega la risorsa di machine learning alla funzione Lambda.

Consulta anche

- [Esecuzione dell'inferenza di Machine Learning](#)
- [the section called “Come configurare l'inferenza di Machine Learning”](#)
- [the section called “Come configurare l'inferenza di Machine Learning ottimizzata”](#)
- [AWS IoT Greengrass Version 1 Documentazione di riferimento API](#)

Come configurare l'inferenza di Machine Learning mediante la AWS Management Console

Per seguire i passaggi di questo tutorial, è necessario AWS IoT Greengrass Core v1.10 o versione successiva.

Puoi eseguire l'inferenza di Machine Learning (ML) in locale su un dispositivo core Greengrass utilizzando i dati generati localmente. Per informazioni, inclusi i requisiti e i vincoli, consulta [Esecuzione dell'inferenza di Machine Learning](#).

Questo tutorial descrive come utilizzare AWS Management Console per configurare un gruppo Greengrass per eseguire un'app di inferenza Lambda che riconosce le immagini da una fotocamera localmente, senza inviare dati al cloud. L'app di inferenza accede al modulo fotocamera su un Raspberry Pi ed esegue l'inferenza utilizzando il modello open source. [SqueezeNet](#)

Il tutorial include le seguenti fasi di alto livello:

1. [Configurare il dispositivo Raspberry Pi](#)

2. [Installa il MXNet framework](#)
3. [Creazione di un pacchetto del modello](#)
4. [Creare e pubblicare una funzione Lambda](#)
5. [Aggiungere la funzione Lambda al gruppo](#)
6. [Aggiunta di risorse al gruppo](#)
7. [Aggiunta di una sottoscrizione al gruppo](#)
8. [Distribuzione del gruppo.](#)
9. [Esecuzione del test dell'app](#)

Prerequisiti

Per completare questo tutorial, è necessario quanto segue:

- Raspberry Pi 4 Model B o Raspberry Pi 3 Model B/B+, configurato e configurato per l'uso con AWS IoT Greengrass. Per configurare Raspberry Pi con AWS IoT Greengrass, eseguire lo script di [configurazione del dispositivo Greengrass](#) o assicurarsi di aver completato il [modulo 1](#) e il [modulo 2](#) di [Guida introduttiva con AWS IoT Greengrass](#).

Note

Il Raspberry Pi potrebbe richiedere un [alimentatore](#) da 2,5 A per eseguire i framework di deep learning generalmente utilizzati per la classificazione delle immagini. Un alimentatore con una potenza nominale inferiore potrebbe causare il riavvio del dispositivo.

- [Modulo della telecamera Raspberry Pi V2 da 8 Megapixel, 1080p](#). Per informazioni su come configurare la fotocamera, consulta [Collegamento della fotocamera](#) nella documentazione di Raspberry Pi.
- Un gruppo e un core Greengrass. Per informazioni su come creare un gruppo o un nucleo Greengrass, vedere. [Guida introduttiva con AWS IoT Greengrass](#)

Note

Questo tutorial utilizza un Raspberry Pi, ma AWS IoT Greengrass supporta altre piattaforme, come [Intel Atom](#) e [NVIDIA Jetson TX2](#). Nell'esempio di Jetson TX2, è possibile utilizzare immagini statiche anziché immagini trasmesse in streaming da una fotocamera. Se si utilizza

l' TX2 esempio Jetson, potrebbe essere necessario installare Python 3.6 anziché Python 3.7. Per informazioni sulla configurazione del dispositivo in modo da poter installare il software Core, consulta [AWS IoT Greengrass the section called “Configurazione di altri dispositivi”](#). Per le piattaforme di terze parti che AWS IoT Greengrass non supportano, è necessario eseguire la funzione Lambda in modalità non containerizzata. Per l'esecuzione in modalità non containerizzata, è necessario eseguire la funzione Lambda come utente root. Per ulteriori informazioni, consulta [the section called “Considerazioni sulla scelta della containerizzazione delle funzioni Lambda”](#) e [the section called “Impostazione dell'identità di accesso predefinita per le funzioni Lambda in un gruppo”](#).

Fase 1: Configurare il dispositivo Raspberry Pi

In questa fase, verranno installati gli aggiornamenti del sistema operativo Raspbian, il software del modulo della telecamera e le dipendenze Python e verrà abilitata l'interfaccia della telecamera.

Esegui i seguenti comandi nel terminale Raspberry Pi.

1. Installare gli aggiornamenti in Raspbian.

```
sudo apt-get update
sudo apt-get dist-upgrade
```

2. Installare l'interfaccia picamera per il modulo della telecamera e le altre librerie Python necessarie per questo tutorial.

```
sudo apt-get install -y python3-dev python3-setuptools python3-pip python3-picamera
```

Convalidare l'installazione:

- Assicurati che l'installazione di Python 3.7 includa pip.

```
python3 -m pip
```

Se pip non è installato, scaricarlo dal [sito Web pip](#) ed eseguire il comando seguente.

```
python3 get-pip.py
```

- Assicurati che la versione Python sia 3.7 o superiore.


```
python3 --version
```

Se l'output elenca una versione precedente, eseguire il comando seguente.

```
sudo apt-get install -y python3.7-dev
```

- Assicurati che Setuptools e Picamera siano stati installati correttamente.

```
sudo -u ggc_user bash -c 'python3 -c "import setuptools"'  
sudo -u ggc_user bash -c 'python3 -c "import picamera"'
```

Se l'output non contiene errori, la convalida ha esito positivo.

Note

Se l'eseguibile Python installato sul dispositivo è `python3.7`, utilizzare `python3.7` invece di `python3` per i comandi in questo tutorial. Assicurati che l'installazione di pip sia mappata alla versione `python3.7` o `python3` corretta per evitare errori di dipendenza.

3. Riavvia il dispositivo Raspberry Pi.

```
sudo reboot
```

4. Apri lo strumento di configurazione di Raspberry Pi.

```
sudo raspi-config
```

5. Utilizza i tasti freccia per aprire Interfacing Options (Opzioni di interfaccia) e abilita l'interfaccia della telecamera. Se richiesto, consenti il riavvio del dispositivo.
6. Utilizza il seguente comando per eseguire il test della configurazione della telecamera.

```
raspistill -v -o test.jpg
```

Viene visualizzata una finestra di anteprima sul dispositivo Raspberry Pi, viene salvata un'immagine denominata `test.jpg` nella directory corrente e vengono visualizzati informazioni sulla telecamera nel terminale Raspberry Pi.

Fase 2: Installare il framework MXNet

In questo passaggio, installa MXNet le librerie sul tuo Raspberry Pi.

1. Accedere al Raspberry Pi da remoto.

```
ssh pi@your-device-ip-address
```

2. Apri la MXNet documentazione, apri [Installazione MXNet](#) e segui le istruzioni per l'installazione MXNet sul dispositivo.

Note

Ti consigliamo di installare la versione 1.5.0 e di compilarlo MXNet dal codice sorgente per questo tutorial per evitare conflitti tra dispositivi.

3. Dopo l'installazione MXNet, convalida la seguente configurazione:

- Assicurati che l'account `ggc_user` di sistema possa utilizzare il MXNet framework.

```
sudo -u ggc_user bash -c 'python3 -c "import mxnet"'
```

- Assicurati che NumPy sia installato.

```
sudo -u ggc_user bash -c 'python3 -c "import numpy"'
```

Fase 3: Creare un pacchetto MXNet modello

In questo passaggio, crea un pacchetto modello che contenga un MXNet modello preaddestrato di esempio da caricare su Amazon Simple Storage Service (Amazon S3). AWS IoT Greengrass può utilizzare un pacchetto modello di Amazon S3, a condizione che utilizzi il formato `tar.gz` o `zip`.

1. Sul tuo computer, scarica l' MXNet esempio per Raspberry Pi da [the section called “Esempi di Machine Learning”](#)
2. Decomprimere il file `mxnet-py3-armv7l.tar.gz` scaricato.
3. Passa alla directory `squeezenet`.

```
cd path-to-downloaded-sample/mxnet-py3-armv7l/models/squeezenet
```

Il file `squeezenet.zip` in questa directory è il pacchetto del modello. Contiene elementi del modello SqueezeNet open source per un modello di classificazione delle immagini. Successivamente, carichi questo pacchetto modello su Amazon S3.

Fase 4: Creare e pubblicare una funzione Lambda

In questo passaggio, crea un pacchetto di distribuzione della funzione Lambda e una funzione Lambda. Quindi pubblicare una versione della funzione e creare un alias.

Innanzitutto, crea il pacchetto di distribuzione della funzione Lambda.

1. Nel computer, passare alla directory `examples` nel pacchetto di esempio decompresso in [the section called “Creazione di un pacchetto del modello”](#).

```
cd path-to-downloaded-sample/mxnet-py3-armv7l/examples
```

La directory `examples` contiene il codice di funzione e le dipendenze.

- `greengrassObjectClassification.py` è il codice di inferenza utilizzato in questo tutorial. È possibile utilizzare questo codice come modello per creare la propria funzione di inferenza.
- `greengrasssdk` è la versione 1.5.0 del AWS IoT Greengrass Core SDK for Python.

Note

Se è disponibile una nuova versione, è possibile scaricarla e aggiornare la versione dell'SDK nel pacchetto di distribuzione. Per ulteriori informazioni, consulta [AWS IoT Greengrass Core SDK for GitHub Python on](#).

2. Comprimere il contenuto della directory `examples` in un file denominato `greengrassObjectClassification.zip`. Questo è il pacchetto di distribuzione.

```
zip -r greengrassObjectClassification.zip .
```

Note

Assicurarsi inoltre che i file .py e le dipendenze si trovino nella radice della directory.

Quindi, crea la funzione Lambda.

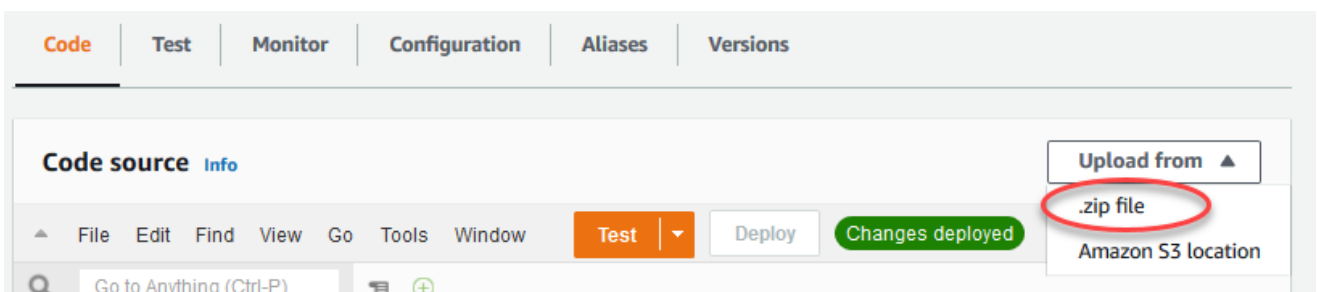
3. Dalla AWS IoT console, scegli Funzioni e Crea funzione.
4. Scegli Author da zero e usa i seguenti valori per creare la tua funzione:
 - Nel campo Function name (Nome funzione), immettere **greengrassObjectClassification**.
 - In Runtime, scegliere Python 3.7.

Per le autorizzazioni, mantieni l'impostazione predefinita. Questo crea un ruolo di esecuzione che concede le autorizzazioni Lambda di base. Questo ruolo non viene utilizzato da AWS IoT Greengrass

5. Scegli Crea funzione.

Ora carica il pacchetto di distribuzione della funzione Lambda e registra il gestore.

6. Scegli la tua funzione Lambda e carica il pacchetto di implementazione della funzione Lambda.
 - a. Nella scheda Codice, in Codice sorgente, scegli Carica da. Dal menu a discesa, scegli un file.zip.



- b. Scegli Carica, quindi scegli il pacchetto di greengrassObjectClassification.zip distribuzione. Quindi, scegliere Save (Salva).

- c. Nella scheda Codice della funzione, in Impostazioni di runtime, scegliete Modifica, quindi immettete i seguenti valori.
- In Runtime, scegliere Python 3.7.
 - Per Handler (Gestore), inserire **greengrassObjectClassification.function_handler**.

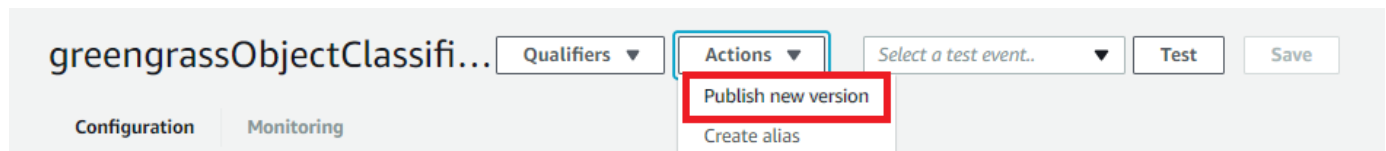
Seleziona Salva.

Successivamente, pubblica la prima versione della tua funzione Lambda. Quindi, creare un [alias per la versione](#).

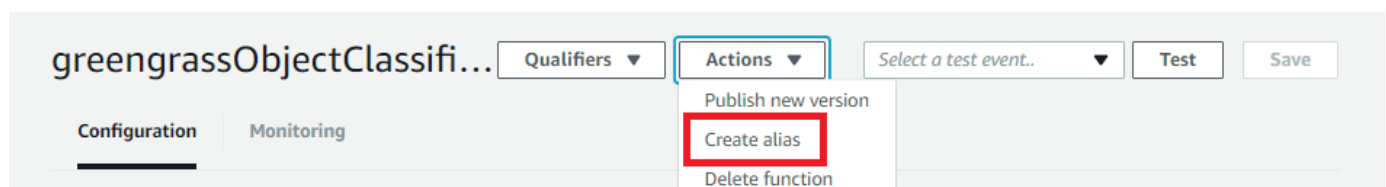
Note

I gruppi Greengrass possono fare riferimento a una funzione Lambda tramite alias (consigliato) o per versione. L'utilizzo di un alias semplifica la gestione degli aggiornamenti del codice perché non è necessario modificare la tabella di sottoscrizione o la definizione del gruppo quando il codice della funzione viene aggiornato. Invece, è sufficiente indirizzare l'alias alla nuova versione della funzione.

7. Nel menu Actions (Operazioni), seleziona Publish new version (Pubblica nuova versione).



8. Per Version description (Descrizione versione), immettere **First version**, quindi scegliere Publish (Pubblica).
9. Nella pagina di configurazione greengrassObjectClassification: 1, dal menu Azioni, scegli Crea alias.



10. Nella pagina Create a new alias (Crea un nuovo alias), utilizza i seguenti valori:

- Per Nome, immetti **m1Test**.

- Per Version (Versione), immettere **1**.

 Note

AWS IoT Greengrass non supporta gli alias Lambda per le versioni \$LATEST.

11. Seleziona Salva.

Ora aggiungete la funzione Lambda al vostro gruppo Greengrass.

Fase 5: Aggiungere la funzione Lambda al gruppo Greengrass

In questo passaggio, aggiungi la funzione Lambda al gruppo e quindi configura il ciclo di vita e le variabili di ambiente.

Innanzitutto, aggiungi la funzione Lambda al tuo gruppo Greengrass.

1. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).
2. Dalla pagina di configurazione del gruppo, scegli la scheda Funzioni Lambda.
3. Nella sezione Funzioni My Lambda, scegli Aggiungi.
4. Per la funzione Lambda, scegli. greengrassObjectClassification
5. Per la versione della funzione Lambda, scegli Alias:MLTest.

Quindi, configura il ciclo di vita e le variabili di ambiente della funzione Lambda.

6. Nella sezione di configurazione della funzione Lambda, apporta i seguenti aggiornamenti.

 Note

Ti consigliamo di eseguire la funzione Lambda senza containerizzazione, a meno che il tuo business case non lo richieda. Ciò consente l'accesso alla GPU e alla fotocamera del dispositivo senza configurare le risorse del dispositivo. Se esegui senza

containerizzazione, devi anche concedere l'accesso root alle tue funzioni Lambda AWS IoT Greengrass .

a. Per eseguire senza containerizzazione:

- Per Utente e gruppo di sistema, scegli. **Another user ID/group ID** Per ID utente di sistema, immettere **0**. Per ID del gruppo di sistema, immettere **0**.

Ciò consente alla funzione Lambda di funzionare come root. Per ulteriori informazioni sull'esecuzione come root, consulta [the section called “Impostazione dell'identità di accesso predefinita per le funzioni Lambda in un gruppo”](#).

 Tip

È inoltre necessario aggiornare il `config.json` file per concedere l'accesso root alla funzione Lambda. Per la procedura, vedi [the section called “Esecuzione di una funzione Lambda come utente root”](#).

- Per la containerizzazione della funzione Lambda, scegli Nessun contenitore.

Per ulteriori informazioni sull'esecuzione senza containerizzazione, consulta. [the section called “Considerazioni sulla scelta della containerizzazione delle funzioni Lambda”](#)

- Per Timeout, immettere **10 seconds**.
- Per Pinned, scegli True.

Per ulteriori informazioni, consulta [the section called “Configurazione del ciclo di vita”](#).

b. Per eseguire invece in modalità containerizzata:

 Note

Si sconsiglia l'esecuzione in modalità containerizzata a meno che il business case non lo richieda.

- Per Utente e gruppo di sistema, scegli Usa i valori predefiniti del gruppo.
- Per la containerizzazione delle funzioni Lambda, scegli Usa default di gruppo.

- Per Memory limit (Limite memoria), immettere **96 MB**.

- Per Timeout, immettere **10 seconds**.
- Per Pinned, scegli True.

Per ulteriori informazioni, consulta [the section called “Configurazione del ciclo di vita”](#).

7. In Environment variables (Variabili di ambiente), creare una coppia chiave-valore. Una coppia chiave-valore è richiesta dalle funzioni che interagiscono con i MXNet modelli su un Raspberry Pi.

Per la chiave, utilizzare MXNET_ENGINE_TYPE. Per il valore, usa. NaiveEngine

Note

Nelle funzioni Lambda definite dall'utente, puoi facoltativamente impostare la variabile di ambiente nel codice della funzione.

8. Mantieni i valori predefiniti per tutte le altre proprietà e scegli Aggiungi funzione Lambda.

Fase 6: aggiunta di risorse al gruppo Greengrass

In questo passaggio, crea risorse per il modulo telecamera e il modello di inferenza ML e associa le risorse alla funzione Lambda. Ciò consente alla funzione Lambda di accedere alle risorse sul dispositivo principale.

Note

Se si esegue in modalità non containerizzata, è AWS IoT Greengrass possibile accedere alla GPU e alla fotocamera del dispositivo senza configurare queste risorse del dispositivo.

Innanzitutto, crea due risorse locali per la telecamera: una per la memoria condivisa e una per l'interfaccia del dispositivo. Per ulteriori informazioni sull'accesso alle risorse locali, consulta [Accedi alle risorse locali con funzioni e connettori Lambda](#).

1. Nella pagina di configurazione del gruppo, scegli la scheda Risorse.
2. Nella sezione Risorse locali, scegli Aggiungi risorsa locale.
3. Nella pagina Aggiungi una risorsa locale, utilizza i seguenti valori:
 - Per Resource Name (Nome risorsa) immetti **videoCoreSharedMemory**.

- Per Resource type (Tipo di risorsa), scegli Device (Dispositivo).
- Per Percorso del dispositivo locale, immettere **/dev/vcsm**.

Il percorso del dispositivo è il percorso assoluto locale della risorsa del dispositivo. Questo percorso può fare riferimento solo a un dispositivo a caratteri o un dispositivo a blocchi in /dev.

- Per il proprietario del gruppo di sistema e le autorizzazioni di accesso ai file, scegli Aggiungi automaticamente le autorizzazioni del file system del gruppo di sistema proprietario della risorsa.

L'opzione System group owner and file access permissions (proprietario del gruppo di sistema e autorizzazioni di accesso ai file) consente di concedere ulteriori autorizzazioni di accesso ai file al processo Lambda. Per ulteriori informazioni, consulta [Autorizzazione di accesso ai file dell'owner del gruppo](#).

4. A questo punto, aggiungi una risorsa del dispositivo locale per l'interfaccia della telecamera.
5. Scegli Aggiungi risorsa locale.
6. Nella pagina Aggiungi una risorsa locale, utilizza i seguenti valori:
 - Per Resource Name (Nome risorsa) immetti **videoCoreInterface**.
 - Per Resource type (Tipo di risorsa), scegli Device (Dispositivo).
 - Per Percorso del dispositivo locale, immettere **/dev/vchiq**.
 - Per il proprietario del gruppo di sistema e le autorizzazioni di accesso ai file, scegli Aggiungi automaticamente le autorizzazioni del file system del gruppo di sistema proprietario della risorsa.
7. Nella parte inferiore della pagina, scegli Aggiungi risorsa.

Ora aggiungi il modello di inferenza come una risorsa Machine Learning. Questo passaggio include il caricamento del pacchetto `squeezenet.zip` modello su Amazon S3.

1. Nella scheda Risorse per il tuo gruppo, nella sezione Machine Learning, scegli Aggiungi risorsa di machine learning.
2. Nella pagina Aggiungi una risorsa di machine learning, per Nome risorsa, inserisci **squeezenet_model**.

3. Per Model source, scegli Usa un modello archiviato in S3, ad esempio un modello ottimizzato tramite Deep Learning Compiler.
4. Per S3 URI, inserisci un percorso in cui viene salvato il bucket S3.
5. Seleziona Sfoglia S3. Si apre una nuova scheda nella console Amazon S3.
6. Nella scheda della console Amazon S3, carica il `squeezenet.zip` file in un bucket S3. Per informazioni, vedi [Come faccio a caricare file e cartelle in un bucket S3?](#) nella Guida per l'utente di Amazon Simple Storage Service.

Note

Affinché il bucket S3 sia accessibile, il nome del bucket deve contenere la stringa **greengrass** e il bucket deve trovarsi nella stessa regione per cui lo utilizzi. AWS IoT Greengrass Scegliere un nome univoco (ad esempio **greengrass-bucket-user-id-epoch-time**). Non utilizzare un punto (.) nel nome del bucket.

7. Nella scheda della AWS IoT Greengrass console, individua e scegli il tuo bucket S3. Individuare e il file `squeezenet.zip` caricato e scegliere Select (Seleziona). Potrebbe essere necessario scegliere Refresh (Aggiorna) per aggiornare l'elenco dei bucket e dei file disponibili.
8. Per Destination path (Percorso di destinazione), immetti **/greengrass-machine-learning/mxnet/squeezenet**.

Questa è la destinazione del modello locale nello spazio dei nomi di runtime Lambda. Quando distribuisce il gruppo, AWS IoT Greengrass recupera il pacchetto del modello sorgente e quindi estrae il contenuto nella directory specificata. La funzione Lambda di esempio per questo tutorial è già configurata per utilizzare questo percorso (nella `model_path` variabile).

9. In Proprietario del gruppo di sistema e autorizzazioni di accesso ai file, scegli Nessun gruppo di sistema.
10. Scegliere Add resource (Aggiungi risorsa).

Utilizzo di modelli addestrati all' SageMaker intelligenza artificiale

Questo tutorial utilizza un modello archiviato in Amazon S3, ma puoi utilizzare facilmente anche modelli SageMaker AI. La AWS IoT Greengrass console ha un'integrazione SageMaker AI integrata, quindi non è necessario caricare manualmente questi modelli su Amazon S3. Per i requisiti e le limitazioni per l'utilizzo dei modelli di SageMaker intelligenza artificiale, consulta [the section called "Origini di modello supportate"](#).

Per utilizzare un modello di SageMaker intelligenza artificiale:

- Per Model source, scegli Usa un modello addestrato all' AWS SageMaker intelligenza artificiale, quindi scegli il nome del processo di formazione del modello.
- Per Percorso di destinazione, inserisci il percorso della directory in cui la funzione Lambda cerca il modello.

Fase 7: aggiunta di una sottoscrizione al gruppo Greengrass

In questa fase, aggiungere una sottoscrizione al gruppo. Questo abbonamento consente alla funzione Lambda di inviare risultati di previsione AWS IoT pubblicandoli su un argomento MQTT.

1. Nella pagina di configurazione del gruppo, scegli la scheda Abbonamenti, quindi scegli Aggiungi abbonamento.
2. Nella pagina dei dettagli dell'abbonamento, configura l'origine e la destinazione come segue:
 - a. In Tipo di sorgente, scegli Funzione Lambda, quindi scegli. greengrassObjectClassification
 - b. Nel tipo di Target, scegli Servizio, quindi scegli IoT Cloud.
3. Nel filtro per argomenti **hello/world**, inserisci, quindi scegli Crea abbonamento.

Fase 8: distribuzione del gruppo Greengrass

In questa fase, distribuire la versione corrente della definizione del gruppo nel dispositivo core Greengrass. La definizione contiene la funzione Lambda, le risorse e le configurazioni di sottoscrizione che hai aggiunto.

1. Assicurati che il AWS IoT Greengrass core sia in esecuzione. Esegui i seguenti comandi nel terminale di Raspberry Pi in base alle esigenze:
 - a. Per controllare se il daemon è in esecuzione:

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

Se l'output contiene una voce root per /greengrass/ggc/packages/1.11.6/bin/daemon, allora il daemon è in esecuzione.

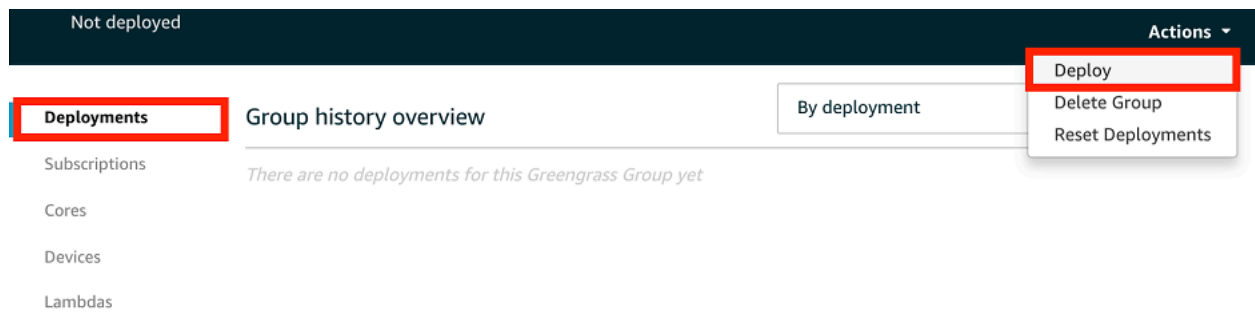
Note

La versione indicata nel percorso dipende dalla versione del software AWS IoT Greengrass Core installata sul dispositivo principale.

- b. Per avviare il demone:

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

2. Nella pagina di configurazione del gruppo, scegli Deploy.



3. Nella scheda Funzioni Lambda, nella sezione Funzioni System Lambda, seleziona Rilevatore IP e scegli Modifica.
4. Nella finestra di dialogo Modifica impostazioni del rilevatore IP, seleziona Rileva e sostituisci automaticamente gli endpoint del broker MQTT.
5. Seleziona Salva.

Questo consente ai dispositivi di acquisire automaticamente informazioni di base sulla connettività, come, ad esempio indirizzo IP, DNS e numero della porta. Il rilevamento automatico è consigliato, ma supporta AWS IoT Greengrass anche gli endpoint specificati manualmente. Ti viene chiesto il metodo di individuazione solo la prima volta che il gruppo viene distribuito.

Note

Se richiesto, concedi l'autorizzazione a creare il ruolo di [servizio Greengrass](#) e associarlo al Account AWS tuo ruolo attuale. Regione AWS Questo ruolo consente di accedere AWS IoT Greengrass alle tue risorse nei AWS servizi.

Nella pagina Deployments (Distribuzioni) vengono visualizzati il timestamp della distribuzione, l'ID versione e lo stato. Una volta completata, lo stato visualizzato per la distribuzione dovrebbe essere Completato.

Per ulteriori informazioni sulle implementazioni, consulta [Distribuisci gruppi AWS IoT Greengrass](#). Per la risoluzione dei problemi, consultare [Risoluzione dei problemi](#).

Fase 9: esecuzione del test dell'app di inferenza

Ora puoi verificare se la distribuzione è configurata correttamente. Per eseguire il test, è necessario abbonarsi all'hello/worldargomento e visualizzare i risultati della previsione pubblicati dalla funzione Lambda.

Note

Se a dispositivo Raspberry Pi è collegato un monitor, il segnale attivo della telecamera viene visualizzato in una finestra di anteprima.

1. Nella AWS IoT console, in Test, scegli MQTT test client.
2. In Subscriptions (Sottoscrizioni), utilizza i seguenti valori:
 - Per l'argomento della sottoscrizione, utilizzare ciao/mondo.
 - In Configurazione aggiuntiva, per la visualizzazione del payload MQTT, scegli Visualizza i payload come stringhe.
3. Scegliere Subscribe (Effettua sottoscrizione).

Se il test ha esito positivo, i messaggi della funzione Lambda vengono visualizzati nella parte inferiore della pagina. Ogni messaggio contiene i primi cinque risultati predittivi dell'immagine nel formato: probabilità, ID classe prevista e nome della classe corrispondente.

Subscribe to a topic

Publish to a topic

hello/world

Publish

Specify a topic and a message to publish with a QoS of 0.

hello/world

```
1 {
2   "message": "Hello from AWS IoT console"
3 }
```

Publish to topic

hello/world	Mar 30, 2018 1:47:07 PM -0700	Export Hide
New Prediction: [(0.31046376, 'n03637318 lampshade, lamp shade'), (0.11445289, 'n04380533 table lamp'), (0.04436367, 'n04254120 soap dispenser'), (0.035816364, 'n04286575 spotlight, spot'), (0.028093718, 'n03201208 dining table, board')]		
hello/world Mar 30, 2018 1:47:01 PM -0700 Export Hide		
New Prediction: [(0.16117829, 'n03876231 paintbrush'), (0.13750333, 'n04442312 toaster'), (0.081819646, 'n03924679 photocopier'), (0.068144165, 'n02783161 ballpoint, ballpoint pen, ballpen, Biro'), (0.044701375, 'n04209239 shower curtain')]		
hello/world Mar 30, 2018 1:46:55 PM -0700 Export Hide		
New Prediction: [(0.46284258, 'n04442312 toaster'), (0.16061385, 'n03908618 pencil box, pencil case'), (0.043834824, 'n03291819 envelope'), (0.027529096, 'n03908714 pencil sharpener'), (0.027273422, 'n04209239 shower curtain')]		

Risoluzione dei problemi di AWS IoT Greengrass inferenza ML

Se il test non viene completato correttamente, puoi provare a eseguire la procedura di risoluzione dei problemi riportata di seguito. Esegui i comandi nel terminale Raspberry Pi.

Controlla i log degli errori

1. Passare all'utente root e navigare alla directory log. L'accesso ai AWS IoT Greengrass log richiede i permessi di root.

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

2. Nella directory system, controllare runtime.log o python_runtime.log.

Nella directory user/*region/account-id*, controllare greengrassObjectClassification.log.

Per ulteriori informazioni, consulta [the section called “Risoluzione dei problemi con i log”](#).

Errore di decompressione in runtime.log

Se runtime.log contiene un errore simile al seguente, assicurati che il pacchetto del modello di origine tar.gz contenga una directory principale.

```
Greengrass deployment error: unable to download the artifact model-arn: Error while processing.  
Error while unpacking the file from /tmp/greengrass/artifacts/model-arn/path to /greengrass/ggc/deployment/path/model-arn,  
error: open /greengrass/ggc/deployment/path/model-arn/squeezenet/squeezenet_v1.1-0000.params: no such file or directory
```

Se il pacchetto non dispone di una directory principale contenente i file del modello, ricomprimi il modello utilizzando il seguente comando:

```
tar -zcvf model.tar.gz ./model
```

Per esempio:

```
##$ tar -zcvf test.tar.gz ./test  
./test  
./test/some.file  
./test/some.file2  
./test/some.file3
```

 Note

Non includere i caratteri finali `/*` in questo comando.

Verifica che la funzione Lambda sia implementata correttamente

1. Elenca il contenuto della Lambda distribuita nella `/lambda` directory. Prima di eseguire il comando, sostituisci i valori dei segnaposti.

```
cd /greengrass/ggc/deployment/lambda/  
arn:aws:lambda:region:account:function:function-name:function-version  
ls -la
```

2. Verifica che la directory contenga gli stessi file inclusi nel pacchetto di distribuzione `greengrassObjectClassification.zip` caricato in [Fase 4: Creare e pubblicare una funzione Lambda](#).

Assicurati inoltre che i file `.py` e le dipendenze si trovino nella root della directory.

Verifica che il modello di inferenza sia stato distribuito correttamente

1. Trova il numero di identificazione del processo (PID) del processo di runtime Lambda:

```
ps aux | grep 'lambda-function-name'
```

Nell'output, il PID appare nella seconda colonna della riga per il processo di runtime Lambda.

2. Inserisci lo spazio dei nomi di runtime Lambda. Assicurati di sostituire il *pid* valore segnaposto prima di eseguire il comando.

Note

Questa directory e il suo contenuto si trovano nello spazio dei nomi di runtime Lambda, quindi non sono visibili in un normale spazio dei nomi Linux.

```
sudo nsenter -t pid -m /bin/bash
```

3. Elenca i contenuti della directory locale specificata per la risorsa ML.

```
cd /greengrass-machine-learning/mxnet/squeezenet/  
ls -ls
```

Dovrebbero essere visualizzati i seguenti file:

```
32 -rw-r--r-- 1 ggc_user ggc_group 31675 Nov 18 15:19 synset.txt  
32 -rw-r--r-- 1 ggc_user ggc_group 28707 Nov 18 15:19 squeezenet_v1.1-symbol.json  
4832 -rw-r--r-- 1 ggc_user ggc_group 4945062 Nov 18 15:19  
squeezenet_v1.1-0000.params
```


Passaggi successivi

Successivamente, esplora altre app di inferenza. AWS IoT Greengrass fornisce altre funzioni Lambda che è possibile utilizzare per provare l'inferenza locale. Il pacchetto degli esempi è disponibile nella cartella delle librerie precompilate scaricata nella [the section called “Installa il MXNet framework”](#).

Configurazione di un dispositivo Intel Atom

Per eseguire questo tutorial su un dispositivo Intel Atom, è necessario fornire immagini di origine, configurare la funzione Lambda e aggiungere un'altra risorsa del dispositivo locale. Per utilizzare la GPU per l'inferenza, assicurarsi che sul dispositivo sia installato il seguente software:

- OpenCL versione 1.0 o successiva
- Python 3.7 e pip

Note

Se il dispositivo è preconfigurato con Python 3.6, puoi invece creare un collegamento simbolico a Python 3.7. Per ulteriori informazioni, consulta [Step 2](#).

- [NumPy](#)
- [OpenCV su Wheels](#)

1. Scarica immagini PNG o JPG statiche per la funzione Lambda da utilizzare per la classificazione delle immagini. L'esempio funziona in modo ottimale con file immagine di dimensioni ridotte.

Salva i file immagine nella directory contenente il file `greengrassObjectClassification.py` (o in una sottodirectory di questa directory). Si trova nel pacchetto di distribuzione della funzione Lambda in cui carichi. [the section called “Creare e pubblicare una funzione Lambda”](#)

Note

Se la utilizzi AWS DeepLens, puoi utilizzare la videocamera di bordo o montare la tua fotocamera per eseguire inferenze sulle immagini acquisite anziché sulle immagini statiche. Tuttavia, ti consigliamo di iniziare con le immagini statiche.

Se si utilizza una telecamera, assicurarsi che il pacchetto APT `awscam` sia installato e aggiornato. Per ulteriori informazioni, consulta [Aggiorna il tuo AWS DeepLens dispositivo nella Guida](#) per gli AWS DeepLens sviluppatori.

- Se non stai usando Python 3.7, assicurati di creare un collegamento simbolico da Python 3.x a Python 3.7. Questo configura il tuo dispositivo per usare Python 3 con. AWS IoT Greengrass. Eseguire il seguente comando per individuare l'installazione di Python:

```
which python3
```

Eseguire il comando seguente per creare il collegamento simbolico.

```
sudo ln -s path-to-python-3.x/python3.x path-to-python-3.7/python3.7
```

Riavviare il dispositivo.

- Modifica la configurazione della funzione Lambda. Segui la procedura riportata in [the section called "Aggiungere la funzione Lambda al gruppo"](#).

Note

Ti consigliamo di eseguire la funzione Lambda senza containerizzazione, a meno che il tuo business case non lo richieda. Ciò consente l'accesso alla GPU e alla fotocamera del dispositivo senza configurare le risorse del dispositivo. Se esegui senza containerizzazione, devi anche concedere l'accesso root alle tue funzioni Lambda AWS IoT Greengrass .

a. Per eseguire senza containerizzazione:

- Per Utente e gruppo di sistema, scegli. **Another user ID/group ID** Per ID utente di sistema, immettere `0`. Per ID del gruppo di sistema, immettere `0`.

Ciò consente alla funzione Lambda di funzionare come root. Per ulteriori informazioni sull'esecuzione come root, consulta [the section called "Impostazione dell'identità di accesso predefinita per le funzioni Lambda in un gruppo"](#).

 Tip

È inoltre necessario aggiornare il `config.json` file per concedere l'accesso root alla funzione Lambda. Per la procedura, vedi [the section called “Esecuzione di una funzione Lambda come utente root”](#).

- Per la containerizzazione della funzione Lambda, scegli Nessun contenitore.

Per ulteriori informazioni sull'esecuzione senza containerizzazione, consulta [the section called “Considerazioni sulla scelta della containerizzazione delle funzioni Lambda”](#)

- Aggiornare il valore Timeout a 5 secondi. In questo modo, il timeout della richiesta non viene eseguito troppo presto. L'esecuzione dell'inferenza richiede alcuni minuti dopo la configurazione.
- In Bloccato, scegli True.
- In Parametri aggiuntivi, per Accesso in lettura alla directory `/sys`, scegli Abilitato.
- Per Lambda lifecycle (Ciclo di vita Lambda), scegli Make this function long-lived and keep it running indefinitely (Rendi questa funzione di lunga durata e mantieni in esecuzione a tempo indeterminato).

b. Per eseguire invece in modalità containerizzata:

 Note

Si sconsiglia l'esecuzione in modalità containerizzata a meno che il business case non lo richieda.

- Aggiornare il valore Timeout a 5 secondi. In questo modo, il timeout della richiesta non viene eseguito troppo presto. L'esecuzione dell'inferenza richiede alcuni minuti dopo la configurazione.
- Per Pinned, scegli True.
- In Parametri aggiuntivi, per Accesso in lettura alla directory `/sys`, scegli Abilitato.

4. Se è in esecuzione in modalità containerizzata, aggiungi la risorsa del dispositivo locale richiesta per concedere l'accesso alla GPU del dispositivo.

 Note

Se esegui in modalità non containerizzata, AWS IoT Greengrass puoi accedere alla GPU del dispositivo senza configurare le risorse del dispositivo.

- a. Nella pagina di configurazione del gruppo, scegli la scheda Risorse.
- b. Scegli Aggiungi risorsa locale.
- c. Definisci la risorsa:
 - Per Resource Name (Nome risorsa) immetti **renderD128**.
 - Per Tipo di risorsa, scegli Dispositivo locale.
 - Per Device path (Percorso dispositivo), immetti **/dev/dri/renderD128**.
 - Per il proprietario del gruppo di sistema e le autorizzazioni di accesso ai file, scegli Aggiungi automaticamente le autorizzazioni del file system del gruppo di sistema proprietario della risorsa.
 - Per le affiliazioni alle funzioni Lambda, concedi l'accesso in lettura e scrittura alla tua funzione Lambda.

Configurazione di un NVIDIA Jetson TX2

Per eseguire questo tutorial su un NVIDIA Jetson TX2, fornisci immagini sorgente e configura la funzione Lambda. Se stai utilizzando la GPU, devi anche aggiungere le risorse locali del dispositivo.

1. Assicurati che il tuo dispositivo Jetson sia configurato in modo da poter installare il software Core. AWS IoT Greengrass Per ulteriori informazioni sulla configurazione del dispositivo, consulta [the section called “Configurazione di altri dispositivi”](#).
2. Apri la MXNet documentazione, vai a [Installazione MXNet su un Jetson](#) e segui le istruzioni per l'installazione MXNet sul dispositivo Jetson.

 Note

Se vuoi creare MXNet dal codice sorgente, segui le istruzioni per creare la libreria condivisa. Modifica le seguenti impostazioni nel tuo `config.mk` file per lavorare con un dispositivo Jetson TX2 :

- Aggiungere `-gencode arch=compute-62, code=sm_62` all'impostazione `CUDA_ARCH`.
- Attivare CUDA.

```
USE_CUDA = 1
```

3. Scarica immagini PNG o JPG statiche per la funzione Lambda da utilizzare per la classificazione delle immagini. L'app funziona in modo ottimale con i file immagine di dimensioni ridotte. In alternativa, puoi implementare una telecamera sulla scheda Jetson per acquisire le immagini di origine.

Salvare i file immagine nella directory contenente il file `greengrassObjectClassification.py`. È possibile salvarli anche in una sottodirectory di questa directory. Questa directory si trova nel pacchetto di distribuzione della funzione Lambda in cui carichi. [the section called “Creare e pubblicare una funzione Lambda”](#)

4. Crea un collegamento simbolico da Python 3.7 a Python 3.6 con cui usare Python 3. AWS IoT Greengrass Eseguire il seguente comando per individuare l'installazione di Python:

```
which python3
```

Eseguire il comando seguente per creare il collegamento simbolico.

```
sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
```

Riavviare il dispositivo.

5. Assicurati che l'account di `ggc_user` sistema possa utilizzare il framework: MXNet

```
"sudo -u ggc_user bash -c 'python3 -c "import mxnet"'
```

6. Modifica la configurazione della funzione Lambda. Segui la procedura riportata in [the section called “Aggiungere la funzione Lambda al gruppo”](#).

Note

Ti consigliamo di eseguire la funzione Lambda senza containerizzazione, a meno che il tuo business case non lo richieda. Ciò consente l'accesso alla GPU e alla

fotocamera del dispositivo senza configurare le risorse del dispositivo. Se esegui senza containerizzazione, devi anche concedere l'accesso root alle tue funzioni Lambda AWS IoT Greengrass .

a. Per eseguire senza containerizzazione:

- Per Utente e gruppo di sistema, scegli. **Another user ID/group ID** Per ID utente di sistema, immettere0. Per ID del gruppo di sistema, immettere0.

Ciò consente alla funzione Lambda di funzionare come root. Per ulteriori informazioni sull'esecuzione come root, consulta [the section called “Impostazione dell'identità di accesso predefinita per le funzioni Lambda in un gruppo”](#).

 Tip

È inoltre necessario aggiornare il `config.json` file per concedere l'accesso root alla funzione Lambda. Per la procedura, vedi [the section called “Esecuzione di una funzione Lambda come utente root”](#).

- Per la containerizzazione della funzione Lambda, scegli Nessun contenitore.

Per ulteriori informazioni sull'esecuzione senza containerizzazione, consulta. [the section called “Considerazioni sulla scelta della containerizzazione delle funzioni Lambda”](#)

- In Parametri aggiuntivi, per Accesso in lettura alla directory `/sys`, scegli Abilitato.
- In Variabili di ambiente, aggiungi le seguenti coppie chiave-valore alla tua funzione Lambda. Questo configura l'utilizzo del framework AWS IoT Greengrass . MXNet

Chiave	Valore
PATH	/usr/local/cuda/bin: \$PATH
MXNET_HOME	\$HOME/mxnet/
PYTHONPATH	\$MXNET_HOME/python:\$PYTHONPATH
CUDA_HOME	/usr/local/cuda

Chiave	Valore
LD_LIBRARY_PATH	\$LD_LIBRARY_PATH:\${CUDA_HOME}/lib64

b. Per eseguire invece in modalità containerizzata:

 Note

Si sconsiglia l'esecuzione in modalità containerizzata a meno che il business case non lo richieda.

- Aumenta il valore di Memory limit (Limite memoria). Utilizzare 500 MB per la CPU o almeno 2000 MB per la GPU.
- In Parametri aggiuntivi, per Accesso in lettura alla directory /sys, scegli Abilitato.
- In Variabili di ambiente, aggiungi le seguenti coppie chiave-valore alla tua funzione Lambda. Questo configura l'utilizzo del framework AWS IoT Greengrass . MXNet

Chiave	Valore
PATH	/usr/local/cuda/bin: \$PATH
MXNET_HOME	\$HOME/mxnet/
PYTHONPATH	\$MXNET_HOME/python:\$PYTHONPATH
CUDA_HOME	/usr/local/cuda
LD_LIBRARY_PATH	\$LD_LIBRARY_PATH:\${CUDA_HOME}/lib64

7. Se è in esecuzione in modalità containerizzata, aggiungi le seguenti risorse del dispositivo locale per concedere l'accesso alla GPU del dispositivo. Segui la procedura riportata in [the section called “Aggiunta di risorse al gruppo”](#).

 Note

Se esegui in modalità non containerizzata, AWS IoT Greengrass puoi accedere alla GPU del dispositivo senza configurare le risorse del dispositivo.

Per ogni risorsa:

- Per Resource type (Tipo di risorsa), scegli Device (Dispositivo).
- Per il proprietario del gruppo di sistema e le autorizzazioni di accesso ai file, scegli Aggiungi automaticamente le autorizzazioni del file system del gruppo di sistema proprietario della risorsa.

Nome	Percorso dispositivo
nvhost-ctrl	/dev/nvhost-ctrl
nvhost-gpu	/dev/nvhost-gpu
nvhost-ctrl-gpu	/dev/ nvhost-ctrl-gpu
nvhost-dbg-gpu	/dev/ nvhost-dbg-gpu
nvhost-prof-gpu	/dev/ nvhost-prof-gpu
nvmap	/dev/nvmap
nvhost-vic	/dev/nvhost-vic
tegra_dc_ctrl	/dev/tegra_dc_ctrl

8. Se è in esecuzione in modalità containerizzata, aggiungi la seguente risorsa di volume locale per concedere l'accesso alla fotocamera del tuo dispositivo. Segui la procedura riportata in [the section called “Aggiunta di risorse al gruppo”](#).

Note

Se si esegue in modalità non containerizzata, è AWS IoT Greengrass possibile accedere alla fotocamera del dispositivo senza configurare le risorse di volume.

- Per Resource type (Tipo di risorsa), scegli Volume.
- Per il proprietario del gruppo di sistema e le autorizzazioni di accesso ai file, scegli Aggiungi automaticamente le autorizzazioni del file system del gruppo di sistema proprietario della risorsa.

Nome	Percorso di origine	Percorso di destinazione
shm	/dev/shm	/dev/shm
tmp	/tmp	/tmp

Come configurare l'inferenza di Machine Learning ottimizzata mediante la AWS Management Console

Per seguire i passaggi di questo tutorial, devi utilizzare AWS IoT Greengrass Core v1.10 o versione successiva.

Puoi utilizzare il compilatore di deep learning SageMaker AI Neo per ottimizzare l'efficienza di previsione dei modelli di inferenza nativi di machine learning in Tensorflow, Apache MXNet PyTorch, ONNX e framework per un ingombro ridotto e prestazioni più veloci. XGBoost Puoi quindi scaricare il modello ottimizzato e installare il runtime di deep learning SageMaker AI Neo e distribuirlo sui tuoi dispositivi per un'inferenza più rapida. AWS IoT Greengrass

Questo tutorial descrive come utilizzare AWS Management Console per configurare un gruppo Greengrass per eseguire un esempio di inferenza Lambda che riconosce le immagini da una fotocamera localmente, senza inviare dati al cloud. L'esempio di inferenza seguente consente di accedere al modulo della telecamera in un Raspberry Pi. In questo tutorial, scarichi un modello preconfezionato addestrato da Resnet-50 e ottimizzato nel compilatore di deep learning Neo. Il

modello viene quindi utilizzato per eseguire la classificazione locale delle immagini sul dispositivo.
AWS IoT Greengrass

Il tutorial include le seguenti fasi di alto livello:

1. [Configurare il dispositivo Raspberry Pi](#)
2. [Installazione di Neo Deep Learning Runtime](#)
3. [Creare una funzione Lambda di inferenza](#)
4. [Aggiungere la funzione Lambda al gruppo](#)
5. [Aggiunta della risorsa del modello ottimizzato Neo al gruppo](#)
6. [Aggiunta della risorsa del dispositivo della telecamera al gruppo](#)
7. [Aggiunta di sottoscrizioni al gruppo](#)
8. [Distribuzione del gruppo.](#)
9. [Test dell'esempio](#)

Prerequisiti

Per completare questo tutorial, è necessario quanto segue:

- Raspberry Pi 4 Model B o Raspberry Pi 3 Model B/B+, configurati e configurati per l'uso con. AWS IoT Greengrass Per configurare Raspberry Pi con AWS IoT Greengrass, eseguire lo script di [configurazione del dispositivo Greengrass](#) o assicurarsi di aver completato il [modulo 1](#) e il [modulo 2](#) di [Guida introduttiva con AWS IoT Greengrass](#).

Note

Il Raspberry Pi potrebbe richiedere un [alimentatore](#) da 2,5 A per eseguire i framework di deep learning generalmente utilizzati per la classificazione delle immagini. Un alimentatore con una potenza nominale inferiore potrebbe causare il riavvio del dispositivo.

- [Modulo della telecamera Raspberry Pi V2 da 8 Megapixel, 1080p](#). Per informazioni sulla configurazione della telecamera, consulta [Connessione della telecamera](#) nella documentazione di Raspberry Pi.
- Un gruppo e un core Greengrass. Per informazioni su come creare un gruppo o un core Greengrass, consulta [Guida introduttiva con AWS IoT Greengrass](#).

 Note

Questo tutorial utilizza un Raspberry Pi, ma AWS IoT Greengrass supporta altre piattaforme, come [Intel Atom](#) e [NVIDIA Jetson. TX2](#). Se si utilizza l'esempio Intel Atom, potrebbe essere necessario installare Python 3.6 invece di Python 3.7. Per informazioni sulla configurazione del dispositivo in modo da poter installare il software Core, consulta AWS IoT Greengrass . [the section called “Configurazione di altri dispositivi”](#)

Per le piattaforme di terze parti che AWS IoT Greengrass non supportano, è necessario eseguire la funzione Lambda in modalità non containerizzata. Per l'esecuzione in modalità non containerizzata, è necessario eseguire la funzione Lambda come utente root. Per ulteriori informazioni, consulta [the section called “Considerazioni sulla scelta della containerizzazione delle funzioni Lambda”](#) e [the section called “Impostazione dell'identità di accesso predefinita per le funzioni Lambda in un gruppo”](#).

Fase 1: Configurare il dispositivo Raspberry Pi

In questa fase, verranno installati gli aggiornamenti del sistema operativo Raspbian, il software del modulo della telecamera e le dipendenze Python e verrà abilitata l'interfaccia della telecamera.

Esegui i seguenti comandi nel terminale Raspberry Pi.

1. Installare gli aggiornamenti in Raspbian.

```
sudo apt-get update
sudo apt-get dist-upgrade
```

2. Installare l'interfaccia picamera per il modulo della telecamera e le altre librerie Python necessarie per questo tutorial.

```
sudo apt-get install -y python3-dev python3-setuptools python3-pip python3-picamera
```

Convalidare l'installazione:

- Assicurati che l'installazione di Python 3.7 includa pip.

```
python3 -m pip
```

Se pip non è installato, scaricarlo dal [sito Web pip](#) ed eseguire il comando seguente.

```
python3 get-pip.py
```

- Assicurati che la versione Python sia 3.7 o superiore.

```
python3 --version
```

Se l'output elenca una versione precedente, eseguire il comando seguente.

```
sudo apt-get install -y python3.7-dev
```

- Assicurati che Setuptools e Picamera siano stati installati correttamente.

```
sudo -u ggc_user bash -c 'python3 -c "import setuptools"'  
sudo -u ggc_user bash -c 'python3 -c "import picamera"'
```

Se l'output non contiene errori, la convalida ha esito positivo.

Note

Se l'eseguibile Python installato sul dispositivo è `python3.7`, utilizzare `python3.7` invece di `python3` per i comandi in questo tutorial. Assicurati che l'installazione di pip sia mappata alla versione `python3.7` o `python3` corretta per evitare errori di dipendenza.

3. Riavvia il dispositivo Raspberry Pi.

```
sudo reboot
```

4. Apri lo strumento di configurazione di Raspberry Pi.

```
sudo raspi-config
```

5. Utilizza i tasti freccia per aprire Interfacing Options (Opzioni di interfaccia) e abilita l'interfaccia della telecamera. Se richiesto, consenti il riavvio del dispositivo.
6. Utilizza il seguente comando per eseguire il test della configurazione della telecamera.

```
raspistill -v -o test.jpg
```

Viene visualizzata una finestra di anteprima sul dispositivo Raspberry Pi, viene salvata un'immagine denominata `test.jpg` nella directory corrente e vengono visualizzate informazioni sulla telecamera nel terminale Raspberry Pi.

Fase 2: installa il runtime di deep learning di Amazon SageMaker Neo

In questo passaggio, installa il Neo deep learning runtime (DLR) sul tuo Raspberry Pi.

Note

Per questo tutorial si consiglia di installare la versione 1.1.0.

1. Accedere al Raspberry Pi da remoto.

```
ssh pi@your-device-ip-address
```

2. Aprire la documentazione DLR, aprire [Installazione DLR](#) e individuare il wheel URL dei dispositivi Raspberry Pi. Seguire quindi le istruzioni per installare il DLR sul dispositivo. Ad esempio, è possibile utilizzare pip:

```
pip3 install rasp3b-wheel-url
```

3. Dopo aver installato il DLR, convalidare la seguente configurazione:

- Assicurarsi che l'account di sistema `ggc_user` possa utilizzare la libreria DLR.

```
sudo -u ggc_user bash -c 'python3 -c "import dlr"'
```

- Assicurati che NumPy sia installato.

```
sudo -u ggc_user bash -c 'python3 -c "import numpy"'
```

Fase 3: Creare una funzione Lambda di inferenza

In questo passaggio, crea un pacchetto di distribuzione della funzione Lambda e una funzione Lambda. Quindi pubblicare una versione della funzione e creare un alias.

1. Sul computer, scaricare l'esempio DLR per Raspberry Pi da [the section called “Esempi di Machine Learning”](#).
2. Decomprimere il file `dlr-py3-armv7l.tar.gz` scaricato.

```
cd path-to-downloaded-sample  
tar -xvzf dlr-py3-armv7l.tar.gz
```

La directory `examples` nel pacchetto di esempio estratto contiene il codice di funzione e le dipendenze.

- `inference.py` è il codice di inferenza utilizzato in questo tutorial. È possibile utilizzare questo codice come modello per creare la propria funzione di inferenza.
- `greengrasssdk` è la versione 1.5.0 del AWS IoT Greengrass Core SDK for Python.

 Note

Se è disponibile una nuova versione, è possibile scaricarla e aggiornare la versione dell'SDK nel pacchetto di distribuzione. Per ulteriori informazioni, consulta [AWS IoT Greengrass Core SDK for GitHub Python on](#).

3. Comprimere il contenuto della directory `examples` in un file denominato `optimizedImageClassification.zip`. Questo è il pacchetto di distribuzione.

```
cd path-to-downloaded-sample/dlr-py3-armv7l/examples  
zip -r optimizedImageClassification.zip .
```

Il pacchetto di distribuzione contiene il codice di funzione e le dipendenze. Ciò include il codice che richiama il runtime di Neo deep learning Python APIs per eseguire inferenze con i modelli del compilatore Neo deep learning.

 Note

Assicurarsi inoltre che i file `.py` e le dipendenze si trovino nella radice della directory.

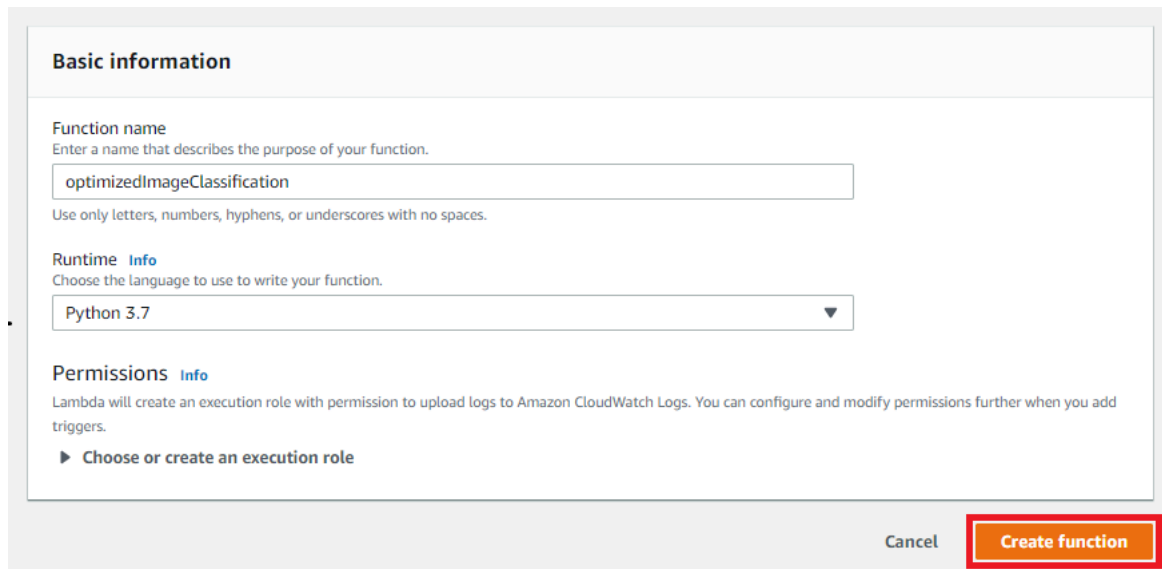
4. Ora aggiungete la funzione Lambda al vostro gruppo Greengrass.

Dalla pagina della console Lambda, scegli Funzioni e scegli Crea funzione.

5. Scegli Author da zero e usa i seguenti valori per creare la tua funzione:

- Nel campo Function name (Nome funzione), immettere **optimizedImageClassification**.
- In Runtime, scegliere Python 3.7.

Per le autorizzazioni, mantieni l'impostazione predefinita. Questo crea un ruolo di esecuzione che concede le autorizzazioni Lambda di base. Questo ruolo non viene utilizzato da AWS IoT Greengrass

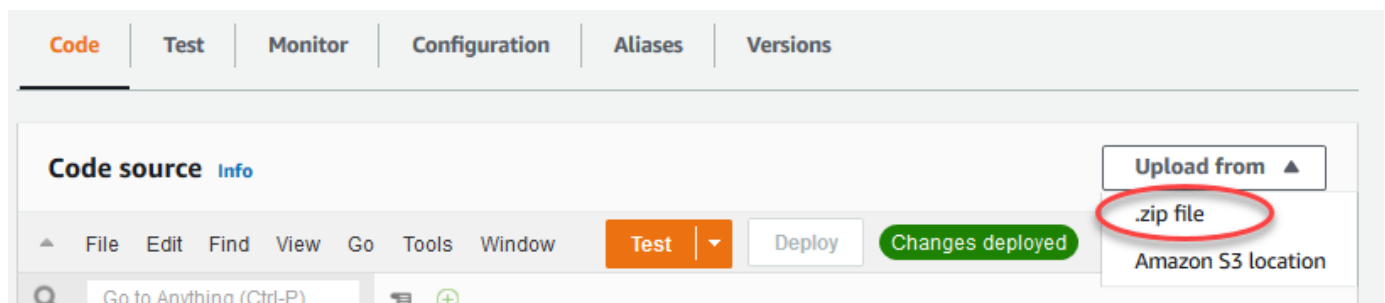


The screenshot shows the 'Basic information' tab of the AWS Lambda 'Create function' wizard. The 'Function name' field contains 'optimizedImageClassification'. The 'Runtime' dropdown is set to 'Python 3.7'. The 'Permissions' section shows a link to 'Choose or create an execution role'. At the bottom right, there are 'Cancel' and 'Create function' buttons, with the latter highlighted by a red rectangle.

6. Scegli Crea funzione.

Ora carica il pacchetto di distribuzione della funzione Lambda e registra il gestore.

1. Nella scheda Codice, in Codice sorgente, scegli Carica da. Dal menu a discesa, scegli un file.zip.



The screenshot shows the 'Code source' section of the AWS Lambda console. The 'Upload from' dropdown menu is open, and the option '.zip file' is selected and circled in red. Other options visible are 'File', 'S3', and 'Lambda'. The 'Test' button is highlighted with an orange background.

2. Scegli il pacchetto **optimizedImageClassification.zip** di distribuzione, quindi scegli Salva.

3. Nella scheda Codice della funzione, in Impostazioni di runtime, scegli Modifica, quindi inserisci i seguenti valori.

- In Runtime, scegliere Python 3.7.
- Per Handler (Gestore), inserire **inference.handler**.

Seleziona Salva.

Successivamente, pubblica la prima versione della tua funzione Lambda. Quindi, creare un [alias per la versione](#).

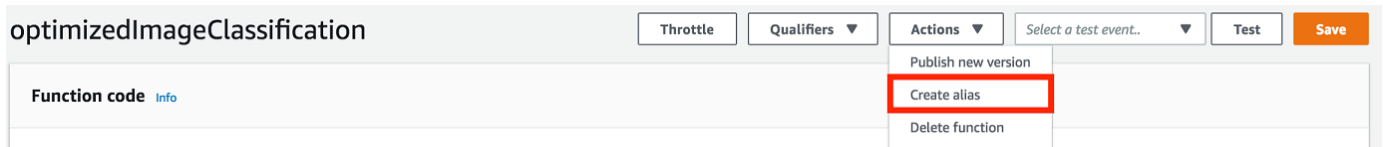
Note

I gruppi Greengrass possono fare riferimento a una funzione Lambda tramite alias (consigliato) o per versione. L'utilizzo di un alias semplifica la gestione degli aggiornamenti del codice perché non è necessario modificare la tabella di sottoscrizione o la definizione del gruppo quando il codice della funzione viene aggiornato. È sufficiente invece indirizzare l'alias alla nuova versione della funzione.

1. Nel menu Actions (Operazioni), seleziona Publish new version (Pubblica nuova versione).

2. Per Version description (Descrizione versione), immettere **First version**, quindi scegliere Publish (Pubblica).

3. Nella pagina di configurazione `optimizedImageClassification`: 1, dal menu Azioni, scegli Crea alias.



4. Nella pagina Create a new alias (Crea un nuovo alias), utilizza i seguenti valori:
 - Per Nome, immetti **m1Test0pt**.
 - Per Version (Versione), immettere **1**.

Note

AWS IoT Greengrass non supporta gli alias Lambda per le versioni \$LATEST.

5. Scegli Create (Crea) .

Ora aggiungete la funzione Lambda al vostro gruppo Greengrass.

Fase 4: Aggiungere la funzione Lambda al gruppo Greengrass

In questo passaggio, aggiungi la funzione Lambda al gruppo, quindi configura il suo ciclo di vita.

Innanzitutto, aggiungi la funzione Lambda al tuo gruppo Greengrass.

1. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).
2. Nella pagina di configurazione dei gruppi, scegli la scheda Funzioni Lambda e scegli Aggiungi.
3. Scegli la funzione Lambda e seleziona. `optimizedImageClassification`
4. Nella versione della funzione Lambda, scegli l'alias della versione che hai pubblicato.

Quindi, configura il ciclo di vita della funzione Lambda.

1. Nella sezione di configurazione della funzione Lambda, apporta i seguenti aggiornamenti.

 Note

Ti consigliamo di eseguire la funzione Lambda senza containerizzazione a meno che il tuo business case non lo richieda. Ciò consente l'accesso alla GPU e alla fotocamera del dispositivo senza configurare le risorse del dispositivo. Se esegui senza containerizzazione, devi anche concedere l'accesso root alle tue funzioni Lambda AWS IoT Greengrass .

a. Per eseguire senza containerizzazione:

- Per Utente e gruppo di sistema, scegli. **Another user ID/group ID** Per ID utente di sistema, immettere **0**. Per ID del gruppo di sistema, immettere **0**.

Ciò consente alla funzione Lambda di funzionare come root. Per ulteriori informazioni sull'esecuzione come root, consulta [the section called “Impostazione dell'identità di accesso predefinita per le funzioni Lambda in un gruppo”](#).

 Tip

È inoltre necessario aggiornare il `config.json` file per concedere l'accesso root alla funzione Lambda. Per la procedura, vedi [the section called “Esecuzione di una funzione Lambda come utente root”](#).

- Per la containerizzazione della funzione Lambda, scegli Nessun contenitore.

Per ulteriori informazioni sull'esecuzione senza containerizzazione, consulta. [the section called “Considerazioni sulla scelta della containerizzazione delle funzioni Lambda”](#)

- Per Timeout, immettere **10 seconds**.
- Per Pinned, scegli True.

Per ulteriori informazioni, consulta [the section called “Configurazione del ciclo di vita”](#).

- In Parametro aggiuntivo, per Accesso in lettura alla directory `/sys`, scegli Abilitato.

b. Per eseguire invece in modalità containerizzata:

Note

Si sconsiglia l'esecuzione in modalità containerizzata a meno che il business case non lo richieda.

- Per Utente e gruppo di sistema, scegli Usa i valori predefiniti del gruppo.
- Per la containerizzazione delle funzioni Lambda, scegli Usa default di gruppo.
- Per Memory limit (Limite memoria), immettere **1024 MB**.
- Per Timeout, immettere **10 seconds**.
- Per Pinned, scegli True.

Per ulteriori informazioni, consulta [the section called “Configurazione del ciclo di vita”](#).

- In Parametri aggiuntivi, per Accesso in lettura alla directory /sys, scegli Abilitato.

2. Scegli Aggiungi funzione Lambda.

Passaggio 5: aggiungere una risorsa modello ottimizzata per SageMaker AI NEO al gruppo Greengrass

In questo passaggio, crea una risorsa per il modello di inferenza ML ottimizzato e caricala in un bucket Amazon S3. Quindi, individua il modello caricato da Amazon S3 nella AWS IoT Greengrass console e associa la risorsa appena creata alla funzione Lambda. In questo modo la funzione potrà accedere alle risorse nel dispositivo core.

1. Nel computer, passare alla directory `resnet50` nel pacchetto di esempio decompresso in [the section called “Creare una funzione Lambda di inferenza”](#).

Note

Se si utilizza l'esempio NVIDIA Jetson, è necessario utilizzare la directory `resnet18` nel pacchetto di esempio. Per ulteriori informazioni, consulta [the section called “Configurazione di un NVIDIA Jetson TX2”](#).

```
cd path-to-downloaded-sample/dlr-py3-armv7l/models/resnet50
```

Questa directory contiene artefatti di modelli precompilati di un modello di classificazione delle immagini basato su Resnet-50.

2. Comprimere i file all'interno della directory `resnet50` in un file denominato `resnet50.zip`.

```
zip -r resnet50.zip .
```

3. Nella pagina di configurazione del AWS IoT Greengrass gruppo, scegli la scheda Risorse. Accedi alla sezione Machine Learning e scegli Add Machine Learning resource (Aggiungi risorsa Machine Learning). Nella pagina Create a Machine Learning resource (Crea una risorsa Machine Learning), per Resource name (Nome risorsa), immetti **resnet50_model**.
4. Per Model source, scegli Usa un modello archiviato in S3, ad esempio un modello ottimizzato tramite Deep Learning Compiler.
5. In URI S3, scegli Browse S3.

Note

Attualmente, i modelli di SageMaker intelligenza artificiale ottimizzati vengono archiviati automaticamente in Amazon S3. Puoi trovare il modello ottimizzato nel tuo bucket Amazon S3 utilizzando questa opzione. Per ulteriori informazioni sull'ottimizzazione dei modelli nell' SageMaker intelligenza artificiale, consulta la documentazione di [SageMaker AI Neo](#).

6. Scegli Upload a model (Carica un modello).
7. Nella scheda della console Amazon S3, carica il file zip in un bucket Amazon S3. Per informazioni, vedi [Come faccio a caricare file e cartelle in un bucket S3?](#) nella Guida per l'utente di Amazon Simple Storage Service.

Note

Il nome del bucket deve contenere la stringa **greengrass**. Scegliere un nome univoco (ad esempio **greengrass-dlr-bucket-user-id-epoch-time**). Non utilizzare un punto (.) nel nome del bucket.

8. Nella scheda AWS IoT Greengrass console, individua e scegli il tuo bucket Amazon S3. Individuare e il file `resnet50.zip` caricato e scegliere Select (Seleziona). Potrebbe essere necessario aggiornare la pagina per aggiornare l'elenco dei bucket e dei file disponibili.
9. In Percorso di destinazione, inserisci. **/ml_model**

Local path

/ml_model

Questa è la destinazione del modello locale nello spazio dei nomi di runtime Lambda. Quando distribuisce il gruppo, AWS IoT Greengrass recupera il pacchetto del modello sorgente e quindi estrae il contenuto nella directory specificata.

Note

Ti consigliamo di utilizzare lo stesso percorso fornito per il percorso locale. Se in questa fase utilizzi un percorso di destinazione del modello locale diverso, alcuni comandi di risoluzione dei problemi specificati in questo tutorial potrebbero risultare imprecisi. Se utilizzi un percorso diverso, dovrai impostare una variabile di ambiente `MODEL_PATH` che utilizzi lo stesso percorso specificato qui. Per informazioni sulle variabili di ambiente, consulta [Variabili di ambiente AWS Lambda](#).

10. Se è in esecuzione in modalità containerizzata:
 - a. In Proprietario del gruppo di sistema e autorizzazioni di accesso ai file, scegli Specificare il gruppo di sistema e le autorizzazioni.
 - b. Scegli Accesso in sola lettura, quindi scegli Aggiungi risorsa.

Fase 6: aggiunta della risorsa del dispositivo della telecamera al gruppo Greengrass

In questo passaggio, crea una risorsa per il modulo fotocamera e associala alla funzione Lambda. Ciò consente alla funzione Lambda di accedere alla risorsa sul dispositivo principale.

 Note

Se si esegue in modalità non containerizzata, è AWS IoT Greengrass possibile accedere alla GPU e alla fotocamera del dispositivo senza configurare questa risorsa del dispositivo.

1. Nella pagina di configurazione del gruppo, scegli la scheda Risorse.
2. Nella scheda Risorse locali, scegli Aggiungi risorsa locale.
3. Nella pagina Aggiungi una risorsa locale, utilizza i seguenti valori:
 - Per Resource Name (Nome risorsa) immetti **videoCoreSharedMemory**.
 - Per Resource type (Tipo di risorsa), scegli Device (Dispositivo).
 - Per Percorso del dispositivo locale, immettere **/dev/vcsm**.

Il percorso del dispositivo è il percorso assoluto locale della risorsa del dispositivo. Questo percorso fa riferimento solo a un dispositivo a caratteri o un dispositivo a blocchi in /dev.

 - Per il proprietario del gruppo di sistema e le autorizzazioni di accesso ai file, scegli Aggiungi automaticamente le autorizzazioni del file system del gruppo di sistema proprietario della risorsa.

L'opzione Group owner file access permission (Autorizzazione per l'accesso al file del proprietario del gruppo) consente di concedere al processo Lambda ulteriori autorizzazioni di accesso ai file. Per ulteriori informazioni, consulta [Autorizzazione di accesso ai file dell'owner del gruppo](#).
4. Nella parte inferiore della pagina, scegli Aggiungi risorsa.
5. Dalla scheda Risorse, crea un'altra risorsa locale scegliendo Aggiungi e utilizza i seguenti valori:
 - Per Resource Name (Nome risorsa) immetti **videoCoreInterface**.
 - Per Resource type (Tipo di risorsa), scegli Device (Dispositivo).
 - Per Percorso del dispositivo locale, immettere **/dev/vchiq**.
 - Per il proprietario del gruppo di sistema e le autorizzazioni di accesso ai file, scegli Aggiungi automaticamente le autorizzazioni del file system del gruppo di sistema proprietario della risorsa.
6. Scegliere Add resource (Aggiungi risorsa).

Fase 7: aggiunta di sottoscrizioni al gruppo Greengrass

In questa fase, si aggiungono le sottoscrizioni al gruppo. Questi abbonamenti consentono alla funzione Lambda di inviare risultati AWS IoT di previsione pubblicandoli su un argomento MQTT.

1. Nella pagina di configurazione del gruppo, scegli la scheda Abbonamenti, quindi scegli Aggiungi abbonamento.
2. Nella pagina Crea un abbonamento, configura l'origine e la destinazione come segue:
 - a. In Tipo di sorgente, scegli Funzione Lambda, quindi scegli. `optimizedImageClassification`
 - b. Nel tipo di Target, scegli Servizio, quindi scegli IoT Cloud.
 - c. Nel filtro Argomento/**resnet-50/predictions**, inserisci, quindi scegli Crea abbonamento.
3. Aggiungere un secondo abbonamento. Scegli la scheda Abbonamenti, scegli Aggiungi abbonamento e configura l'origine e la destinazione come segue:
 - a. In Tipo di origine, scegli Servizi, quindi scegli IoT Cloud.
 - b. In Tipo di destinazione, scegli Funzione Lambda, quindi scegli. `optimizedImageClassification`
 - c. Nel filtro Argomento, inserisci/**resnet-50/test**, quindi scegli Crea abbonamento.

Fase 8: distribuzione del gruppo Greengrass

In questa fase, distribuire la versione corrente della definizione del gruppo nel dispositivo core Greengrass. La definizione contiene la funzione Lambda, le risorse e le configurazioni di sottoscrizione che hai aggiunto.

1. Assicurati che il AWS IoT Greengrass core sia in esecuzione. Esegui i seguenti comandi nel terminale di Raspberry Pi in base alle esigenze.
 - a. Per controllare se il daemon è in esecuzione:

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

Se l'output contiene una voce `root` per `/greengrass/ggc/packages/latest-core-version/bin/daemon`, allora il daemon è in esecuzione.

- b. Per avviare il demone:

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

2. Nella pagina di configurazione del gruppo, scegli Deploy.
3. Nella scheda Funzioni Lambda, seleziona Rilevatore IP e scegli Modifica.
4. Nella finestra di dialogo Modifica impostazioni del rilevatore IP, seleziona Rileva automaticamente e sostituisci gli endpoint del broker MQTT e scegli Salva.

Questo consente ai dispositivi di acquisire automaticamente informazioni di base sulla connettività, come, ad esempio indirizzo IP, DNS e numero della porta. Il rilevamento automatico è consigliato, ma supporta AWS IoT Greengrass anche gli endpoint specificati manualmente. Ti viene chiesto il metodo di individuazione solo la prima volta che il gruppo viene distribuito.

Note

Se richiesto, concedi l'autorizzazione a creare il ruolo di [servizio Greengrass](#) e associarlo al Account AWS tuo ruolo attuale. Regione AWS Questo ruolo consente di accedere AWS IoT Greengrass alle tue risorse nei AWS servizi.

Nella pagina Deployments (Distribuzioni) vengono visualizzati il timestamp della distribuzione, l'ID versione e lo stato. Una volta completata, lo stato visualizzato per la distribuzione dovrebbe essere Completato.

Per ulteriori informazioni sulle implementazioni, consulta [Distribuisci gruppi AWS IoT Greengrass](#). Per la risoluzione dei problemi, consultare [Risoluzione dei problemi](#).

Esecuzione del test dell'esempio di inferenza

Ora puoi verificare se la distribuzione è configurata correttamente. Per eseguire il test, devi abbonarti all'argomento `/resnet-50/predictions` e pubblicare eventuali messaggi nell'argomento `/resnet-50/test`. Questo attiva la funzione Lambda per scattare una foto con il tuo Raspberry Pi ed eseguire inferenze sull'immagine che cattura.

Note

Se si utilizza l'esempio di NVIDIA Jetson, assicurarsi di utilizzare invece gli argomenti `resnet-18/predictions` e `resnet-18/test`.

Note

Se a dispositivo Raspberry Pi è collegato un monitor, il segnale attivo della telecamera viene visualizzato in una finestra di anteprima.

1. Nella home page della AWS IoT console, in Test, scegli MQTT test client.
2. Per gli abbonamenti, scegliete Iscriviti a un argomento. Utilizzare i seguenti valori. Non modificare i valori predefiniti delle altre opzioni:
 - Per Argomento sottoscrizione, immetti **/resnet-50/predictions**.
 - In Configurazione aggiuntiva, per la visualizzazione del payload MQTT, scegli Visualizza i payload come stringhe.
3. Scegliere Subscribe (Effettua sottoscrizione).
4. Scegliete Pubblica su un argomento, immettetelo **/resnet-50/test** come nome dell'argomento e scegliete Pubblica.
5. Se il test viene completato senza errori, il messaggio pubblicato richiederà a Raspberry Pi di acquisire un'immagine. Nella parte inferiore della pagina viene visualizzato un messaggio della funzione Lambda. Questo messaggio contiene il risultato predittivo dell'immagine nel formato: nome classe prevista, probabilità e picco di utilizzo della memoria.

Configurazione di un dispositivo Intel Atom

Per eseguire questo tutorial su un dispositivo Intel Atom, è necessario fornire immagini di origine, configurare la funzione Lambda e aggiungere un'altra risorsa del dispositivo locale. Per utilizzare la GPU per l'inferenza, assicurarsi che sul dispositivo sia installato il seguente software:

- OpenCL versione 1.0 o successiva
- Python 3.7 e pip
- [NumPy](#)

- [OpenCV su Wheels](#)

1. Scarica immagini PNG o JPG statiche per la funzione Lambda da utilizzare per la classificazione delle immagini. L'esempio funziona in modo ottimale con file immagine di dimensioni ridotte.

Salva i file immagine nella directory contenente il file `inference.py` (o in una sottodirectory di questa directory). Si trova nel pacchetto di distribuzione della funzione Lambda in cui carichi. [the section called “Creare una funzione Lambda di inferenza”](#)

 Note

Se la utilizzi AWS DeepLens, puoi utilizzare la videocamera di bordo o montare la tua fotocamera per eseguire inferenze sulle immagini acquisite anziché sulle immagini statiche. Tuttavia, ti consigliamo di iniziare con le immagini statiche.

Se si utilizza una telecamera, assicurarsi che il pacchetto APT `awscam` sia installato e aggiornato. Per ulteriori informazioni, consulta [Aggiorna il tuo AWS DeepLens dispositivo nella Guida](#) per gli AWS DeepLens sviluppatori.

2. Modifica la configurazione della funzione Lambda. Segui la procedura riportata in [the section called “Aggiungere la funzione Lambda al gruppo”](#).

 Note

Ti consigliamo di eseguire la funzione Lambda senza containerizzazione a meno che il tuo business case non lo richieda. Ciò consente l'accesso alla GPU e alla fotocamera del dispositivo senza configurare le risorse del dispositivo. Se esegui senza containerizzazione, devi anche concedere l'accesso root alle tue funzioni Lambda AWS IoT Greengrass .

- a. Per eseguire senza containerizzazione:

- Per Utente e gruppo di sistema, scegli. **Another user ID/group ID** Per ID utente di sistema, immettere `0`. Per ID del gruppo di sistema, immettere `0`.

Ciò consente alla funzione Lambda di funzionare come root. Per ulteriori informazioni sull'esecuzione come root, consulta [the section called “Impostazione dell'identità di accesso predefinita per le funzioni Lambda in un gruppo”](#).

 Tip

È inoltre necessario aggiornare il `config.json` file per concedere l'accesso root alla funzione Lambda. Per la procedura, vedi [the section called “Esecuzione di una funzione Lambda come utente root”](#).

- Per la containerizzazione della funzione Lambda, scegli Nessun contenitore.

Per ulteriori informazioni sull'esecuzione senza containerizzazione, consulta [the section called “Considerazioni sulla scelta della containerizzazione delle funzioni Lambda”](#)

- Aumenta il valore di Timeout a 2 minuti. In questo modo, il timeout della richiesta non viene eseguito troppo presto. L'esecuzione dell'inferenza richiede alcuni minuti dopo la configurazione.
- Per Pinned, scegli True.
- In Parametri aggiuntivi, per Accesso in lettura alla directory `/sys`, scegli Abilitato.

b. Per eseguire invece in modalità containerizzata:

 Note

Si sconsiglia l'esecuzione in modalità containerizzata a meno che il business case non lo richieda.

- Aumenta il valore di Memory limit (Limite memoria) a 3000 MB.
- Aumenta il valore di Timeout a 2 minuti. In questo modo, il timeout della richiesta non viene eseguito troppo presto. L'esecuzione dell'inferenza richiede alcuni minuti dopo la configurazione.
- Per Pinned, scegli True.
- In Parametri aggiuntivi, per Accesso in lettura alla directory `/sys`, scegli Abilitato.

3. Aggiungere la risorsa del modello ottimizzato Neo al gruppo Caricare le risorse del modello nella directory `resnet50` del pacchetto di esempio decompresso in [the section called “Creare una funzione Lambda di inferenza”](#). Questa directory contiene artefatti di modelli precompilati di un modello di classificazione delle immagini basato su Resnet-50. Segui la procedura descritta in [the section called “Aggiunta della risorsa del modello ottimizzato Neo al gruppo”](#), con i seguenti aggiornamenti:

- Comprimere i file all'interno della directory `resnet50` in un file denominato `resnet50.zip`.
 - Nella pagina Create a Machine Learning resource (Crea una risorsa Machine Learning), per Resource name (Nome risorsa), immetti **resnet50_model**.
 - Caricare il file `resnet50.zip`
4. Se è in esecuzione in modalità containerizzata, aggiungi la risorsa del dispositivo locale richiesta per concedere l'accesso alla GPU del dispositivo.

Note

Se esegui in modalità non containerizzata, AWS IoT Greengrass puoi accedere alla GPU del dispositivo senza configurare le risorse del dispositivo.

- Nella pagina di configurazione del gruppo, scegli la scheda Risorse.
- Nella sezione Risorse locali, scegli Aggiungi risorsa locale.
- Definisci la risorsa:
 - Per Resource Name (Nome risorsa) immetti **renderD128**.
 - Per Resource type (Tipo di risorsa), scegli Device (Dispositivo).
 - Per Percorso del dispositivo locale, immettere **/dev/dri/renderD128**.
 - Per il proprietario del gruppo di sistema e le autorizzazioni di accesso ai file, scegli Aggiungi automaticamente le autorizzazioni del file system del gruppo di sistema proprietario della risorsa.

Configurazione di un NVIDIA Jetson TX2

Per eseguire questo tutorial su un NVIDIA Jetson TX2, fornisci immagini sorgente, configura la funzione Lambda e aggiungi altre risorse locali del dispositivo.

1. Assicurati che il tuo dispositivo Jetson sia configurato in modo da poter installare il software AWS IoT Greengrass Core e utilizzare la GPU per l'inferenza. Per ulteriori informazioni sulla configurazione del dispositivo, consulta [the section called "Configurazione di altri dispositivi"](#). Per utilizzare la GPU per l'inferenza su un NVIDIA Jetson TX2, devi installare CUDA 10.0 e cuDNN 7.0 sul tuo dispositivo quando immagini la tua scheda con Jetpack 4.3.

2. Scarica immagini PNG o JPG statiche per la funzione Lambda da utilizzare per la classificazione delle immagini. L'esempio funziona in modo ottimale con file immagine di dimensioni ridotte.

Salvare i file immagine nella directory contenente il file `inference.py`. È possibile salvarli anche in una sottodirectory di questa directory. Questa directory si trova nel pacchetto di distribuzione della funzione Lambda in cui carichi. [the section called “Creare una funzione Lambda di inferenza”](#)

 Note

In alternativa, puoi scegliere di implementare una telecamera sulla scheda Jetson per acquisire le immagini di origine. Tuttavia, ti consigliamo di iniziare con le immagini statiche.

3. Modifica la configurazione della funzione Lambda. Segui la procedura riportata in [the section called “Aggiungere la funzione Lambda al gruppo”](#).

 Note

Ti consigliamo di eseguire la funzione Lambda senza containerizzazione a meno che il tuo business case non lo richieda. Ciò consente l'accesso alla GPU e alla fotocamera del dispositivo senza configurare le risorse del dispositivo. Se esegui senza containerizzazione, devi anche concedere l'accesso root alle tue funzioni Lambda AWS IoT Greengrass .

a. Per eseguire senza containerizzazione:

- Per Esegui come, scegli. **Another user ID/group ID** Per UID, inserisci **0**. Per GUID, immettere. **0**

Ciò consente alla funzione Lambda di funzionare come root. Per ulteriori informazioni sull'esecuzione come root, consulta [the section called “Impostazione dell'identità di accesso predefinita per le funzioni Lambda in un gruppo”](#).

 Tip

È inoltre necessario aggiornare il `config.json` file per concedere l'accesso root alla funzione Lambda. Per la procedura, vedi [the section called “Esecuzione di una funzione Lambda come utente root”](#).

- Per la containerizzazione della funzione Lambda, scegli Nessun contenitore.

Per ulteriori informazioni sull'esecuzione senza containerizzazione, consulta [the section called “Considerazioni sulla scelta della containerizzazione delle funzioni Lambda”](#)

- Aumenta il valore Timeout a 5 minuti. In questo modo, il timeout della richiesta non viene eseguito troppo presto. L'esecuzione dell'inferenza richiede alcuni minuti dopo la configurazione.
- Per Pinned, scegli True.
- In Parametri aggiuntivi, per Accesso in lettura alla directory `/sys`, scegli Abilitato.

b. Per eseguire invece in modalità containerizzata:

 Note

Si sconsiglia l'esecuzione in modalità containerizzata a meno che il business case non lo richieda.

- Aumenta il valore di Memory limit (Limite memoria). Per utilizzare il modello fornito in modalità GPU, utilizza almeno 2000 MB.
- Aumenta il valore Timeout a 5 minuti. In questo modo, il timeout della richiesta non viene eseguito troppo presto. L'esecuzione dell'inferenza richiede alcuni minuti dopo la configurazione.
- Per Pinned, scegli True.
- In Parametri aggiuntivi, per Accesso in lettura alla directory `/sys`, scegli Abilitato.

4. Aggiungere la risorsa del modello ottimizzato Neo al gruppo Caricare le risorse del modello nella directory `resnet18` del pacchetto di esempio decompresso in [the section called “Creare una funzione Lambda di inferenza”](#). Questa directory contiene artefatti di modelli precompilati di un modello di classificazione delle immagini basato su Resnet-18. Segui la procedura descritta in

[the section called “Aggiunta della risorsa del modello ottimizzato Neo al gruppo”](#), con i seguenti aggiornamenti:

- Comprimere i file all'interno della directory `resnet18` in un file denominato `resnet18.zip`.
 - Nella pagina **Create a Machine Learning resource** (Crea una risorsa Machine Learning), per **Resource name** (Nome risorsa), immetti **resnet18_model**.
 - Caricare il file `resnet18.zip`
5. Se è in esecuzione in modalità containerizzata, aggiungi le risorse del dispositivo locale necessarie per concedere l'accesso alla GPU del dispositivo.

 Note

Se esegui in modalità non containerizzata, AWS IoT Greengrass puoi accedere alla GPU del dispositivo senza configurare le risorse del dispositivo.

- Nella pagina di configurazione del gruppo, scegli la scheda **Risorse**.
- Nella sezione **Risorse locali**, scegli **Aggiungi risorsa locale**.
- Definisci ogni risorsa:
 - Per **Resource name** (Nome risorsa) e **Device path** (Percorso dispositivo), utilizza i valori nella tabella seguente. Crea una risorsa del dispositivo per ogni riga nella tabella.
 - Per **Resource type** (Tipo di risorsa), scegli **Device** (Dispositivo).
 - Per il proprietario del gruppo di sistema e le autorizzazioni di accesso ai file, scegli **Aggiungi automaticamente le autorizzazioni del file system del gruppo di sistema proprietario della risorsa**.

Nome	Percorso dispositivo
<code>nvhost-ctrl</code>	<code>/dev/nvhost-ctrl</code>
<code>nvhost-gpu</code>	<code>/dev/nvhost-gpu</code>
<code>nvhost-ctrl-gpu</code>	<code>/dev/ nvhost-ctrl-gpu</code>

Nome	Percorso dispositivo
nvhost-dbg-gpu	/dev/ nvhost-dbg-gpu
nvhost-prof-gpu	/dev/ nvhost-prof-gpu
nvmap	/dev/nvmap
nvhost-vic	/dev/nvhost-vic
tegra_dc_ctrl	/dev/tegra_dc_ctrl

6. Se è in esecuzione in modalità containerizzata, aggiungi la seguente risorsa di volume locale per concedere l'accesso alla fotocamera del tuo dispositivo. Segui la procedura riportata in [the section called “Aggiunta della risorsa del modello ottimizzato Neo al gruppo”](#).

 Note

Se si esegue in modalità non containerizzata, è AWS IoT Greengrass possibile accedere alla fotocamera del dispositivo senza configurare le risorse del dispositivo.

- Per Resource type (Tipo di risorsa), scegli Volume.
- Per il proprietario del gruppo di sistema e le autorizzazioni di accesso ai file, scegli Aggiungi automaticamente le autorizzazioni del file system del gruppo di sistema proprietario della risorsa.

Nome	Percorso di origine	Percorso di destinazione
shm	/dev/shm	/dev/shm
tmp	/tmp	/tmp

7. Aggiornare le sottoscrizioni di gruppo per utilizzare la directory corretta. Segui la procedura descritta in [the section called “Aggiunta di sottoscrizioni al gruppo”](#), con i seguenti aggiornamenti:
- Per il primo filtro di argomenti, immettere **/resnet-18/predictions**.

- Per il secondo filtro di argomenti, immettere **/resnet-18/test**.
8. Aggiornare le sottoscrizioni di test per utilizzare la directory corretta. Segui la procedura descritta in [the section called “Test dell'esempio”](#), con i seguenti aggiornamenti:
 - Per Abbonamenti, scegli Sottoscrivi a un argomento. Per Argomento sottoscrizione, immetti **/resnet-18/predictions**.
 - Nella pagina /resnet-18/predictions, specificare l'argomento /resnet-18/test in cui effettuare la pubblicazione.

Risoluzione dei problemi di AWS IoT Greengrass inferenza ML

Se il test non viene completato correttamente, puoi provare a eseguire la procedura di risoluzione dei problemi riportata di seguito. Esegui i comandi nel terminale Raspberry Pi.

Controlla i log degli errori

1. Passare all'utente root e navigare alla directory log. L'accesso ai AWS IoT Greengrass log richiede i permessi di root.

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

2. Verifica la presenza `runtime.log` di eventuali errori.

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

Puoi anche cercare eventuali errori nel registro delle funzioni Lambda definito dall'utente:

```
cat user/your-region/your-account-id/lambda-function-name.log | grep 'ERROR'
```

Per ulteriori informazioni, consulta [the section called “Risoluzione dei problemi con i log”](#).

Verifica che la funzione Lambda sia implementata correttamente

1. Elenca il contenuto della Lambda distribuita nella /lambda directory. Prima di eseguire il comando, sostituisci i valori dei segnaposti.

```
cd /greengrass/ggc/deployment/lambda/  
arn:aws:lambda:region:account:function:function-name:function-version  
ls -la
```

2. Verifica che la directory contenga gli stessi file inclusi nel pacchetto di distribuzione `optimizedImageClassification.zip` caricato in [Fase 3: Creare una funzione Lambda di inferenza](#).

Assicurati inoltre che i file `.py` e le dipendenze si trovino nella root della directory.

Verifica che il modello di inferenza sia stato distribuito correttamente

1. Trova il numero di identificazione del processo (PID) del processo di runtime Lambda:

```
ps aux | grep lambda-function-name
```

Nell'output, il PID appare nella seconda colonna della riga per il processo di runtime Lambda.

2. Inserisci lo spazio dei nomi di runtime Lambda. Assicurati di sostituire il *pid* valore segnaposto prima di eseguire il comando.

Note

Questa directory e il suo contenuto si trovano nello spazio dei nomi di runtime Lambda, quindi non sono visibili in un normale spazio dei nomi Linux.

```
sudo nsenter -t pid -m /bin/bash
```

3. Elenca i contenuti della directory locale specificata per la risorsa ML.

Note

Se il percorso della risorsa ML è diverso da `ml_model`, dovrai sostituirlo qui.

```
cd /ml_model  
ls -ls
```

Dovrebbero essere visualizzati i seguenti file:

```
56 -rw-r--r-- 1 ggc_user ggc_group 56703 Oct 29 20:07 model.json  
196152 -rw-r--r-- 1 ggc_user ggc_group 200855043 Oct 29 20:08 model.params  
256 -rw-r--r-- 1 ggc_user ggc_group 261848 Oct 29 20:07 model.so  
32 -rw-r--r-- 1 ggc_user ggc_group 30564 Oct 29 20:08 synset.txt
```

La funzione Lambda non trova **/dev/dri/renderD128**

Questa situazione può verificarsi se OpenCL non è in grado di connettersi ai dispositivi GPU necessari. È necessario creare risorse di dispositivo per i dispositivi necessari per la funzione Lambda.

Passaggi successivi

Esplorare altri modelli ottimizzati. Per informazioni, consulta la [documentazione di SageMaker AI Neo](#).

Gestisci i flussi di dati sul core AWS IoT Greengrass

AWS IoT Greengrass stream manager semplifica e rende più affidabile il trasferimento di dati IoT ad alto volume a. Cloud AWS Stream Manager elabora i flussi di dati localmente e li esporta automaticamente. Cloud AWS Questa funzionalità si integra con gli scenari edge più comuni, come l'inferenza di machine learning (ML), in cui i dati vengono elaborati e analizzati localmente prima di essere esportati verso le destinazioni di archiviazione Cloud AWS o locali.

Stream manager semplifica lo sviluppo di applicazioni. Le applicazioni IoT possono utilizzare un meccanismo standardizzato per elaborare flussi a elevato volume e gestire policy di conservazione dei dati locali anziché creare funzionalità di gestione dei flussi personalizzate. Le applicazioni IoT possono leggere e scrivere nei flussi. Possono definire policy per tipo di storage, dimensioni e conservazione dei dati in base al flusso per controllare in che modo stream manager elabora ed esporta i flussi.

Stream manager è progettato per funzionare in ambienti con connettività intermittente o limitata. Puoi definire l'utilizzo della larghezza di banda, il comportamento di timeout e la modalità di gestione dei dati del flusso quando il core è connesso o disconnesso. Per i dati critici, è possibile impostare le priorità per controllare l'ordine in cui i flussi vengono esportati verso. Cloud AWS

È possibile configurare le esportazioni automatiche verso Cloud AWS l'archiviazione o l'ulteriore elaborazione e analisi. Stream Manager supporta l'esportazione verso le seguenti Cloud AWS destinazioni.

- Canali in AWS IoT Analytics. AWS IoT Analytics consente di eseguire analisi avanzate sui dati per aiutare a prendere decisioni aziendali e migliorare i modelli di apprendimento automatico. Per ulteriori informazioni, consulta [Cos'è AWS IoT Analytics?](#) nella Guida AWS IoT Analytics per l'utente.
- Stream in Kinesis Data Streams. Kinesis Data Streams viene comunemente utilizzato per aggregare dati di grandi volumi e caricarli in un data warehouse o in un cluster di riduzione delle mappe. Per ulteriori informazioni, consulta [Cos'è Amazon Kinesis Data Streams?](#) nella Amazon Kinesis Developer Guide.
- Proprietà delle risorse in AWS IoT SiteWise. AWS IoT SiteWise consente di raccogliere, organizzare e analizzare i dati provenienti da apparecchiature industriali su larga scala. Per ulteriori informazioni, consulta [Cos'è AWS IoT SiteWise?](#) nella Guida AWS IoT SiteWise per l'utente.

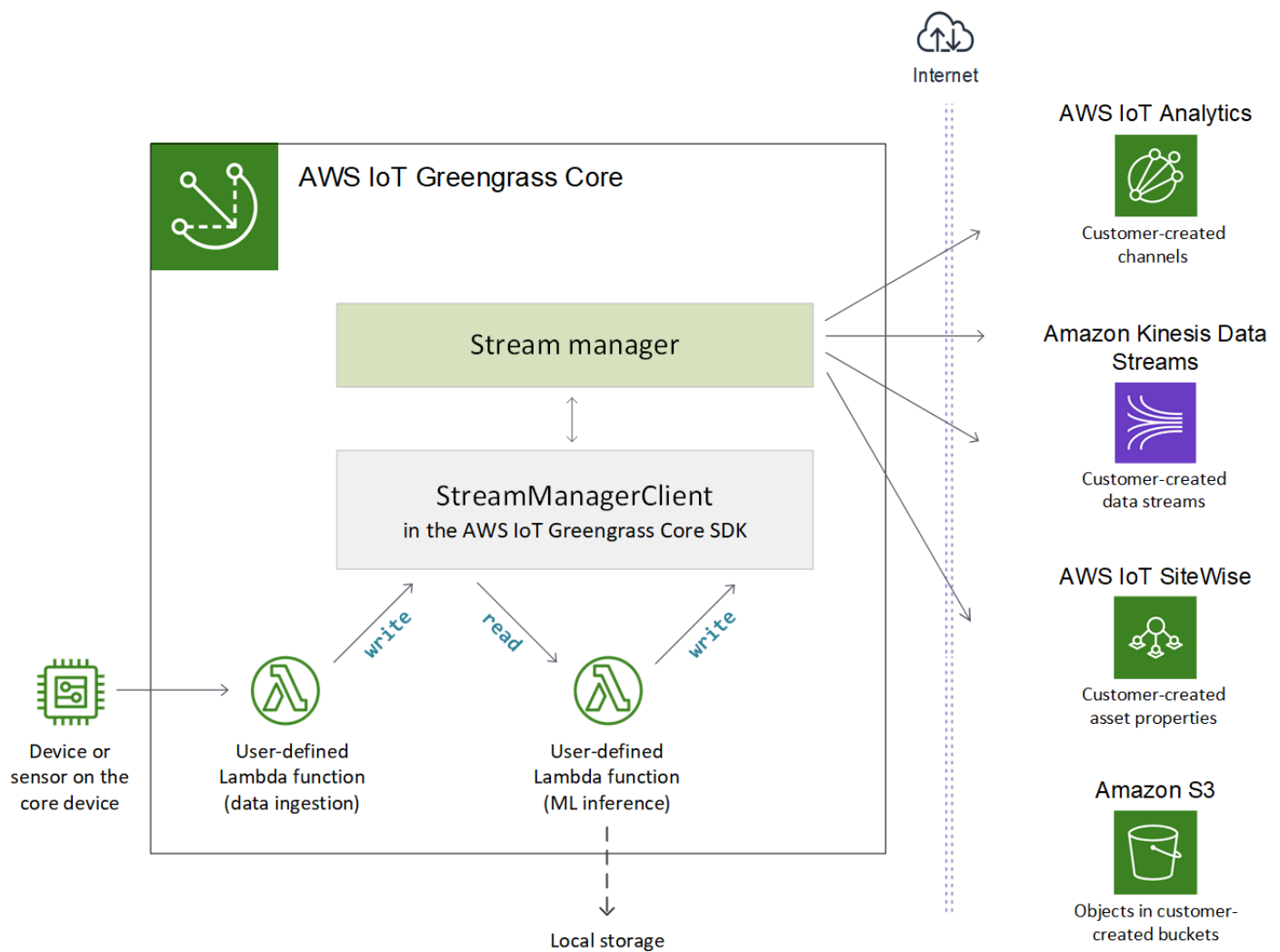
- Oggetti in Amazon S3. Puoi usare Amazon S3 per archiviare e recuperare grandi quantità di dati. Per ulteriori informazioni, consulta [Cos'è Amazon S3?](#) nella Guida per sviluppatori di Amazon Simple Storage Service.

Flusso di lavoro della gestione dei flussi

Le tue applicazioni IoT interagiscono con lo stream manager tramite AWS IoT Greengrass Core SDK. In un flusso di lavoro semplice, una funzione Lambda definita dall'utente in esecuzione sul core Greengrass consuma dati IoT, come le metriche di temperatura e pressione delle serie temporali. La funzione Lambda potrebbe filtrare o comprimere i dati e quindi chiamare AWS IoT Greengrass Core SDK per scrivere i dati in uno stream in stream manager. Lo stream manager può esportare lo stream Cloud AWS automaticamente, in base alle politiche definite per lo stream. Le funzioni Lambda definite dall'utente possono anche inviare dati direttamente ai database locali o agli archivi di archiviazione.

Le tue applicazioni IoT possono includere più funzioni Lambda definite dall'utente che leggono o scrivono negli stream. Queste funzioni Lambda locali possono leggere e scrivere negli stream per filtrare, aggregare e analizzare i dati localmente. Questo consente di rispondere rapidamente a eventi locali ed estrarre informazioni utili prima che i dati vengano trasferiti dal core nel cloud o in destinazioni locali.

Un flusso di lavoro di esempio è mostrato nel diagramma seguente.



Per utilizzare lo stream manager, inizia configurando i parametri dello stream manager per definire le impostazioni di runtime a livello di gruppo che si applicano a tutti gli stream sul core Greengrass. Queste impostazioni personalizzabili consentono di controllare il modo in cui lo stream manager archivia, elabora ed esporta i flussi in base alle esigenze aziendali e ai vincoli ambientali. Per ulteriori informazioni, consulta [the section called “Configurazione di Stream Manager”](#).

Dopo aver configurato lo stream manager, puoi creare e distribuire le tue applicazioni IoT. Si tratta in genere di funzioni Lambda definite dall'utente che vengono `StreamManagerClient` utilizzate AWS IoT Greengrass nell'SDK Core per creare e interagire con gli stream. Durante la creazione dello stream, la funzione Lambda definisce le policy per flusso, come le destinazioni di esportazione, la priorità e la persistenza. Per ulteriori informazioni, inclusi frammenti di codice per le operazioni, consulta `StreamManagerClient` [the section called “Utilizzalo StreamManagerClient per lavorare con gli stream”](#)

Per i tutorial che configurano un flusso di lavoro semplice, consulta o [the section called “Esportazione di flussi di dati \(Console\)”](#) [the section called “Esportazione di flussi di dati \(CLI\)”](#)

Requisiti

Per l'utilizzo di stream manager si applicano i seguenti requisiti:

- È necessario utilizzare il software AWS IoT Greengrass Core v1.10 o versione successiva, con lo stream manager abilitato. Per ulteriori informazioni, consulta [the section called “Configurazione di Stream Manager”](#).

Lo stream manager non è supportato nelle distribuzioni. OpenWrt

- Java 8 Runtime (JDK 8) deve essere installato sul core.
 - Per distribuzioni basate su Debian (incluso Raspbian) o distribuzioni basate su Ubuntu, eseguire il comando seguente:

```
sudo apt install openjdk-8-jdk
```

- Per distribuzioni basate su Red Hat (incluso Amazon Linux), eseguire il comando seguente:

```
sudo yum install java-1.8.0-openjdk
```

Per ulteriori informazioni, consulta [How to download and install prebuilt OpenJDK packages](#) nella documentazione di OpenJDK.

- Stream Manager richiede un minimo di 70 MB di RAM in aggiunta al software AWS IoT Greengrass Core di base. Il requisito di memoria totale dipende dal carico di lavoro.
- Le funzioni Lambda definite dall'utente devono utilizzare [AWS IoT Greengrass Core](#) SDK per interagire con lo stream manager. Il AWS IoT Greengrass Core SDK è disponibile in diverse lingue, ma solo le seguenti versioni supportano le operazioni di gestione degli stream:
 - Java SDK (v1.4.0 o versione successiva)
 - Python SDK (v1.5.0 o successivo)
 - Node.js SDK (v1.6.0 o successivo)

Scarica la versione dell'SDK che corrisponde al runtime della tua funzione Lambda e includila nel pacchetto di distribuzione della funzione Lambda.

Note

Il AWS IoT Greengrass Core SDK per Python richiede Python 3.7 o versione successiva e ha altre dipendenze dai pacchetti. Per ulteriori informazioni, consulta [Creare un pacchetto di distribuzione di funzioni Lambda \(console\)](#) o [Creare un pacchetto di distribuzione di funzioni Lambda \(CLI\)](#).

- Se definisci le destinazioni di Cloud AWS esportazione per uno stream, devi creare i tuoi obiettivi di esportazione e concedere le autorizzazioni di accesso nel ruolo del gruppo Greengrass. A seconda della destinazione, potrebbero applicarsi anche altri requisiti. Per ulteriori informazioni, consultare:
 - [the section called “AWS IoT Analytics canali”](#)
 - [the section called “Flussi di dati Amazon Kinesis”](#)
 - [the section called “AWS IoT SiteWise proprietà degli asset”](#)
 - [the section called “Oggetti Amazon S3”](#)

L'utente è responsabile del mantenimento di queste Cloud AWS risorse.

Sicurezza dei dati

Quando utilizzi stream manager, tiene presente le seguenti considerazioni di sicurezza.

Sicurezza dei dati locali

AWS IoT Greengrass non crittografa i dati di flusso inattivi o in transito localmente tra i componenti del dispositivo principale.

- Dati inattivi. I dati di streaming vengono archiviati localmente in una directory di archiviazione sul core Greengrass. Per la sicurezza dei dati, AWS IoT Greengrass si affida alle autorizzazioni dei file Unix e alla crittografia dell'intero disco, se abilitata. Puoi utilizzare il parametro [STREAM_MANAGER_STORE_ROOT_DIR](#) opzionale per specificare la directory di storage. Se si modifica questo parametro in un secondo momento per utilizzare una directory di archiviazione diversa, AWS IoT Greengrass non elimina la directory di archiviazione precedente o il relativo contenuto.

- **Dati in transito a livello locale.** AWS IoT Greengrass non crittografa i dati di flusso in transito locale sul core tra fonti di dati, funzioni Lambda, AWS IoT Greengrass Core SDK e stream manager.
- **Dati in transito verso Cloud AWS.** I flussi di dati esportati da Stream Manager Cloud AWS utilizzano la crittografia client di AWS servizio standard con Transport Layer Security (TLS).

Per ulteriori informazioni, consulta [the section called “Crittografia dei dati”](#).

Autenticazione client

I client Stream Manager utilizzano AWS IoT Greengrass Core SDK per comunicare con lo stream manager. Quando l'autenticazione client è abilitata, solo le funzioni Lambda del gruppo Greengrass possono interagire con gli stream in stream manager. Quando l'autenticazione client è disabilitata, qualsiasi processo in esecuzione sul core Greengrass (ad esempio i [container Docker](#)) può interagire con i flussi in stream manager. È opportuno disabilitare l'autenticazione solo se richiesto dal business case.

Utilizza il parametro [STREAM_MANAGER_AUTHENTICATE_CLIENT](#) per impostare la modalità di autenticazione client. È possibile configurare questo parametro dalla console o dall'API. AWS IoT Greengrass Le modifiche diventano effettive dopo la distribuzione del gruppo.

	Abilitato	Disabilitato
Valore del parametro	true (predefinito e consigliato)	false
Client consentiti	Funzioni Lambda definite dall'utente nel gruppo Greengrass	Funzioni Lambda definite dall'utente nel gruppo Greengrass Altri processi in esecuzione sul dispositivo core Greengrass

Consulta anche

- [the section called “Configurazione di Stream Manager”](#)
- [the section called “Utilizzalo StreamManagerClient per lavorare con gli stream”](#)
- [the section called “Esporta le configurazioni per le destinazioni supportate Cloud AWS”](#)
- [the section called “Esportazione di flussi di dati \(Console\)”](#)
- [the section called “Esportazione di flussi di dati \(CLI\)”](#)

Configurare AWS IoT Greengrass lo stream manager

AWS IoT Greengrass Fondamentalmente, lo stream manager può archiviare, elaborare ed esportare i dati dei dispositivi IoT. Stream manager fornisce i parametri utilizzati per configurare le impostazioni di runtime a livello del gruppo. Queste impostazioni si applicano a tutti i flussi sul core Greengrass. Puoi utilizzare la AWS IoT console o l' AWS IoT Greengrass API per configurare le impostazioni dello stream manager. Le modifiche diventano effettive dopo la distribuzione del gruppo.

Note

Dopo aver configurato lo stream manager, puoi creare e distribuire applicazioni IoT che funzionano sul core Greengrass e interagiscono con lo stream manager. Queste applicazioni IoT sono in genere funzioni Lambda definite dall'utente. Per ulteriori informazioni, consulta [the section called “Utilizzalo StreamManagerClient per lavorare con gli stream”](#).

Parametri di Stream Manager

Stream manager fornisce i seguenti parametri che consentono di definire le impostazioni a livello del gruppo. Tutti i parametri sono opzionali:

Directory di storage

Nome parametro: `STREAM_MANAGER_STORE_ROOT_DIR`

Il percorso assoluto della directory locale utilizzata per archiviare i flussi. Questo valore deve iniziare con una barra (ad esempio, `/data`).

Per informazioni relative alla protezione dei dati del flusso, consulta [the section called “Sicurezza dei dati locali”](#).

Versione minima AWS IoT Greengrass Core: 1.10.0

Porta del server

Nome parametro: `STREAM_MANAGER_SERVER_PORT`

Il numero di porta locale utilizzato per comunicare con stream manager. Il valore predefinito è 8088.

Versione minima del AWS IoT Greengrass core: 1.10.0

Autentica client

Nome parametro: `STREAM_MANAGER_AUTHENTICATE_CLIENT`

Indica se i client devono essere autenticati per interagire con stream manager. Tutte le interazioni tra i client e lo stream manager sono controllate dal AWS IoT Greengrass Core SDK. Questo parametro determina quali client possono chiamare il AWS IoT Greengrass Core SDK per lavorare con gli stream. Per ulteriori informazioni, consulta [the section called “Autenticazione client”](#).

I valori validi sono `true` e `false`. Il valore predefinito è `true` (consigliato).

- `true`. Consente solo le funzioni Greengrass Lambda come client. I client con funzioni Lambda utilizzano i protocolli di AWS IoT Greengrass base interni per l'autenticazione con Core SDK. AWS IoT Greengrass
- `false`. Consente a qualsiasi processo in esecuzione sul AWS IoT Greengrass core di essere un client. Non impostare su `false` meno che non venga richiesto dal business case. Ad esempio, imposta questo valore `false` solo se i processi non Lambda sul dispositivo principale devono comunicare direttamente con lo stream manager, come i [contenitori Docker](#) in esecuzione sul core.

Versione minima AWS IoT Greengrass Core: 1.10.0

Larghezza di banda massima

Nome parametro: `STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH`

La larghezza di banda massima media (in kilobit al secondo) che può essere utilizzata per esportare i dati. L'impostazione predefinita consente l'uso illimitato della larghezza di banda disponibile.

Versione minima del AWS IoT Greengrass core: 1.10.0

Dimensione del pool di thread

Nome parametro: `STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE`

Il numero massimo di thread attivi che possono essere utilizzati per esportare i dati. Il valore predefinito è 5.

La dimensione ottimale dipende dall'hardware, dal volume del flusso e dal numero pianificato di flussi di esportazione. Se la velocità di esportazione è bassa, puoi regolare questa impostazione per trovare la dimensione ottimale per l'hardware e il business case. La CPU e la memoria dell'hardware del dispositivo core sono fattori limitanti. Per iniziare, è possibile provare a impostare questo valore uguale al numero di core di processore sul dispositivo.

Fare attenzione a non impostare una dimensione superiore a quella supportata dall'hardware. Ogni flusso consuma risorse hardware, pertanto occorre provare a limitare il numero di flussi di esportazione sui dispositivi vincolati.

Versione minima del AWS IoT Greengrass core: 1.10.0

Argomenti JVM

Nome parametro: `JVM_ARGS`

Argomenti Java Virtual Machine personalizzati da passare a Stream Manager all'avvio. Più argomenti devono essere separati da spazi.

Utilizza questo parametro solo quando devi sostituire le impostazioni predefinite utilizzate dalla JVM. Ad esempio, potrebbe essere necessario aumentare la dimensione heap predefinita se prevedi di esportare un numero elevato di flussi.

Versione minima del AWS IoT Greengrass core: 1.10.0

Directory di file di input di sola lettura

Nome parametro: `STREAM_MANAGER_READ_ONLY_DIRS`

Un elenco separato da virgole di percorsi assoluti alle directory esterne al file system root che memorizzano i file di input. Stream Manager legge e carica i file su Amazon S3 e monta le directory come di sola lettura. Per ulteriori informazioni sull'esportazione in Amazon S3, consulta [the section called “Oggetti Amazon S3”](#)

Utilizza questo parametro solo se sono vere le seguenti condizioni:

- La directory dei file di input per uno stream che esporta in Amazon S3 si trova in una delle seguenti posizioni:
 - Una partizione diversa dal file system root.
 - Si /tmp trova nel file system root.
- La [containerizzazione predefinita](#) del gruppo Greengrass è il contenitore Greengrass.

Valore di esempio: /mnt/directory-1,/mnt/directory-2,/tmp

Versione minima Core: 1.11.0 AWS IoT Greengrass

Dimensione minima per il caricamento in più parti

Nome parametro:

STREAM_MANAGER_EXPORTER_S3_DESTINATION_MULTIPART_UPLOAD_MIN_PART_SIZE_BYTES

La dimensione minima (in byte) di una parte in un caricamento multiparte su Amazon S3. Stream Manager utilizza questa impostazione e la dimensione del file di input per determinare come raggruppare i dati in una richiesta PUT composta da più parti. Il valore minimo e predefinito è di 5242880 byte (5 MB).

Note

Stream Manager utilizza la `sizeThresholdForMultipartUploadBytes` proprietà dello stream per determinare se esportare in Amazon S3 come caricamento singolo o multiparte. Le funzioni Lambda definite dall'utente impostano questa soglia quando creano uno stream che esporta in Amazon S3. La soglia predefinita è 5 MB.

Versione minima AWS IoT Greengrass del core: 1.11.0

Configurazione delle impostazioni di Stream Manager (console)

È possibile utilizzare la AWS IoT console per le seguenti attività di gestione:

- [Verificare se Stream Manager è abilitato](#)
- [Abilitazione o disabilitazione di Stream Manager durante la creazione del gruppo](#)
- [Abilitazione o disabilitazione di Stream Manager per un gruppo esistente](#)

- [Modifica delle impostazioni di Stream Manager](#)

Le modifiche diventano effettive dopo la distribuzione del gruppo Greengrass. Per un tutorial che mostra come implementare un gruppo Greengrass che contiene una funzione Lambda che interagisce con lo stream manager, vedi. [the section called “Esportazione di flussi di dati \(Console\)”](#)

 Note

Quando si utilizza la console per abilitare lo stream manager e distribuire il gruppo, la dimensione della memoria per lo stream manager è impostata su 4194304 KB (4 GB) per impostazione predefinita. Ti consigliamo di impostare la dimensione della memoria su almeno 128000 KB.

Per verificare se stream manager è abilitato (console)

1. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).
2. Scegliere il gruppo target.
3. Scegli la scheda Funzioni Lambda.
4. In Funzioni System Lambda, seleziona Stream manager e scegli Modifica.
5. Controlla lo stato abilitato o disabilitato. Vengono visualizzate anche le eventuali impostazioni di stream manager personalizzate configurate.

Per abilitare o disabilitare stream manager durante la creazione del gruppo (console)

1. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).
2. Selezionare Create Group (Crea gruppo). L'opzione scelta nella pagina successiva determina come configurare stream manager per il gruppo.
3. Procedi attraverso la sezione Assegna un nome al tuo gruppo e scegli una delle pagine principali di Greengrass.

4. Seleziona Crea gruppo.
5. Nella pagina di configurazione del gruppo, scegli la scheda Funzioni Lambda, seleziona Stream manager e scegli Modifica.
 - Per abilitare lo stream manager con le impostazioni predefinite, scegli Abilita con impostazioni predefinite.
 - Per abilitare stream manager con impostazioni personalizzate, scegliere Customize settings (Personalizza impostazioni).
 1. Nella pagina Configura Stream manager, scegli Abilita con impostazioni personalizzate.
 2. In Custom settings (Impostazioni personalizzate), immettere i valori per i parametri di stream manager. Per ulteriori informazioni, consulta [the section called “Parametri di Stream Manager”](#). Lascia i campi vuoti AWS IoT Greengrass per consentire l'utilizzo dei valori predefiniti.
 - Per disabilitare lo stream manager, scegli Disabilita.
 1. Nella pagina Configure stream manager (Configura stream manager), scegliere Disable (Disabilita).
6. Seleziona Salva.
7. Continuare attraverso le pagine rimanenti per creare il tuo gruppo.
8. Nella pagina Dispositivi client, scarica le risorse di sicurezza, esamina le informazioni, quindi scegli Fine.

 Note

Quando stream manager è abilitato, è necessario [installare il runtime Java 8](#) sul dispositivo core prima di distribuire il gruppo.

Per abilitare o disabilitare stream manager per un gruppo esistente (console)

1. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).
2. Scegliere il gruppo target.
3. Scegli la scheda Funzioni Lambda.
4. In Funzioni System Lambda, seleziona Stream manager e scegli Modifica.
5. Controlla lo stato abilitato o disabilitato. Vengono visualizzate anche le eventuali impostazioni di stream manager personalizzate configurate.

Per modificare le impostazioni di stream manager (console)

1. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).
2. Scegliere il gruppo target.
3. Scegli la scheda Funzioni Lambda.
4. In Funzioni System Lambda, seleziona Stream manager e scegli Modifica.
5. Controlla lo stato abilitato o disabilitato. Vengono visualizzate anche le eventuali impostazioni di stream manager personalizzate configurate.
6. Seleziona Salva.

Configurazione delle impostazioni di Stream Manager (CLI)

Nel AWS CLI, utilizzare la funzione `GGStreamManager Lambda` del sistema per configurare lo stream manager. Le funzioni System Lambda sono componenti del software AWS IoT Greengrass Core. Per lo stream manager e alcune altre funzioni Lambda del sistema, è possibile configurare la funzionalità Greengrass gestendo gli oggetti corrispondenti `Function` e `FunctionDefinitionVersion` nel gruppo Greengrass. Per ulteriori informazioni, consulta [the section called “Panoramica del modello di oggetti del gruppo”](#).

È possibile utilizzare l'API per le seguenti attività di gestione. Gli esempi in questa sezione mostrano come utilizzare AWS CLI, ma puoi anche chiamare l' AWS IoT Greengrass API direttamente o utilizzare un AWS SDK.

- [Verificare se Stream Manager è abilitato](#)
- [Abilitare, disabilitare o configurare lo stream manager](#)

Le modifiche diventano effettive dopo la distribuzione del gruppo. Per un tutorial che mostra come implementare un gruppo Greengrass con una funzione Lambda che interagisce con lo stream manager, vedi. [the section called “Esportazione di flussi di dati \(CLI\)”](#)

 Tip

Per verificare se lo stream manager è abilitato e se è in esecuzione dal dispositivo principale, puoi eseguire il seguente comando in un terminale del dispositivo.

```
ps aux | grep -i 'streammanager'
```

Per verificare se stream manager è abilitato (CLI)

Stream Manager è abilitato se la versione di definizione della funzione distribuita include la funzione `GGStreamManager` Lambda di sistema. Per verificare, procedere come segue:

1. Ottieni il gruppo Greengrass IDs di destinazione e la versione del gruppo. Questa procedura presuppone che questa sia la versione più recente del gruppo e del gruppo. La seguente query restituisce il gruppo creato più di recente.

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

In alternativa, puoi eseguire query in base al nome. I nomi dei gruppi non devono essere univoci, pertanto potrebbero essere restituiti più gruppi.

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

 Note

Puoi trovare questi valori anche nella AWS IoT console. L'ID gruppo viene visualizzato nella pagina Settings (Impostazioni) del gruppo. IDs Le versioni del gruppo vengono visualizzate nella scheda Distribuzioni del gruppo.

2. Copiare i valori Id e LatestVersion dal gruppo target nell'output.
3. Ottieni la versione gruppo più recente.
 - Sostituisci l'*group-id* con l'Id copiato.
 - Sostituisci l'*latest-group-version-id* con l'LatestVersion copiato.

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

4. Dall'FunctionDefinitionVersionArnoutput, ottieni la definizione della funzione e la versione IDs della definizione della funzione.
 - L'ID di definizione della funzione è il GUID che segue il functions segmento nell'Amazon Resource Name (ARN).
 - L'ID versione della definizione della funzione è il GUID che segue il segmento versions nell'ARN.

```
arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/  
functions/function-definition-id/versions/function-definition-version-id
```

5. Ottenere la versione di definizione della funzione.
 - Sostituisci *function-definition-id* con l'ID di definizione della funzione.
 - Sostituisci *function-definition-version-id* con l'ID della versione della definizione della funzione.

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

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

Se l'array `functions` nell'output include la funzione `GGStreamManager`, allora stream manager è abilitato. Le eventuali variabili di ambiente definite per la funzione rappresentano impostazioni personalizzate per stream manager.

Per abilitare, disabilitare o configurare lo stream manager (CLI)

Nel AWS CLI, utilizzare la funzione `GGStreamManager` Lambda del sistema per configurare lo stream manager. Le modifiche diventano effettive dopo la distribuzione del gruppo.

- Per abilitare stream manager, includi `GGStreamManager` nell'array `functions` della versione di definizione della funzione. Per configurare le impostazioni personalizzate, definisci le variabili di ambiente per i [parametri di Gestore di flussi](#) corrispondenti.
- Per disabilitare stream manager, rimuovi `GGStreamManager` dall'array `functions` della versione di definizione della funzione.

Gestore di flussi con impostazioni predefinite

La configurazione di esempio seguente consente di abilitare stream manager con impostazioni predefinite. Imposta l'ID della funzione arbitraria su `streamManager`.

```
{
  "FunctionArn": "arn:aws:lambda::function:GGStreamManager:1",
  "FunctionConfiguration": {
    "MemorySize": 4194304,
    "Pinned": true,
    "Timeout": 3
  },
  "Id": "streamManager"
}
```

Note

Per quanto riguarda le `FunctionConfiguration` proprietà, potresti sapere quanto segue:

- `MemorySize` è impostato su 4194304 KB (4 GB) con le impostazioni predefinite. È sempre possibile modificare questo valore. Ti consigliamo di impostarlo su almeno 128000 KB.
- `Pinned` deve essere impostato su `true`.
- `Timeout` è richiesto dalla versione di definizione della funzione, ma non viene utilizzato da `GGStreamManager`.

Gestore di flussi con impostazioni personalizzate

La configurazione di esempio seguente abilita lo stream manager con valori personalizzati per i parametri della directory di archiviazione, della porta del server e della dimensione del pool di thread.

```
{
  "FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
  "FunctionConfiguration": {
    "Environment": {
      "Variables": {
        "STREAM_MANAGER_STORE_ROOT_DIR": "/data",
        "STREAM_MANAGER_SERVER_PORT": "1234",
        "STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE": "4"
      }
    },
    "MemorySize": 4194304,
    "Pinned": true,
    "Timeout": 3
  },
  "Id": "streamManager"
}
```

AWS IoT Greengrass utilizza valori predefiniti per i [parametri dello stream manager](#) che non sono specificati come variabili di ambiente.

Stream manager con impostazioni personalizzate per le esportazioni di Amazon S3

La seguente configurazione di esempio abilita lo stream manager con valori personalizzati per la directory di caricamento e parametri minimi di dimensione di caricamento multiparte.

```
{
  "FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
```

```
"FunctionConfiguration": {
  "Environment": {
    "Variables": {
      "STREAM_MANAGER_READ_ONLY_DIRS": "/mnt/directory-1,/mnt/
directory-2,/tmp",

"STREAM_MANAGER_EXPORTER_S3_DESTINATION_MULTIPART_UPLOAD_MIN_PART_SIZE_BYTES":
"10485760"
    }
  },
  "MemorySize": 4194304,
  "Pinned": true,
  "Timeout": 3
},
"Id": "streamManager"
}
```

Per abilitare, disabilitare o configurare lo stream manager (CLI)

1. Ottieni il gruppo Greengrass IDs di destinazione e la versione del gruppo. Questa procedura presuppone che questa sia la versione più recente del gruppo e del gruppo. La seguente query restituisce il gruppo creato più di recente.

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

In alternativa, puoi eseguire query in base al nome. I nomi dei gruppi non devono essere univoci, pertanto potrebbero essere restituiti più gruppi.

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

Note

Puoi trovare questi valori anche nella AWS IoT console. L'ID gruppo viene visualizzato nella pagina Settings (Impostazioni) del gruppo. IDs Le versioni del gruppo vengono visualizzate nella scheda Distribuzioni del gruppo.

2. Copiare i valori Id e LatestVersion dal gruppo target nell'output.
3. Ottieni la versione gruppo più recente.
 - Sostituisci l'*group-id* con l'Id copiato.
 - Sostituisci l'*latest-group-version-id* con l'LatestVersion copiato.

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

4. Copia la versione CoreDefinitionVersionArn e tutte le altre ARNs dall'output, tranne. FunctionDefinitionVersionArn Questi valori verranno utilizzati successivamente quando si crea una versione di gruppo.
5. Da FunctionDefinitionVersionArn nell'output, copia l'ID della definizione della funzione. L'ID è il GUID che segue il segmento functions nell'ARN, come mostrato nell'esempio seguente.

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

Note

In alternativa, è possibile creare una definizione di funzione eseguendo il [create-function-definition](#) comando e quindi copiando l'ID dall'output.

6. Aggiungi una versione di definizione della funzione alla definizione della funzione.
 - *function-definition-id* Sostituiscilo con Id quello che hai copiato per la definizione della funzione.
 - Nell'functionsarray, includi tutte le altre funzioni che desideri rendere disponibili sul core Greengrass. Puoi utilizzare il comando get-function-definition-version per ottenere l'elenco delle funzioni esistenti.

Abilitazione di stream manager con impostazioni predefinite

L'esempio seguente abilita lo stream manager, includendo la `GGStreamManager` funzione nell'`functionsarray`. In questo esempio vengono utilizzati i valori predefiniti per i [parametri di Gestore di flussi](#).

```
aws greengrass create-function-definition-version \  
--function-definition-id function-definition-id \  
--functions '[  
  {  
    "FunctionArn": "arn:aws:lambda::function:GGStreamManager:1",  
    "FunctionConfiguration": {  
      "MemorySize": 4194304,  
      "Pinned": true,  
      "Timeout": 3  
    },  
    "Id": "streamManager"  
  },  
  {  
    "FunctionArn": "arn:aws:lambda:us-  
west-2:123456789012:function:MyLambdaFunction:MyAlias",  
    "FunctionConfiguration": {  
      "Executable": "myLambdaFunction.function_handler",  
      "MemorySize": 16000,  
      "Pinned": true,  
      "Timeout": 5  
    },  
    "Id": "myLambdaFunction"  
  },  
  ... more user-defined functions  
]  
'
```

Note

La `myLambdaFunction` funzione negli esempi rappresenta una delle funzioni Lambda definite dall'utente.

Abilitazione di stream manager con impostazioni personalizzate

L'esempio seguente abilita Gestore di flussi includendo la funzione `GGStreamManager` nell'array `functions`. Tutte le impostazioni dello stream manager sono opzionali, a meno che non si desideri modificare i valori predefiniti. Questo esempio mostra come utilizzare le variabili di ambiente per impostare valori personalizzati.

```
aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
--functions '[
    {
        "FunctionArn": "arn:aws:lambda::function:GGStreamManager:1",
        "FunctionConfiguration": {
            "Environment": {
                "Variables": {
                    "STREAM_MANAGER_STORE_ROOT_DIR": "/data",
                    "STREAM_MANAGER_SERVER_PORT": "1234",
                    "STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE": "4"
                }
            },
            "MemorySize": 4194304,
            "Pinned": true,
            "Timeout": 3
        },
        "Id": "streamManager"
    },
    {
        "FunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:MyLambdaFunction:MyAlias",
        "FunctionConfiguration": {
            "Executable": "myLambdaFunction.function_handler",
            "MemorySize": 16000,
            "Pinned": true,
            "Timeout": 5
        },
        "Id": "myLambdaFunction"
    },
    ... more user-defined functions
]
```


 Note

Per quanto riguarda le `FunctionConfiguration` proprietà, potresti sapere quanto segue:

- `MemorySize` è impostato su 4194304 KB (4 GB) con le impostazioni predefinite. È sempre possibile modificare questo valore. Ti consigliamo di `MemorySize` impostarlo su almeno 128000 KB.
- `Pinned` deve essere impostato su `true`.
- `Timeout` è richiesto dalla versione di definizione della funzione, ma non viene utilizzato da `GGStreamManager`.

Disabilitazione di Gestore di flussi

L'esempio seguente omette la funzione `GGStreamManager`, che disabilita Gestore di flussi.

```
aws greengrass create-function-definition-version \  
--function-definition-id function-definition-id \  
--functions '[  
    {  
        "FunctionArn": "arn:aws:lambda:us-  
west-2:123456789012:function:MyLambdaFunction:MyAlias",  
        "FunctionConfiguration": {  
            "Executable": "myLambdaFunction.function_handler",  
            "MemorySize": 16000,  
            "Pinned": true,  
            "Timeout": 5  
        },  
        "Id": "myLambdaFunction"  
    },  
    ... more user-defined functions  
]  
'
```

 Note

Se non desideri distribuire alcuna funzione Lambda, puoi omettere completamente la versione della definizione della funzione.

7. Copia l'Arn della definizione di funzione dall'output.
8. Crea una versione di gruppo che contenga la funzione Lambda del sistema.
 - Sostituisci *group-id* con Id for the group.
 - Sostituiscilo *core-definition-version-arn* con CoreDefinitionVersionArn quello che hai copiato dalla versione più recente del gruppo.
 - Sostituiscilo *function-definition-version-arn* con Arn quello che hai copiato per la nuova versione della definizione di funzione.
 - Sostituisci ARNs i componenti degli altri gruppi (ad esempio, SubscriptionDefinitionVersionArn o DeviceDefinitionVersionArn) che hai copiato dalla versione più recente del gruppo.
 - Rimuovi eventuali parametri inutilizzati. Ad esempio, rimuovi `--resource-definition-version-arn` se la versione gruppo non contiene risorse.

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

9. Copia il valore Version dall'output. Questo è l'ID della nuova versione gruppo.
10. Distribuisci il gruppo con la nuova versione del gruppo.
 - Sostituiscilo *group-id* con Id quello che hai copiato per il gruppo.
 - Sostituisci *group-version-id* con Version quello che hai copiato per la nuova versione del gruppo.

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

Segui questa procedura se desideri modificare nuovamente le impostazioni dello stream manager in un secondo momento. Assicurati di creare una versione della definizione della funzione che includa la `GGStreamManager` funzione con la configurazione aggiornata. La versione di gruppo deve fare riferimento a tutte le versioni dei componenti ARNs che si desidera distribuire nel core. Le modifiche diventano effettive dopo la distribuzione del gruppo.

Consulta anche

- [Gestione dei flussi di dati](#)
- [the section called “Utilizzalo StreamManagerClient per lavorare con gli stream”](#)
- [the section called “Esporta le configurazioni per le destinazioni supportate Cloud AWS”](#)
- [the section called “Esportazione di flussi di dati \(Console\)”](#)
- [the section called “Esportazione di flussi di dati \(CLI\)”](#)

Utilizzalo StreamManagerClient per lavorare con gli stream

Le funzioni Lambda definite dall'utente in esecuzione AWS IoT Greengrass sul core possono utilizzare `StreamManagerClient` l'oggetto nel [AWS IoT Greengrass Core](#) SDK per creare [stream in stream manager e quindi interagire con gli stream](#). Quando una funzione Lambda crea un flusso, definisce le Cloud AWS destinazioni, l'assegnazione delle priorità e altre politiche di esportazione e conservazione dei dati per lo stream. Per inviare dati allo stream manager, le funzioni Lambda aggiungono i dati allo stream. Se viene definita una destinazione di esportazione per lo stream, stream manager esporta il flusso automaticamente.

Note

In genere, i client di stream manager sono funzioni Lambda definite dall'utente. Se il tuo business case lo richiede, puoi anche consentire ai processi non Lambda in esecuzione sul core Greengrass (ad esempio, un contenitore Docker) di interagire con lo stream manager. Per ulteriori informazioni, consulta [the section called “Autenticazione client”](#).

Gli snippet di questo argomento mostrano come i client `StreamManagerClient` chiamano i metodi per lavorare con gli stream. Per i dettagli di implementazione sui metodi e i relativi argomenti, utilizzate i collegamenti al riferimento SDK elencati dopo ogni frammento. Per i tutorial che includono

una funzione Lambda di Python completa, consulta o. [the section called “Esportazione di flussi di dati \(Console\)”](#) [the section called “Esportazione di flussi di dati \(CLI\)”](#)

La tua funzione Lambda dovrebbe creare un'istanza `StreamManagerClient` all'esterno del gestore delle funzioni. Se viene creata un'istanza nel gestore, la funzione crea un `client` e una connessione al gestore flussi ogni volta che viene richiamata.

Note

Se si esegue un'istanza `StreamManagerClient` nel gestore, è necessario chiamare esplicitamente il metodo `close()` quando `client` completa il suo lavoro. In caso contrario, `client` mantiene la connessione aperta e un altro thread in esecuzione fino alla chiusura dello script.

`StreamManagerClient` supporta le seguenti operazioni:

- [the section called “Creazione del flusso di messaggi”](#)
- [the section called “Aggiunta di un messaggio”](#)
- [the section called “Lettura di messaggi”](#)
- [the section called “Visualizzazione dell'elenco di flussi”](#)
- [the section called “Descrizione del flusso di messaggi”](#)
- [the section called “Aggiorna il flusso di messaggi”](#)
- [the section called “Eliminazione del flusso di messaggi”](#)

Creazione del flusso di messaggi

Per creare uno stream, una funzione Lambda definita dall'utente chiama il metodo `create` e passa un oggetto `MessageStreamDefinition`. Questo oggetto specifica il nome univoco dello stream e definisce come lo stream manager deve gestire i nuovi dati quando viene raggiunta la dimensione massima del flusso. È possibile utilizzare `MessageStreamDefinition` e relativi tipi di dati (ad esempio `ExportDefinition`, `StrategyOnFull` e `Persistence`) per definire altre proprietà del flusso. Ciò include:

- L'obiettivo AWS IoT Analytics, Kinesis Data AWS IoT SiteWise Streams e le destinazioni Amazon S3 per le esportazioni automatiche. Per ulteriori informazioni, consulta [the section called “Esporta le configurazioni per le destinazioni supportate Cloud AWS”](#).

- **Priorità di esportazione.** Stream manager esporta i flussi con priorità più alta prima dei flussi con priorità più bassa.
- **Dimensione e intervallo massimi del batch per AWS IoT Analytics Kinesis Data Streams e destinazioni.** AWS IoT SiteWise Stream manager esporta i messaggi quando viene soddisfatta una delle due condizioni.
- **Time-to-live (TTL).** Il tempo necessario per garantire che i dati del flusso siano disponibili per l'elaborazione. È necessario assicurarsi che i dati possano essere utilizzati entro questo periodo di tempo. Questa non è una policy di eliminazione. È possibile che i dati non vengano eliminati immediatamente dopo il periodo TTL.
- **Persistenza del flusso.** Scegliere di salvare i flussi nel file system per mantenere i dati tra riavvii core o salvare i flussi in memoria.
- **Numero di sequenza iniziale.** Specificate il numero di sequenza del messaggio da utilizzare come messaggio iniziale nell'esportazione.

Per ulteriori informazioni su `MessageStreamDefinition`, consultate il riferimento SDK per la lingua di destinazione:

- [MessageStreamDefinition](#) nell'SDK Java
- [MessageStreamDefinition](#) nell'SDK Node.js
- [MessageStreamDefinition](#) nell'SDK Python

Note

`StreamManagerClient` fornisce anche una destinazione di destinazione che è possibile utilizzare per esportare flussi su un server HTTP. Questo target è destinato esclusivamente a scopi di test. Non è stabile né è supportato per l'uso in ambienti di produzione.

Dopo aver creato uno stream, le funzioni Lambda possono [aggiungere messaggi](#) allo stream per inviare dati per l'esportazione e [leggere i messaggi](#) dallo stream per l'elaborazione locale. Il numero di flussi creati dipende dalle funzionalità hardware e dal business case. Una strategia consiste nel creare un flusso per ogni canale di destinazione nel AWS IoT Analytics nostro flusso di dati Kinesis, sebbene sia possibile definire più destinazioni per un flusso. Un flusso ha una lunga durata.

Requisiti

Questa operazione ha i seguenti requisiti:

- Versione minima AWS IoT Greengrass Core: 1.10.0
- Versione minima AWS IoT Greengrass Core SDK: Python: 1.5.0 | Java: 1.4.0 | Node.js: 1.6.0

Note

La creazione di stream con una destinazione AWS IoT SiteWise di esportazione Amazon S3 presenta i seguenti requisiti:

- Versione minima AWS IoT Greengrass Core: 1.11.0
- Versione minima AWS IoT Greengrass Core SDK: Python: 1.6.0 | Java: 1.5.0 | Node.js: 1.7.0

Esempi

Il frammento di codice seguente crea un flusso denominato `StreamName`. Definisce le proprietà del flusso nei tipi di dati subordinati. `MessageStreamDefinition`

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 Cloud AWS.
            kinesis=None,
            iot_analytics=None,
            iot_sitewise=None,
```

```

        s3_task_executor=None
    )
))
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.

```

[Riferimento all'SDK Python: create_message_stream | MessageStreamDefinition](#)

Java

```

try (final StreamManagerClient client =
    GreengrassClientBuilder.streamManagerClient().build()) {
    client.createMessageStream(
        new MessageStreamDefinition()
            .withName("StreamName") // Required.
            .withMaxSize(268435456L) // Default is 256 MB.
            .withStreamSegmentSize(16777216L) // Default is 16 MB.
            .withTimeToLiveMillis(null) // By default, no TTL is enabled.
            .withStrategyOnFull(StrategyOnFull.OverwriteOldestData) //
Required.

            .withPersistence(Persistence.File) // Default is File.
            .withFlushOnWrite(false) // Default is false.
            .withExportDefinition( // Optional. Choose where/how the stream
is exported to the Cloud AWS.
                new ExportDefinition()
                    .withKinesis(null)
                    .withIotAnalytics(null)
                    .withIotSitewise(null)
                    .withS3TaskExecutor(null)
                )
            );
} catch (StreamManagerException e) {
    // Properly handle exception.
}

```

Riferimento a Java [createMessageStream](#) SDK: | [MessageStreamDefinition](#)

Node.js

```

const client = new StreamManagerClient();
client.onConnected(async () => {
  try {
    await client.createMessageStream(
      new MessageStreamDefinition()
        .WithName("StreamName") // Required.
        .withMaxSize(268435456) // Default is 256 MB.
        .withStreamSegmentSize(16777216) // Default is 16 MB.
        .withTimeToLiveMillis(null) // By default, no TTL is enabled.
        .withStrategyOnFull(StrategyOnFull.OverwriteOldestData) //
Required.
        .withPersistence(Persistence.File) // Default is File.
        .withFlushOnWrite(false) // Default is false.
        .withExportDefinition( // Optional. Choose where/how the stream is
exported to the Cloud AWS.
          new ExportDefinition()
            .withKinesis(null)
            .withIotAnalytics(null)
            .withIotSitewise(null)
            .withS3TaskExecutor(null)
          )
        );
  } catch (e) {
    // Properly handle errors.
  }
});
client.onError((err) => {
  // Properly handle connection errors.
  // This is called only when the connection to the StreamManager server fails.
});

```

Riferimento all'SDK Node.js: | [createMessageStreamMessageStreamDefinition](#)

Per ulteriori informazioni sulla configurazione delle destinazioni di esportazione, consulta. [the section called “Esporta le configurazioni per le destinazioni supportate Cloud AWS”](#)

Aggiunta di un messaggio

Per inviare dati allo stream manager per l'esportazione, le funzioni Lambda aggiungono i dati allo stream di destinazione. La destinazione di esportazione determina il tipo di dati da passare a questo metodo.

Requisiti

Questa operazione presenta i seguenti requisiti:

- Versione minima AWS IoT Greengrass Core: 1.10.0
- Versione minima AWS IoT Greengrass Core SDK: Python: 1.5.0 | Java: 1.4.0 | Node.js: 1.6.0

Note

L'aggiunta di messaggi con una destinazione AWS IoT SiteWise di esportazione Amazon S3 presenta i seguenti requisiti:

- Versione minima AWS IoT Greengrass Core: 1.11.0
- Versione minima AWS IoT Greengrass Core SDK: Python: 1.6.0 | Java: 1.5.0 | Node.js: 1.7.0

Esempi

AWS IoT Analytics o destinazioni di esportazione Kinesis Data Streams

Il frammento di codice seguente aggiunge un messaggio al flusso denominato `StreamName`. Per le AWS IoT Analytics nostre destinazioni Kinesis Data Streams, le funzioni Lambda aggiungono un blob di dati.

Questo frammento ha i seguenti requisiti:

- Versione minima AWS IoT Greengrass Core: 1.10.0
- Versione minima AWS IoT Greengrass Core SDK: Python: 1.5.0 | Java: 1.4.0 | Node.js: 1.6.0

Python

```
client = StreamManagerClient()
```

```
try:
    sequence_number = client.append_message(stream_name="StreamName",
    data=b'Arbitrary bytes data')
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

[Riferimento Python SDK: append_message](#)

Java

```
try (final StreamManagerClient client =
    GreengrassClientBuilder.streamManagerClient().build()) {
    long sequenceNumber = client.appendMessage("StreamName", "Arbitrary byte
    array".getBytes());
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

[Riferimento Java SDK: 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.
});
```

[Riferimento SDK Node.js: appendMessage](#)

AWS IoT SiteWise destinazioni di esportazione

Il frammento di codice seguente aggiunge un messaggio al flusso denominato `StreamName`. Per le AWS IoT SiteWise destinazioni, le funzioni Lambda aggiungono un oggetto serializzato. `PutAssetPropertyValueEntry` Per ulteriori informazioni, consulta [the section called “Esportazione in AWS IoT SiteWise”](#).

Note

Quando si inviano dati a AWS IoT SiteWise, i dati devono soddisfare i requisiti dell'azione. `BatchPutAssetPropertyValue` Per ulteriori informazioni, consulta [BatchPutAssetPropertyValue](#) nella documentazione di riferimento dell'API AWS IoT SiteWise.

Questo frammento presenta i seguenti requisiti:

- Versione minima AWS IoT Greengrass Core: 1.11.0
- Versione minima AWS IoT Greengrass Core SDK: Python: 1.6.0 | Java: 1.5.0 | Node.js: 1.7.0

Python

```
client = StreamManagerClient()

try:
    # SiteWise requires unique timestamps in all messages. Add some randomness to
    # time and offset.

    # Note: To create a new asset property data, you should use the classes defined
    # in the
    # greengrasssdk.stream_manager module.

    time_in_nanos = TimeInNanos(
        time_in_seconds=calendar.timegm(time.gmtime()) - random.randint(0, 60),
        offset_in_nanos=random.randint(0, 10000)
    )
    variant = Variant(double_value=random.random())
    asset = [AssetPropertyValue(value=variant, quality=Quality.GOOD,
        timestamp=time_in_nanos)]
```

```

    putAssetPropertyValueEntry =
    PutAssetPropertyValueEntry(entry_id=str(uuid.uuid4()),
    property_alias="PropertyAlias", property_values=asset)
    sequence_number = client.append_message(stream_name="StreamName",
    data=Util.validate_and_serialize_to_json_bytes(putAssetPropertyValueEntry))
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.

```

[Riferimento all'SDK Python: `append\_message` | `PutAssetPropertyValueEntry`](#)

Java

```

try (final StreamManagerClient client =
    GreengrassClientBuilder.streamManagerClient().build()) {
    Random rand = new Random();
    // Note: To create a new asset property data, you should use the classes defined
    in the
    // com.amazonaws.greengrass.streammanager.model.sitewise package.
    List<AssetPropertyValue> entries = new ArrayList<>() ;

    // IoTSiteWise requires unique timestamps in all messages. Add some randomness
    to time and offset.
    final int maxTimeRandomness = 60;
    final int maxOffsetRandomness = 10000;
    double randomValue = rand.nextDouble();
    TimeInNanos timestamp = new TimeInNanos()
        .withTimeInSeconds(Instant.now().getEpochSecond() -
    rand.nextInt(maxTimeRandomness))
        .withOffsetInNanos((long) (rand.nextInt(maxOffsetRandomness)));
    AssetPropertyValue entry = new AssetPropertyValue()
        .withValue(new Variant().withDoubleValue(randomValue))
        .withQuality(Quality.GOOD)
        .withTimestamp(timestamp);
    entries.add(entry);

    PutAssetPropertyValueEntry putAssetPropertyValueEntry = new
    PutAssetPropertyValueEntry()
        .withEntryId(UUID.randomUUID().toString())
        .withPropertyAlias("PropertyAlias")
        .withPropertyValues(entries);

```

```

    long sequenceNumber = client.appendMessage("StreamName",
        ValidateAndSerialize.validateAndSerializeToJsonBytes(putAssetPropertyValueEntry));
} catch (StreamManagerException e) {
    // Properly handle exception.
}

```

Riferimento Java SDK: [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.
});

```

Riferimento all'SDK Node.js: [appendMessage | PutAssetPropertyValueEntry](#)

Destinazioni di esportazione Amazon S3

Il seguente frammento aggiunge un'attività di esportazione allo stream denominato. `StreamName`. Per le destinazioni Amazon S3, le funzioni Lambda aggiungono un oggetto serializzato che contiene informazioni sul file di input di origine e sull'`S3ExportTaskDefinition` oggetto Amazon S3 di destinazione. Se l'oggetto specificato non esiste, Stream Manager lo crea per te. Per ulteriori informazioni, consulta [the section called “Esportazione su Amazon S3”](#).

Questo frammento ha i seguenti requisiti:

- Versione minima AWS IoT Greengrass Core: 1.11.0
- Versione minima AWS IoT Greengrass Core SDK: Python: 1.6.0 | Java: 1.5.0 | Node.js: 1.7.0

Python

```
client = StreamManagerClient()

try:
    # Append an Amazon S3 Task definition and print the sequence number.
    s3_export_task_definition = S3ExportTaskDefinition(input_url="URLToFile",
    bucket="BucketName", key="KeyName")
    sequence_number = client.append_message(stream_name="StreamName",
    data=Util.validate_and_serialize_to_json_bytes(s3_export_task_definition))
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

[Riferimento all'SDK Python: `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.
}
```

[Riferimento a Java SDK: 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.
});
```

[Riferimento all'SDK Node.js: appendMessage | S3 ExportTaskDefinition](#)

Lettura di messaggi

Leggi i messaggi da uno stream.

Requisiti

Questa operazione ha i seguenti requisiti:

- Versione minima AWS IoT Greengrass Core: 1.10.0

- Versione minima AWS IoT Greengrass Core SDK: Python: 1.5.0 | Java: 1.4.0 | Node.js: 1.6.0

Esempi

Il frammento di codice seguente legge i messaggi dal flusso denominato `StreamName`. Il metodo di lettura accetta un oggetto `ReadMessagesOptions` facoltativo che specifica il numero di sequenza da cui iniziare la lettura, i numeri minimo e massimo da leggere e un timeout per la lettura dei messaggi.

Python

```
client = StreamManagerClient()

try:
    message_list = client.read_messages(
        stream_name="StreamName",
        # By default, if no options are specified, it tries to read one message from
        # the beginning of the stream.
        options=ReadMessagesOptions(
            desired_start_sequence_number=100,
            # Try to read from sequence number 100 or greater. By default, this is
            0.
            min_message_count=10,
            # Try to read 10 messages. If 10 messages are not available, then
            # NotEnoughMessagesException is raised. By default, this is 1.
            max_message_count=100, # Accept up to 100 messages. By default this is
            1.
            read_timeout_millis=5000
            # Try to wait at most 5 seconds for the min_message_count to be
            # fulfilled. By default, this is 0, which immediately returns the messages or an
            # exception.
        )
    )
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

[Riferimento all'SDK Python: read_messages | ReadMessagesOptions](#)

Java

```
try (final StreamManagerClient client =
    GreengrassClientBuilder.streamManagerClient().build()) {
    List<Message> messages = client.readMessages("StreamName",
        // By default, if no options are specified, it tries to read one message
        from the beginning of the stream.
        new ReadMessagesOptions()
            // Try to read from sequence number 100 or greater. By default
            this is 0.
            .withDesiredStartSequenceNumber(100L)
            // Try to read 10 messages. If 10 messages are not available,
            then NotEnoughMessagesException is raised. By default, this is 1.
            .withMinMessageCount(10L)
            // Accept up to 100 messages. By default this is 1.
            .withMaxMessageCount(100L)
            // Try to wait at most 5 seconds for the min_message_count to
            be fulfilled. By default, this is 0, which immediately returns the messages or an
            exception.
            .withReadTimeoutMillis(Duration.ofSeconds(5L).toMillis()));
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Riferimento a Java SDK: [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.
});
```

Riferimento all'SDK Node.js: [ReadMessages](#) | [ReadMessagesOptions](#)

Visualizzazione dell'elenco di flussi

Ottieni l'elenco degli stream nello stream manager.

Requisiti

Questa operazione ha i seguenti requisiti:

- Versione minima AWS IoT Greengrass Core: 1.10.0
- Versione minima AWS IoT Greengrass Core SDK: Python: 1.5.0 | Java: 1.4.0 | Node.js: 1.6.0

Esempi

Il frammento di codice seguente ottiene un elenco dei flussi (per nome) in stream manager.

Python

```
client = StreamManagerClient()

try:
    stream_names = client.list_streams()
except StreamManagerException:
```

```
pass
# Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
# Properly handle errors.
```

[Riferimento all'SDK Python: list_streams](#)

Java

```
try (final StreamManagerClient client =
    GreengrassClientBuilder.streamManagerClient().build()) {
    List<String> streamNames = client.listStreams();
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

[Riferimento Java SDK: 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.
});
```

[Riferimento all'SDK Node.js: ListStreams](#)

Descrizione del flusso di messaggi

Ottieni i metadati relativi a uno stream, tra cui la definizione, la dimensione e lo stato dell'esportazione.

Requisiti

Questa operazione ha i seguenti requisiti:

- Versione minima AWS IoT Greengrass Core: 1.10.0
- Versione minima AWS IoT Greengrass Core SDK: Python: 1.5.0 | Java: 1.4.0 | Node.js: 1.6.0

Esempi

Il frammento di codice seguente ottiene i metadati relativi al flusso denominato `StreamName`, inclusi la definizione, le dimensioni e gli stati di esportatore del flusso.

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.
```

[Riferimento all'SDK Python: describe_message_stream](#)

Java

```
try (final StreamManagerClient client =
    GreengrassClientBuilder.streamManagerClient().build()) {
    MessageStreamInfo description = client.describeMessageStream("StreamName");
```

```

String lastErrorMessage =
description.getExportStatuses().get(0).getErrorMessage();
if (lastErrorMessage != null && !lastErrorMessage.equals("")) {
    // The last export of export destination 0 failed with some error.
    // Here is the last sequence number that was successfully exported.
    description.getExportStatuses().get(0).getLastExportedSequenceNumber();
}

if (description.getStorageStatus().getNewestSequenceNumber() >
    description.getExportStatuses().get(0).getLastExportedSequenceNumber())
{
    // The end of the stream is ahead of the last exported sequence number.
}
} catch (StreamManagerException e) {
    // Properly handle exception.
}

```

Riferimento a Java SDK: [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.

```

```
});
```

Riferimento all'SDK Node.js: [describeMessageStream](#)

Aggiorna il flusso di messaggi

Aggiorna le proprietà di uno stream esistente. Potresti voler aggiornare uno stream se i tuoi requisiti cambiano dopo la creazione dello stream. Per esempio:

- Aggiungi una nuova [configurazione di esportazione](#) per una Cloud AWS destinazione.
- Aumenta la dimensione massima di uno stream per modificare il modo in cui i dati vengono esportati o conservati. Ad esempio, la dimensione dello stream in combinazione con la tua strategia sulle impostazioni complete potrebbe comportare l'eliminazione o il rifiuto dei dati prima che lo stream manager possa elaborarli.
- Metti in pausa e riprendi le esportazioni, ad esempio se le attività di esportazione richiedono molto tempo e desideri razionare i dati di caricamento.

Le tue funzioni Lambda seguono questo processo di alto livello per aggiornare uno stream:

1. [Ottieni la descrizione dello stream.](#)
2. Aggiorna le proprietà di destinazione sugli oggetti corrispondenti `MessageStreamDefinition` e subordinati.
3. Passa l'aggiornamento `MessageStreamDefinition`. Assicurati di includere le definizioni complete degli oggetti per lo stream aggiornato. Le proprietà non definite tornano ai valori predefiniti.

È possibile specificare il numero di sequenza del messaggio da utilizzare come messaggio iniziale nell'esportazione.

Requisiti

Questa operazione presenta i seguenti requisiti:

- Versione minima AWS IoT Greengrass Core: 1.11.0
- Versione minima AWS IoT Greengrass Core SDK: Python: 1.6.0 | Java: 1.5.0 | Node.js: 1.7.0

Esempi

Il seguente frammento aggiorna lo stream denominato. StreamName Aggiorna più proprietà di un flusso che esporta in Kinesis Data Streams.

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.
```

Riferimento a Python SDK: | [updateMessageStreamMessageStreamDefinition](#)

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 Cloud
AWS.
            messageStreamInfo.getDefinition().getExportDefinition().
            // Updating Export definition to add a Kinesis Stream
configuration.

            .withKinesis(new ArrayList<KinesisConfig>() {{
                add(new KinesisConfig()
                    .withIdentifier(EXPORT_IDENTIFIER)
                    .withKinesisStreamName("test"));
            }})
        );
    } catch (StreamManagerException e) {
        // Properly handle exception.
    }

```

[Riferimento a Java SDK: 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
Cloud AWS.
            messageStreamInfo.definition.exportDefinition
            // Updating Export definition to add a Kinesis Stream
configuration.
            .withKinesis([new
KinesisConfig().withIdentifier(uuidv4()).withKinesisStreamName(uuidv4())])
        )
    );
} catch (e) {
    // Properly handle errors.
}
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Riferimento all'[updateMessageStream](#) SDK Node.js: | [MessageStreamDefinition](#)

Vincoli per l'aggiornamento degli stream

I seguenti vincoli si applicano all'aggiornamento dei flussi. A meno che non sia indicato nell'elenco seguente, gli aggiornamenti hanno effetto immediato.

- Non puoi aggiornare la persistenza di uno stream. Per modificare questo comportamento, [elimina lo stream](#) e [crea uno stream](#) che definisca la nuova politica di persistenza.
- Puoi aggiornare la dimensione massima di uno stream solo nelle seguenti condizioni:
 - La dimensione massima deve essere maggiore o uguale alla dimensione corrente dello stream. Per trovare queste informazioni, [descrivi lo stream](#) e poi controlla lo stato di archiviazione dell'`MessageStreamInfo` oggetto restituito.
 - La dimensione massima deve essere maggiore o uguale alla dimensione del segmento dello stream.
- Puoi aggiornare la dimensione del segmento di stream a un valore inferiore alla dimensione massima dello stream. L'impostazione aggiornata si applica ai nuovi segmenti.

- Gli aggiornamenti alla proprietà `time to live (TTL)` si applicano alle nuove operazioni di aggiunta. Se riduci questo valore, stream manager potrebbe anche eliminare i segmenti esistenti che superano il TTL.
- Gli aggiornamenti alla strategia sulla proprietà completa si applicano alle nuove operazioni di aggiunta. Se imposti la strategia per sovrascrivere i dati più vecchi, stream manager potrebbe anche sovrascrivere i segmenti esistenti in base alla nuova impostazione.
- Gli aggiornamenti alla proprietà `flush on write` si applicano ai nuovi messaggi.
- Gli aggiornamenti alle configurazioni di esportazione si applicano alle nuove esportazioni. La richiesta di aggiornamento deve includere tutte le configurazioni di esportazione che si desidera supportare. Altrimenti, stream manager le elimina.
 - Quando aggiorni una configurazione di esportazione, specifica l'identificatore della configurazione di esportazione di destinazione.
 - Per aggiungere una configurazione di esportazione, specificate un identificatore univoco per la nuova configurazione di esportazione.
 - Per eliminare una configurazione di esportazione, omettete la configurazione di esportazione.
- Per [aggiornare](#) il numero di sequenza iniziale di una configurazione di esportazione in uno stream, è necessario specificare un valore inferiore al numero di sequenza più recente. Per trovare queste informazioni, [descriveri lo stream](#) e quindi controlla lo stato di archiviazione dell'`MessageStreamInfo` oggetto restituito.

Eliminazione del flusso di messaggi

Elimina un flusso. Quando si elimina un flusso, tutti i dati memorizzati per il flusso vengono eliminati dal disco.

Requisiti

Questa operazione ha i seguenti requisiti:

- Versione minima AWS IoT Greengrass Core: 1.10.0
- Versione minima AWS IoT Greengrass Core SDK: Python: 1.5.0 | Java: 1.4.0 | Node.js: 1.6.0

Esempi

Il frammento di codice seguente elimina il flusso denominato `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.
```

Riferimento all'SDK Python: [deleteMessageStream](#)

Java

```
try (final StreamManagerClient client =
    GreengrassClientBuilder.streamManagerClient().build()) {
    client.deleteMessageStream("StreamName");
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Riferimento Java SDK: [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.
```

```
});
```

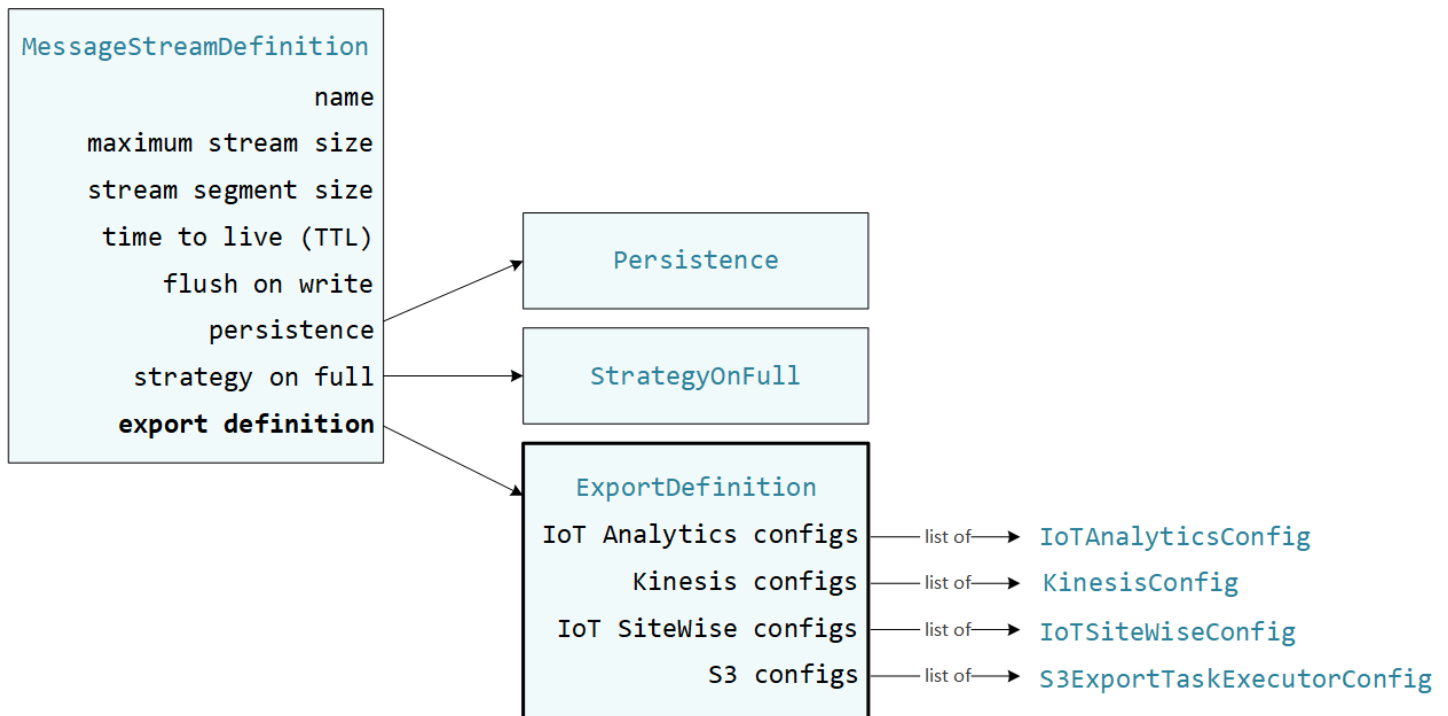
Riferimento all'SDK Node.js: [deleteMessageStream](#)

Consulta anche

- [Gestione dei flussi di dati](#)
- [the section called “Configurazione di Stream Manager”](#)
- [the section called “Esporta le configurazioni per le destinazioni supportate Cloud AWS”](#)
- [the section called “Esportazione di flussi di dati \(Console\)”](#)
- [the section called “Esportazione di flussi di dati \(CLI\)”](#)
- StreamManagerClient nel riferimento AWS IoT Greengrass Core SDK:
 - [Python](#)
 - [Java](#)
 - [Node.js](#)

Esporta le configurazioni per le destinazioni supportate Cloud AWS

Le funzioni Lambda definite dall'utente vengono StreamManagerClient utilizzate AWS IoT Greengrass nell'SDK Core per interagire con lo stream manager. Quando una funzione Lambda [crea uno stream](#) o [aggiorna uno stream, passa un](#) MessageStreamDefinition oggetto che rappresenta le proprietà dello stream, inclusa la definizione di esportazione. L'ExportDefinition oggetto contiene le configurazioni di esportazione definite per lo stream. Stream Manager utilizza queste configurazioni di esportazione per determinare dove e come esportare lo stream.



È possibile definire zero o più configurazioni di esportazione su uno stream, incluse più configurazioni di esportazione per un singolo tipo di destinazione. Ad esempio, puoi esportare uno stream su due AWS IoT Analytics canali e un flusso di dati Kinesis.

In caso di tentativi di esportazione falliti, stream manager riprova continuamente a esportare i dati verso il a Cloud AWS intervalli fino a cinque minuti. Il numero di tentativi di nuovo tentativo non ha un limite massimo.

Note

`StreamManagerClient` fornisce anche una destinazione di destinazione che è possibile utilizzare per esportare i flussi su un server HTTP. Questo target è destinato esclusivamente a scopi di test. Non è stabile né è supportato per l'uso in ambienti di produzione.

Cloud AWS Destinazioni supportate

- [AWS IoT Analytics canali](#)
- [Flussi di dati Amazon Kinesis](#)
- [AWS IoT SiteWise proprietà degli asset](#)
- [Oggetti Amazon S3](#)

Sei responsabile del mantenimento di queste Cloud AWS risorse.

AWS IoT Analytics canali

Stream manager supporta le esportazioni automatiche verso AWS IoT Analytics. AWS IoT Analytics consente di eseguire analisi avanzate sui dati per aiutare a prendere decisioni aziendali e migliorare i modelli di apprendimento automatico. Per ulteriori informazioni, consulta [Cos'è AWS IoT Analytics?](#) nella Guida AWS IoT Analytics per l'utente.

Nel AWS IoT Greengrass Core SDK, le funzioni Lambda utilizzano per definire `IoTAnalyticsConfig` la configurazione di esportazione per questo tipo di destinazione. Per ulteriori informazioni, consulta il riferimento SDK per la lingua di destinazione:

- [Io TAnalytics Config nell'SDK](#) Python
- [Io TAnalytics Config nell'SDK](#) Java
- [Io TAnalytics Config nell'SDK](#) Node.js

Requisiti

Questa destinazione di esportazione presenta i seguenti requisiti:

- I canali di destinazione in ingresso AWS IoT Analytics devono appartenere allo Regione AWS stesso Account AWS gruppo Greengrass.
- [the section called “Ruolo del gruppo Greengrass”](#) Devono consentire `l'iotanalytics:BatchPutMessage` autorizzazione per i canali di destinazione. Per esempio:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iotanalytics:BatchPutMessage"
      ],
      "Resource": [
        "arn:aws:iotanalytics:region:account-id:channel/channel_1_name",
        "arn:aws:iotanalytics:region:account-id:channel/channel_2_name"
      ]
    }
  ]
}
```

```
}
```

È possibile concedere un accesso granulare o condizionale alle risorse, ad esempio utilizzando uno schema di denominazione con caratteri jolly*. Per ulteriori informazioni, consulta [Aggiungere e rimuovere le policy IAM nella IAM User Guide](#).

Esportazione in AWS IoT Analytics

Per creare un flusso in cui esportare AWS IoT Analytics, le funzioni Lambda [creano un flusso](#) con una definizione di esportazione che include uno o più `IoTAnalyticsConfig` oggetti. Questo oggetto definisce le impostazioni di esportazione, come il canale di destinazione, la dimensione del batch, l'intervallo del batch e la priorità.

Quando le funzioni Lambda ricevono dati dai dispositivi, [aggiungono messaggi](#) che contengono un blob di dati al flusso di destinazione.

Quindi, stream manager esporta i dati in base alle impostazioni del batch e alla priorità definite nelle configurazioni di esportazione dello stream.

Flussi di dati Amazon Kinesis

Stream Manager supporta le esportazioni automatiche verso Amazon Kinesis Data Streams. Kinesis Data Streams viene comunemente utilizzato per aggregare dati di grandi volumi e caricarli in un data warehouse o in un cluster di riduzione delle mappe. Per ulteriori informazioni, consulta [Cos'è Amazon Kinesis Data Streams?](#) nella Amazon Kinesis Developer Guide.

Nel AWS IoT Greengrass Core SDK, le funzioni Lambda utilizzano per definire `KinesisConfig` la configurazione di esportazione per questo tipo di destinazione. Per ulteriori informazioni, consulta il riferimento SDK per la lingua di destinazione:

- [KinesisConfig](#) nell'SDK Python
- [KinesisConfig](#) nell'SDK Java
- [KinesisConfig](#) nell'SDK Node.js

Requisiti

Questa destinazione di esportazione presenta i seguenti requisiti:

- I flussi di destinazione in Kinesis Data Streams devono appartenere allo Regione AWS stesso gruppo Account AWS Greengrass.
- [the section called “Ruolo del gruppo Greengrass”](#) Devono consentire l'`kinesis:PutRecords` autorizzazione per indirizzare i flussi di dati. Per esempio:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kinesis:PutRecords"
      ],
      "Resource": [
        "arn:aws:kinesis:region:account-id:stream/stream_1_name",
        "arn:aws:kinesis:region:account-id:stream/stream_2_name"
      ]
    }
  ]
}
```

È possibile concedere un accesso granulare o condizionale alle risorse, ad esempio utilizzando uno schema di denominazione con caratteri jolly*. Per ulteriori informazioni, consulta [Aggiungere e rimuovere le policy IAM nella IAM User Guide](#).

Esportazione in Kinesis Data Streams

Per creare un flusso che esporta in Kinesis Data Streams, le [funzioni Lambda creano](#) un flusso con una definizione di esportazione che include uno o più oggetti. `KinesisConfig` Questo oggetto definisce le impostazioni di esportazione, come il flusso di dati di destinazione, la dimensione del batch, l'intervallo del batch e la priorità.

Quando le funzioni Lambda ricevono dati dai dispositivi, [aggiungono messaggi](#) che contengono un blob di dati al flusso di destinazione. Quindi, stream manager esporta i dati in base alle impostazioni del batch e alla priorità definite nelle configurazioni di esportazione dello stream.

Stream Manager genera un UUID univoco e casuale come chiave di partizione per ogni record caricato su Amazon Kinesis.

AWS IoT SiteWise proprietà degli asset

Stream manager supporta le esportazioni automatiche verso AWS IoT SiteWise. AWS IoT SiteWise consente di raccogliere, organizzare e analizzare dati provenienti da apparecchiature industriali su larga scala. Per ulteriori informazioni, consulta [Cos'è AWS IoT SiteWise?](#) nella Guida AWS IoT SiteWise per l'utente.

Nel AWS IoT Greengrass Core SDK, le funzioni Lambda utilizzano per definire `IoTSiteWiseConfig` la configurazione di esportazione per questo tipo di destinazione. Per ulteriori informazioni, consulta il riferimento SDK per la lingua di destinazione:

- [IoTSiteWiseConfig](#) nell'SDK Python
- [IoTSiteWiseConfig](#) nell'SDK Java
- [IoTSiteWiseConfig](#) nell'SDK Node.js

Note

AWS fornisce anche la [the section called “IoT SiteWise”](#), che è una soluzione preconfigurata che è possibile utilizzare con i sorgenti OPC-UA.

Requisiti

Questa destinazione di esportazione presenta i seguenti requisiti:

- Le proprietà degli asset di destinazione AWS IoT SiteWise devono appartenere al gruppo Greengrass Account AWS e a Regione AWS quelle del gruppo Greengrass.

Note

Per l'elenco delle regioni che lo AWS IoT SiteWise supportano, consulta gli [AWS IoT SiteWise endpoint e le quote nel AWS Riferimento](#) generale.

- [the section called “Ruolo del gruppo Greengrass”](#) Devono consentire `l'iotsitewise:BatchPutAssetPropertyValue` autorizzazione per indirizzare le proprietà degli asset. La seguente politica di esempio utilizza la chiave `iotsitewise:assetHierarchyPath` condition per concedere l'accesso a una risorsa principale di destinazione e ai suoi figli. Puoi

rimuoverla Condition dalla policy per consentire l'accesso a tutte le tue AWS IoT SiteWise risorse o specificare ARNs singole risorse.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iotsitewise:BatchPutAssetPropertyValue",
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "iotsitewise:assetHierarchyPath": [
            "/root node asset ID",
            "/root node asset ID/*"
          ]
        }
      }
    }
  ]
}
```

Puoi concedere un accesso granulare o condizionale alle risorse, ad esempio utilizzando uno schema di denominazione con caratteri jolly*. Per ulteriori informazioni, consulta [Aggiungere e rimuovere le policy IAM nella IAM User Guide](#).

Per importanti informazioni sulla sicurezza, consulta [BatchPutAssetPropertyValue l'autorizzazione](#) nella Guida AWS IoT SiteWise per l'utente.

Esportazione in AWS IoT SiteWise

Per creare un flusso in cui esportare AWS IoT SiteWise, le funzioni Lambda [creano un flusso](#) con una definizione di esportazione che include uno o più `IoTSiteWiseConfig` oggetti. Questo oggetto definisce le impostazioni di esportazione, come la dimensione del batch, l'intervallo del batch e la priorità.

Quando le funzioni Lambda ricevono dati sulle proprietà degli asset dai dispositivi, aggiungono messaggi che contengono i dati allo stream di destinazione. I messaggi sono `PutAssetPropertyValueEntry` oggetti serializzati in JSON che contengono valori di proprietà

per una o più proprietà delle risorse. Per ulteriori informazioni, consulta [Aggiungi](#) messaggio per le destinazioni di esportazione. AWS IoT SiteWise

Note

Quando invii dati a AWS IoT SiteWise, i tuoi dati devono soddisfare i requisiti dell'`BatchPutAssetPropertyValue` azione. Per ulteriori informazioni, consulta [BatchPutAssetPropertyValue](#) nella documentazione di riferimento dell'API AWS IoT SiteWise.

Quindi, stream manager esporta i dati in base alle impostazioni del batch e alla priorità definite nelle configurazioni di esportazione dello stream.

Puoi modificare le impostazioni dello stream manager e la logica della funzione Lambda per progettare la tua strategia di esportazione. Per esempio:

- Per esportazioni quasi in tempo reale, imposta impostazioni di intervallo e dimensioni dei batch ridotte e aggiungi i dati allo stream quando vengono ricevuti.
- Per ottimizzare il batching, mitigare i vincoli di larghezza di banda o ridurre al minimo i costi, le funzioni Lambda possono raggruppare i punti dati timestamp-quality-value (TQV) ricevuti per una singola proprietà dell'asset prima di aggiungere i dati allo stream. Una strategia consiste nell'inserire in un unico messaggio le voci relative a un massimo di 10 diverse combinazioni proprietà-asset, o alias di proprietà, anziché inviare più di una voce per la stessa proprietà. [Questo aiuta lo stream manager a rimanere entro le quote.](#) [AWS IoT SiteWise](#)

Oggetti Amazon S3

Stream Manager supporta le esportazioni automatiche verso Amazon S3. Puoi usare Amazon S3 per archiviare e recuperare grandi quantità di dati. Per ulteriori informazioni, consulta [Cos'è Amazon S3?](#) nella Guida per sviluppatori di Amazon Simple Storage Service.

Nel AWS IoT Greengrass Core SDK, le funzioni Lambda utilizzano per definire `S3ExportTaskExecutorConfig` la configurazione di esportazione per questo tipo di destinazione. Per ulteriori informazioni, consulta il riferimento SDK per la lingua di destinazione:

- [S3 ExportTaskExecutorConfig](#) nell'SDK Python
- [S3](#) nell'SDK Java [ExportTaskExecutorConfig](#)
- [S3 ExportTaskExecutorConfig](#) nell'SDK Node.js

Requisiti

Questa destinazione di esportazione presenta i seguenti requisiti:

- I bucket Amazon S3 di Target devono appartenere allo stesso gruppo Account AWS Greengrass.
- Se la [containerizzazione predefinita](#) per il gruppo Greengrass è il contenitore Greengrass, è necessario impostare il parametro [STREAM_MANAGER_READ_ONLY_DIRS per utilizzare una directory di file](#) di input che si trova o non si trova nel file system root. /tmp
- Se una funzione Lambda in esecuzione in modalità contenitore Greengrass scrive i file di input nella directory dei file di input, è necessario creare una risorsa di volume locale per la directory e montare la directory nel contenitore con autorizzazioni di scrittura. Ciò garantisce che i file vengano scritti nel file system root e siano visibili all'esterno del contenitore. Per ulteriori informazioni, consulta [Accesso alle risorse locali](#).
- [the section called “Ruolo del gruppo Greengrass”](#) Deve consentire le seguenti autorizzazioni per i bucket di destinazione. Per esempio:

```
{
  "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/*"
      ]
    }
  ]
}
```

È possibile concedere un accesso granulare o condizionale alle risorse, ad esempio utilizzando uno schema di denominazione con caratteri jolly. * Per ulteriori informazioni, consulta [Aggiungere e rimuovere le policy IAM nella IAM User Guide](#).

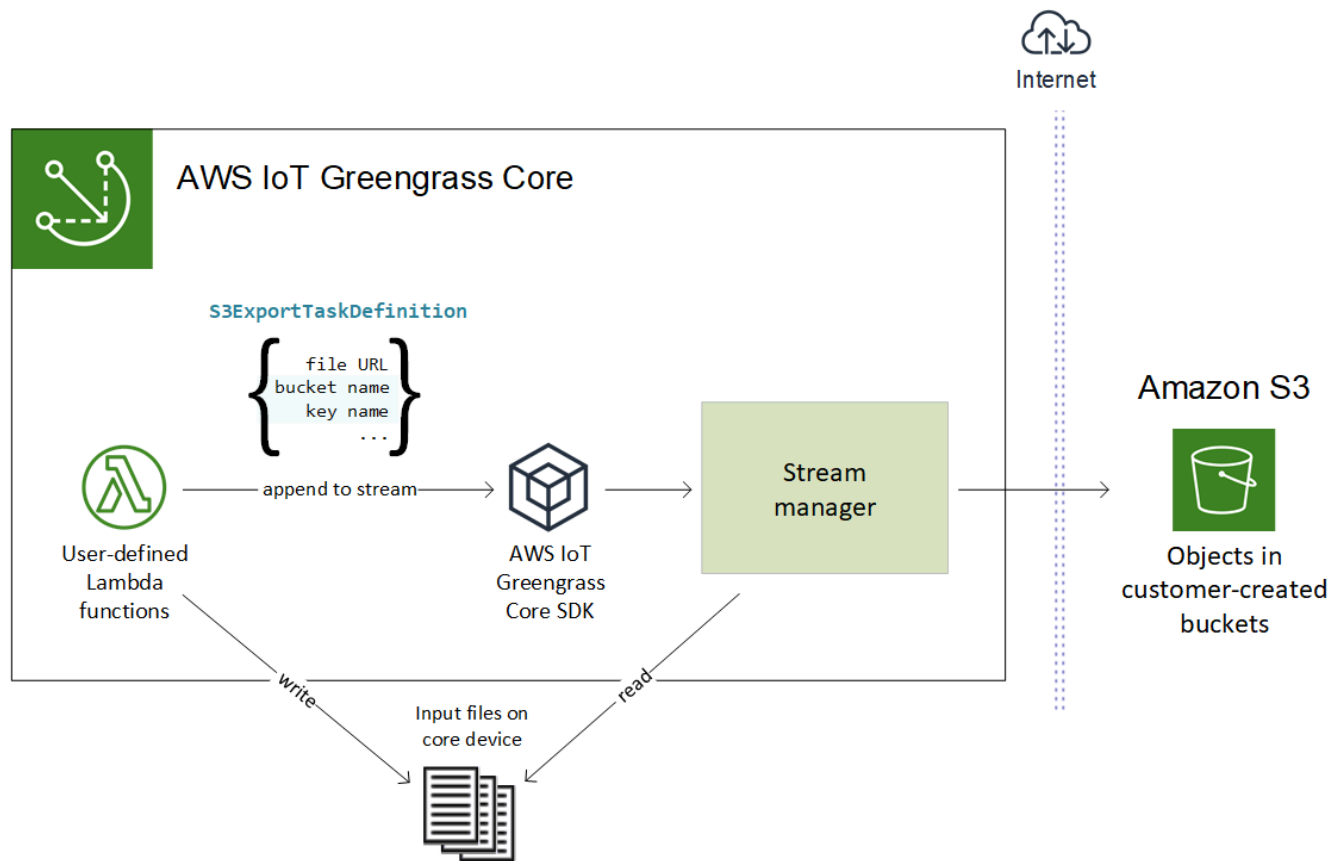
Esportazione su Amazon S3

Per creare uno stream che esporta in Amazon S3, le funzioni Lambda utilizzano l'`S3ExportTaskExecutorConfig` oggetto per configurare la politica di esportazione. La policy definisce le impostazioni di esportazione, come la soglia e la priorità di caricamento multipart. Per le esportazioni di Amazon S3, stream manager carica i dati che legge dai file locali sul dispositivo principale. Per avviare un caricamento, le funzioni Lambda aggiungono un'attività di esportazione allo stream di destinazione. L'attività di esportazione contiene informazioni sul file di input e sull'oggetto Amazon S3 di destinazione. Stream manager esegue le attività nella sequenza in cui vengono aggiunte allo stream.

Note

Il bucket di destinazione deve già esistere nel tuo Account AWS. Se non esiste un oggetto per la chiave specificata, lo stream manager crea l'oggetto per te.

Questo flusso di lavoro di alto livello è illustrato nel diagramma seguente.



Stream Manager utilizza la proprietà della soglia di caricamento [multiparte](#), [l'impostazione della dimensione minima](#) della parte e la dimensione del file di input per determinare come caricare i dati. La soglia di caricamento in più parti deve essere maggiore o uguale alla dimensione minima della parte. Se desideri caricare dati in parallelo, puoi creare più stream.

Le chiavi che specificano gli oggetti Amazon S3 di destinazione possono includere DateFormatter stringhe [Java](#) valide nei segnaposto. `!{timestamp: value}` Puoi utilizzare questi segnaposto con timestamp per partizionare i dati in Amazon S3 in base all'ora in cui i dati del file di input sono stati caricati. Ad esempio, il seguente nome di chiave si risolve in un valore come `my-key/2020/12/31/data.txt`

```
my-key/!{timestamp:YYYY}/!{timestamp:MM}/!{timestamp:dd}/data.txt
```

 Note

Se desideri monitorare lo stato dell'esportazione di uno stream, crea prima uno stream di stato e poi configura il flusso di esportazione per utilizzarlo. Per ulteriori informazioni, consulta [the section called “Monitora le attività di esportazione”](#).

Gestisci i dati di input

Puoi creare codice utilizzato dalle applicazioni IoT per gestire il ciclo di vita dei dati di input. Il seguente flusso di lavoro di esempio mostra come utilizzare le funzioni Lambda per gestire questi dati.

1. Un processo locale riceve dati da dispositivi o periferiche e quindi li scrive su file in una directory sul dispositivo principale. Questi sono i file di input per lo stream manager.

 Note

Per determinare se è necessario configurare l'accesso alla directory dei file di input, vedete il parametro [STREAM_MANAGER_READ_ONLY_DIRS](#).

[Il processo in cui viene eseguito lo stream manager eredita tutte le autorizzazioni del file system dell'identità di accesso predefinita per il gruppo](#). Lo stream manager deve disporre dell'autorizzazione per accedere ai file di input. È possibile utilizzare il `chmod(1)` comando per modificare l'autorizzazione dei file, se necessario.

2. Una funzione Lambda analizza la directory e [aggiunge un'attività di esportazione](#) al flusso di destinazione quando viene creato un nuovo file. L'attività è un `S3ExportTaskDefinition` oggetto serializzato in JSON che specifica l'URL del file di input, il bucket e la chiave Amazon S3 di destinazione e i metadati utente opzionali.
3. Stream Manager legge il file di input ed esporta i dati in Amazon S3 nell'ordine delle attività aggiunte. Il bucket di destinazione deve già esistere nel tuo Account AWS. Se non esiste un oggetto per la chiave specificata, lo stream manager crea l'oggetto per te.
4. La funzione Lambda [legge i messaggi](#) da un flusso di stato per monitorare lo stato dell'esportazione. Una volta completate le attività di esportazione, la funzione Lambda può eliminare i file di input corrispondenti. Per ulteriori informazioni, consulta [the section called “Monitora le attività di esportazione”](#).

Monitora le attività di esportazione

Puoi creare codice utilizzato dalle applicazioni IoT per monitorare lo stato delle esportazioni Amazon S3. Le funzioni Lambda devono creare un flusso di stato e quindi configurare il flusso di esportazione per scrivere aggiornamenti di stato nel flusso di stato. Un singolo flusso di stato può ricevere aggiornamenti di stato da più flussi esportati in Amazon S3.

Innanzitutto, [crea uno stream](#) da utilizzare come flusso di stato. Puoi configurare le dimensioni e le politiche di conservazione dello stream per controllare la durata dei messaggi di stato. Per esempio:

- `PersistenceImposta` su `Memory` se non desideri archiviare i messaggi di stato.
- `Imposta StrategyOnFull` in `OverwriteOldestData` modo che i nuovi messaggi di stato non vadano persi.

Quindi, crea o aggiorna il flusso di esportazione per utilizzare il flusso di stato. In particolare, imposta la proprietà di configurazione dello stato della configurazione di `S3ExportTaskExecutorConfig` esportazione dello stream. Questo dice al gestore dello stream di scrivere messaggi di stato sulle attività di esportazione nello stream di stato. Nell'`StatusConfig` oggetto, specifica il nome del flusso di stato e il livello di verbosità. I seguenti valori supportati vanno da `least verbose (ERROR)` a `most verbose ()`. `TRACE` Il valore predefinito è `INFO`.

- `ERROR`
- `WARN`
- `INFO`
- `DEBUG`
- `TRACE`

Il seguente flusso di lavoro di esempio mostra come le funzioni Lambda potrebbero utilizzare un flusso di stato per monitorare lo stato delle esportazioni.

1. Come descritto nel flusso di lavoro precedente, una funzione Lambda [aggiunge un'attività di esportazione](#) a uno stream configurato per scrivere messaggi di stato sulle attività di esportazione in un flusso di stato. L'operazione di aggiunta restituisce un numero di sequenza che rappresenta l'ID dell'attività.

2. Una funzione Lambda [legge i messaggi](#) in sequenza dallo stream di stato, quindi filtra i messaggi in base al nome del flusso e all'ID dell'attività o in base a una proprietà dell'attività di esportazione dal contesto del messaggio. Ad esempio, la funzione Lambda può filtrare in base all'URL del file di input dell'attività di esportazione, che è rappresentato dall'`S3ExportTaskDefinition` oggetto nel contesto del messaggio.

I seguenti codici di stato indicano che un'attività di esportazione ha raggiunto lo stato di completamento:

- **Success.** Il caricamento è stato completato con successo.
- **Failure.** Lo stream manager ha riscontrato un errore, ad esempio, il bucket specificato non esiste. Dopo aver risolto il problema, puoi aggiungere nuovamente l'attività di esportazione allo stream.
- **Canceled.** L'attività è stata interrotta perché la definizione dello stream o dell'esportazione è stata eliminata o il periodo time-to-live (TTL) dell'attività è scaduto.

Note

L'attività potrebbe anche avere lo stato `InProgress`. `Warning` Stream manager emette avvisi quando un evento restituisce un errore che non influisce sull'esecuzione dell'attività. Ad esempio, la mancata pulizia di un caricamento parziale interrotto restituisce un avviso.

3. Una volta completate le attività di esportazione, la funzione Lambda può eliminare i file di input corrispondenti.

L'esempio seguente mostra come una funzione Lambda potrebbe leggere ed elaborare i messaggi di stato.

Python

```
import time
from greengrasssdk.stream_manager import (
    ReadMessagesOptions,
    Status,
    StatusConfig,
    StatusLevel,
    StatusMessage,
    StreamManagerClient,
)
```

```
from greengrasssdk.stream_manager.util import Util

client = StreamManagerClient()

try:
    # Read the statuses from the export status stream
    is_file_uploaded_to_s3 = False
    while not is_file_uploaded_to_s3:
        try:
            messages_list = client.read_messages(
                "StatusStreamName", ReadMessagesOptions(min_message_count=1,
read_timeout_millis=1000)
            )
            for message in messages_list:
                # Deserialize the status message first.
                status_message = Util.deserialize_json_bytes_to_obj(message.payload,
StatusMessage)

                # Check the status of the status message. If the status is
"Success",
                # the file was successfully uploaded to S3.
                # If the status was either "Failure" or "Cancelled", the server was
unable to upload the file to S3.
                # We will print the message for why the upload to S3 failed from the
status message.
                # If the status was "InProgress", the status indicates that the
server has started uploading
                # the S3 task.
                if status_message.status == Status.Success:
                    logger.info("Successfully uploaded file at path " + file_url + "
to S3.")
                    is_file_uploaded_to_s3 = True
                elif status_message.status == Status.Failure or
status_message.status == Status.Canceled:
                    logger.info(
                        "Unable to upload file at path " + file_url + " to S3.
Message: " + status_message.message
                    )
                    is_file_uploaded_to_s3 = True
                time.sleep(5)
            except StreamManagerException:
                logger.exception("Exception while running")
except StreamManagerException:
    pass
```

```
# Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
# Properly handle errors.
```

[Riferimento all'SDK Python: read_messages | StatusMessage](#)

Java

```
import com.amazonaws.greengrass.streammanager.client.StreamManagerClient;
import com.amazonaws.greengrass.streammanager.client.utils.ValidateAndSerialize;
import com.amazonaws.greengrass.streammanager.model.ReadMessagesOptions;
import com.amazonaws.greengrass.streammanager.model.Status;
import com.amazonaws.greengrass.streammanager.model.StatusConfig;
import com.amazonaws.greengrass.streammanager.model.StatusLevel;
import com.amazonaws.greengrass.streammanager.model.StatusMessage;

try (final StreamManagerClient client =
    GreengrassClientBuilder.streamManagerClient().build()) {
    try {
        boolean isS3UploadComplete = false;
        while (!isS3UploadComplete) {
            try {
                // Read the statuses from the export status stream
                List<Message> messages = client.readMessages("StatusStreamName",
                    new
                ReadMessagesOptions().withMinMessageCount(1L).withReadTimeoutMillis(1000L));
                for (Message message : messages) {
                    // Deserialize the status message first.
                    StatusMessage statusMessage =
                ValidateAndSerialize.deserializeJsonBytesToObj(message.getPayload(),
                StatusMessage.class);
                    // Check the status of the status message. If the status is
                "Success", the file was successfully uploaded to S3.
                    // If the status was either "Failure" or "Canceled", the server
                was unable to upload the file to S3.
                    // We will print the message for why the upload to S3 failed
                from the status message.
                    // If the status was "InProgress", the status indicates that the
                server has started uploading the S3 task.
                    if (Status.Success.equals(statusMessage.getStatus())) {
                        System.out.println("Successfully uploaded file at path " +
                FILE_URL + " to S3.");
                        isS3UploadComplete = true;
                    }
                }
            } catch (Exception e) {
                // Handle exception
            }
        }
    } catch (Exception e) {
        // Handle exception
    }
}
```

```

        } 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.
}
}

```

Riferimento a Java SDK: [ReadMessages](#) | [StatusMessage](#)

Node.js

```

const {
    StreamManagerClient, ReadMessagesOptions,
    Status, StatusConfig, StatusLevel, StatusMessage,
    util,
} = require('aws-greengrass-core-sdk').StreamManager;

const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        let isS3UploadComplete = false;
        while (!isS3UploadComplete) {
            try {
                // Read the statuses from the export status stream
                const messages = await c.readMessages("StatusStreamName",
                    new ReadMessagesOptions()
                        .withMinMessageCount(1)
                        .withReadTimeoutMillis(1000));
            }

```

```

        messages.forEach((message) => {
            // Deserialize the status message first.
            const statusMessage =
util.deserializeJsonBytesToObj(message.payload, StatusMessage);
            // Check the status of the status message. If the status is
'Success', the file was successfully uploaded to S3.
            // If the status was either 'Failure' or 'Cancelled', the server
was unable to upload the file to S3.
            // We will print the message for why the upload to S3 failed
from the status message.
            // If the status was "InProgress", the status indicates that the
server has started uploading the S3 task.
            if (statusMessage.status === Status.Success) {
                console.log(`Successfully uploaded file at path ${FILE_URL}
to S3.`);
                isS3UploadComplete = true;
            } else if (statusMessage.status === Status.Failure ||
statusMessage.status === Status.Canceled) {
                console.log(`Unable to upload file at path ${FILE_URL} to
S3. Message: ${statusMessage.message}`);
                isS3UploadComplete = true;
            }
        });
        // Sleep for sometime for the S3 upload task to complete before
trying to read the status message.
        await new Promise((r) => setTimeout(r, 5000));
    } catch (e) {
        // Ignored
    }
} catch (e) {
    // Properly handle errors.
}
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});

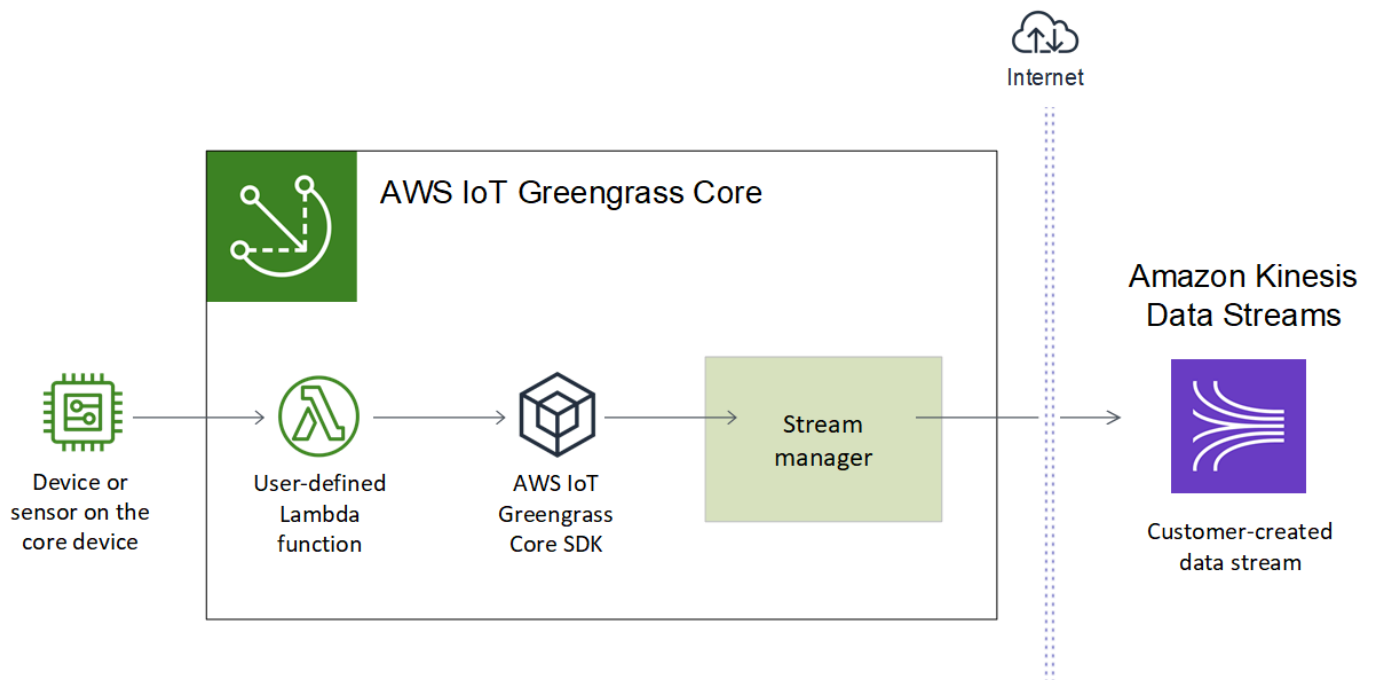
```

Riferimento all'SDK Node.js: [ReadMessages](#) | [StatusMessage](#)

Esportazione di flussi di dati su Cloud AWS (console)

Questo tutorial mostra come utilizzare la AWS IoT console per configurare e distribuire un AWS IoT Greengrass gruppo con stream manager abilitato. Il gruppo contiene una funzione Lambda definita dall'utente che scrive su uno stream in stream manager, che viene quindi esportata automaticamente in Cloud AWS.

Stream Manager rende più efficienti e affidabili l'acquisizione, l'elaborazione e l'esportazione di flussi di dati ad alto volume. In questo tutorial, crei una funzione `TransferStream` Lambda che utilizza dati IoT. La funzione Lambda utilizza AWS IoT Greengrass Core SDK per creare uno stream in stream manager e quindi leggerlo e scriverlo. Stream Manager esporta quindi il flusso in Kinesis Data Streams. Il diagramma seguente mostra questo flusso di lavoro.



L'obiettivo di questo tutorial è mostrare come le funzioni Lambda definite dall'utente utilizzano `StreamManagerClient` l'oggetto nel Core SDK per interagire con AWS IoT Greengrass lo stream manager. Per semplicità, la funzione Python Lambda creata per questo tutorial genera dati di dispositivo simulati.

Prerequisiti

Per completare questo tutorial, è necessario quanto segue:

- Un gruppo Greengrass e un core Greengrass (v1.10 o successivo). Per informazioni su come creare un gruppo e un core Greengrass, vedere [Iniziare con AWS IoT Greengrass](#). Il tutorial *Getting Started* include anche i passaggi per l'installazione del software AWS IoT Greengrass Core.

Note

Stream manager non è supportato nelle OpenWrt distribuzioni.

- Java 8 Runtime (JDK 8) installato sul dispositivo principale.
 - Per distribuzioni basate su Debian (incluso Raspbian) o distribuzioni basate su Ubuntu, eseguire il comando seguente:

```
sudo apt install openjdk-8-jdk
```

- Per distribuzioni basate su Red Hat (incluso Amazon Linux), eseguire il comando seguente:

```
sudo yum install java-1.8.0-openjdk
```

Per ulteriori informazioni, consulta [How to download and install prebuilt OpenJDK packages](#) nella documentazione di OpenJDK.

- AWS IoT Greengrass Core SDK per Python v1.5.0 o successivo. Per utilizzarlo `StreamManagerClient` nel AWS IoT Greengrass Core SDK per Python, devi:
 - Installa Python 3.7 o versione successiva sul dispositivo principale.
 - Includi l'SDK e le sue dipendenze nel pacchetto di distribuzione della funzione Lambda. Le istruzioni vengono fornite in questo tutorial.

Tip

È possibile utilizzare `StreamManagerClient` con Java o NodeJS. Per esempio di codice, consulta [AWS IoT Greengrass Core SDK per Java](#) [AWS IoT Greengrass e Core SDK per Node.js](#) su GitHub

- Un flusso di destinazione denominato **MyKinesisStream** creato in Amazon Kinesis Data Streams nella Regione AWS stesso gruppo Greengrass. Per ulteriori informazioni, consulta [Create a stream](#) nella Amazon Kinesis Developer Guide.

 Note

In questo tutorial, lo stream manager esporta i dati in Kinesis Data Streams, con conseguente addebito a carico dell'utente. Account AWS Per informazioni sui prezzi, consulta la pagina dei prezzi di [Kinesis Data Streams](#).

Per evitare costi aggiuntivi, puoi eseguire questo tutorial senza creare un flusso di dati Kinesis. In questo caso, controlla i log per vedere che lo stream manager ha tentato di esportare lo stream in Kinesis Data Streams.

- Una policy IAM aggiunta a [the section called “Ruolo del gruppo Greengrass”](#) quella che consente l'`kinesis:PutRecords` sul flusso di dati di destinazione, come mostrato nell'esempio seguente:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kinesis:PutRecords"
      ],
      "Resource": [
        "arn:aws:kinesis:region:account-id:stream/MyKinesisStream"
      ]
    }
  ]
}
```

Il tutorial include le seguenti fasi di alto livello:

1. [Creare un pacchetto di distribuzione della funzione Lambda](#)
2. [Creazione di una funzione Lambda](#)
3. [Aggiunta di una funzione al gruppo](#)
4. [Abilitazione di Stream Manager](#)
5. [Configurazione della registrazione locale](#)
6. [Distribuzione del gruppo.](#)
7. [Eseguire il test dell'applicazione](#)

Il completamento di questo tutorial richiede circa 20 minuti.

Fase 1: Creare un pacchetto di distribuzione della funzione Lambda

In questo passaggio, crei un pacchetto di distribuzione della funzione Lambda che contiene il codice della funzione Python e le dipendenze. Carichi questo pacchetto più tardi quando crei la funzione Lambda in AWS Lambda. La funzione Lambda utilizza AWS IoT Greengrass Core SDK per creare e interagire con flussi locali.

Note

Le funzioni Lambda definite dall'utente devono utilizzare [Core SDK per interagire con AWS IoT Greengrass lo](#) stream manager. Per ulteriori informazioni sui requisiti di Stream Manager di Greengrass, consulta l'argomento relativo ai [requisiti di Stream Manager di Greengrass](#).

1. Scarica il [AWS IoT Greengrass Core SDK per Python](#) v1.5.0 o versione successiva.
2. Decomprimere il pacchetto scaricato per ottenere l'SDK. Il kit SDK è la cartella `greengrasssdk`.
3. Installa le dipendenze dei pacchetti da includere con l'SDK nel pacchetto di distribuzione delle funzioni Lambda.
 1. Passare alla directory SDK che contiene il file `requirements.txt`. Questo file elenca le dipendenze.
 2. Installare le dipendenze SDK. Ad esempio, eseguire il comando `pip` seguente per installarle nella directory corrente:

```
pip install --target . -r requirements.txt
```

4. Salvare la seguente funzione del codice Python nel file locale `transfer_stream.py`.

Tip

Per esempio di codice che utilizza Java e NodeJS, consulta AWS IoT Greengrass Core SDK [per AWS IoT Greengrass Java e Core SDK per Node.js](#) su GitHub

```
import asyncio
import logging
```

```
import random
import time

from greengrasssdk.stream_manager import (
    ExportDefinition,
    KinesisConfig,
    MessageStreamDefinition,
    ReadMessagesOptions,
    ResourceNotFoundException,
    StrategyOnFull,
    StreamManagerClient,
)

# This example creates a local stream named "SomeStream".
# It starts writing data into that stream and then stream manager automatically
# exports
# the data to a customer-created Kinesis data stream named "MyKinesisStream".
# This example runs forever until the program is stopped.

# The size of the local stream on disk will not exceed the default (which is 256
# MB).
# Any data appended after the stream reaches the size limit continues to be
# appended, and
# stream manager deletes the oldest data until the total stream size is back under
# 256 MB.
# The Kinesis data stream in the cloud has no such bound, so all the data from this
# script is
# uploaded to Kinesis and you will be charged for that usage.

def main(logger):
    try:
        stream_name = "SomeStream"
        kinesis_stream_name = "MyKinesisStream"

        # Create a client for the StreamManager
        client = StreamManagerClient()

        # Try deleting the stream (if it exists) so that we have a fresh start
        try:
            client.delete_message_stream(stream_name=stream_name)
        except ResourceNotFoundException:
            pass
```

```
        exports = ExportDefinition(
            kinesis=[KinesisConfig(identifier="KinesisExport" + stream_name,
kinesis_stream_name=kinesis_stream_name)]
        )
        client.create_message_stream(
            MessageStreamDefinition(
                name=stream_name,
strategy_on_full=StrategyOnFull.OverwriteOldestData, export_definition=exports
            )
        )

        # Append two messages and print their sequence numbers
        logger.info(
            "Successfully appended message to stream with sequence number %d",
            client.append_message(stream_name, "ABCDEFGHJKLMNOP".encode("utf-8")),
        )
        logger.info(
            "Successfully appended message to stream with sequence number %d",
            client.append_message(stream_name, "QRSTUVWXYZ".encode("utf-8")),
        )

        # Try reading the two messages we just appended and print them out
        logger.info(
            "Successfully read 2 messages: %s",
            client.read_messages(stream_name,
ReadMessagesOptions(min_message_count=2, read_timeout_millis=1000)),
        )

        logger.info("Now going to start writing random integers between 0 and 1000
to the stream")
        # Now start putting in random data between 0 and 1000 to emulate device
sensor input
        while True:
            logger.debug("Appending new random integer to stream")
            client.append_message(stream_name, random.randint(0,
1000).to_bytes(length=4, signed=True, byteorder="big"))
            time.sleep(1)

        except asyncio.TimeoutError:
            logger.exception("Timed out while executing")
        except Exception:
            logger.exception("Exception while running")
```

```
def function_handler(event, context):  
    return  
  
logging.basicConfig(level=logging.INFO)  
# Start up this sample code  
main(logger=logging.getLogger())
```

5. Comprimerle le voci seguenti nel file `transfer_stream_python.zip`. Questo è il tuo pacchetto di implementazione della funzione Lambda.

- `transfer_stream.py`. La logica dell'app.
- `greengrasssdk`. Libreria richiesta per le funzioni Python Greengrass Lambda che pubblicano messaggi MQTT.

Le [operazioni di Stream Manager](#) sono disponibili nella versione 1.5.0 o successiva di AWS IoT Greengrass Core SDK for Python.

- Le dipendenze installate per AWS IoT Greengrass Core SDK for Python (ad esempio, le `directory`). `cbor2`

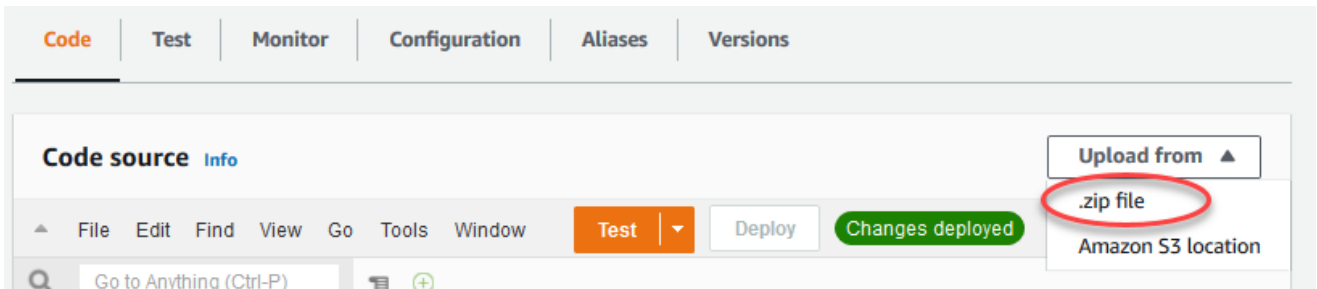
Quando crei il file zip, includi solo questi elementi, non la cartella che li contiene.

Passaggio 2: creazione di una funzione Lambda

In questo passaggio, si utilizza la AWS Lambda console per creare una funzione Lambda e configurarla per utilizzare il pacchetto di distribuzione. In seguito, pubblicherai una versione della funzione e creerai un alias.

1. Innanzitutto, crea la funzione Lambda.
 - a. In AWS Management Console, scegli Servizi e apri la AWS Lambda console.
 - b. Scegli Crea funzione, quindi scegli Autore da zero.
 - c. Nella sezione Basic information (Informazioni di base), specifica i seguenti valori:
 - Nel campo Function name (Nome funzione), immettere **TransferStream**.
 - In Runtime, scegliere Python 3.7.

- Per le autorizzazioni, mantieni l'impostazione predefinita. Questo crea un ruolo di esecuzione che concede le autorizzazioni Lambda di base. Questo ruolo non viene utilizzato da AWS IoT Greengrass
- d. Nella parte inferiore della pagina, scegli Crea funzione.
2. Quindi, registra il gestore e carica il pacchetto di distribuzione della funzione Lambda.
 - a. Nella scheda Codice, in Codice sorgente, scegli Carica da. Dal menu a discesa, scegli un file.zip.



- b. Scegli Carica, quindi scegli il pacchetto di `transfer_stream_python.zip` distribuzione. Quindi, scegliere Save (Salva).
- c. Nella scheda Codice della funzione, in Impostazioni di runtime, scegliete Modifica, quindi immettete i seguenti valori.
 - In Runtime, scegliere Python 3.7.
 - In Handler (Gestore), immetti **`transfer_stream.function_handler`**
- d. Seleziona Salva.

Note

Il pulsante Test sulla AWS Lambda console non funziona con questa funzione. Il AWS IoT Greengrass Core SDK non contiene moduli necessari per eseguire le funzioni Greengrass Lambda in modo indipendente nella console. AWS Lambda Questi moduli (ad esempio, `greengrass_common`) vengono forniti alle funzioni dopo essere stati distribuiti nel core Greengrass.

3. Ora, pubblica la prima versione della tua funzione Lambda e crea un [alias per la](#) versione.

 Note

I gruppi Greengrass possono fare riferimento a una funzione Lambda tramite alias (consigliato) o per versione. L'utilizzo di un alias semplifica la gestione degli aggiornamenti del codice perché non è necessario modificare la tabella di sottoscrizione o la definizione del gruppo quando il codice della funzione viene aggiornato. Invece, è sufficiente indirizzare l'alias alla nuova versione della funzione.

- a. Nel menu Actions (Operazioni), seleziona Publish new version (Pubblica nuova versione).
- b. Per Version description (Descrizione versione), immettere **First version**, quindi scegliere Publish (Pubblica).
- c. Nella pagina di configurazione TransferStream: 1, dal menu Azioni, scegli Crea alias.
- d. Nella pagina Create a new alias (Crea un nuovo alias), utilizza i seguenti valori:
 - Per Nome, immetti **GG_TransferStream**.
 - In Version (Versione), selezionare 1.

 Note

AWS IoT Greengrass non supporta gli alias Lambda per le versioni \$LATEST.

- e. Scegli Create (Crea) .

Ora sei pronto per aggiungere la funzione Lambda al tuo gruppo Greengrass.

Fase 3: Aggiungere una funzione Lambda al gruppo Greengrass

In questo passaggio, aggiungi la funzione Lambda al gruppo e quindi configuri il ciclo di vita e le variabili di ambiente. Per ulteriori informazioni, consulta [the section called “Controllo dell'esecuzione della funzione Greengrass Lambda”](#).

1. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).
2. Scegliere il gruppo target.

3. Nella pagina di configurazione del gruppo, scegli la scheda Funzioni Lambda.
4. In Funzioni My Lambda, scegli Aggiungi.
5. Nella pagina Aggiungi funzione Lambda, scegli la funzione Lambda per la tua funzione Lambda.
6. Per la versione Lambda, scegli Alias:gg_. TransferStream

Ora configurate le proprietà che determinano il comportamento della funzione Lambda nel gruppo Greengrass.

7. Nella sezione di configurazione della funzione Lambda, apporta le seguenti modifiche:
 - Impostare Memory limit (Limite di memoria) su 32 MB.
 - Per Pinned, scegli True.

Note

Una funzione Lambda di lunga durata (o bloccata) si avvia automaticamente dopo l'avvio e continua AWS IoT Greengrass a funzionare nel proprio contenitore. Ciò è in contrasto con una funzione Lambda su richiesta, che si avvia quando viene richiamata e si interrompe quando non ci sono più attività da eseguire. Per ulteriori informazioni, consulta [the section called “Configurazione del ciclo di vita”](#).

8. Scegli Aggiungi funzione Lambda.

Fase 4: abilitazione di Stream Manager

In questa fase, assicurati che stream manager sia abilitato.

1. Nella pagina di configurazione del gruppo, scegli la scheda Funzioni Lambda.
2. In Funzioni System Lambda, seleziona Stream manager e controlla lo stato. Se disabilitato, scegliere Edit (Modifica). Quindi, scegliere Enable (Abilita) e Save (Salva). Puoi utilizzare le impostazioni dei parametri predefinite per questo tutorial. Per ulteriori informazioni, consulta [the section called “Configurazione di Stream Manager”](#).

Note

Quando si utilizza la console per abilitare lo stream manager e distribuire il gruppo, la dimensione della memoria per lo stream manager è impostata su 4194304 KB (4 GB) per

impostazione predefinita. Ti consigliamo di impostare la dimensione della memoria su almeno 128000 KB.

Fase 5: configurazione della registrazione locale

In questo passaggio, configuri i componenti di AWS IoT Greengrass sistema, le funzioni Lambda definite dall'utente e i connettori del gruppo per scrivere i log nel file system del dispositivo principale. Puoi utilizzare i log per risolvere eventuali problemi che potrebbero verificarsi. Per ulteriori informazioni, consulta [the section called “Monitoraggio con i log AWS IoT Greengrass”](#).

1. In Local logs configuration (Configurazione log locale), verificare se è configurata la registrazione locale.
2. Se i log non sono configurati per i componenti del sistema Greengrass o per le funzioni Lambda definite dall'utente, scegli Modifica.
3. Scegli il livello di registro delle funzioni User Lambda e il livello di registro del sistema Greengrass.
4. Mantenere i valori predefiniti per il livello di registrazione e il limite di spazio su disco e scegliere Save (Salva).

Fase 6: distribuzione del gruppo Greengrass

Distribuire il gruppo al nuovo dispositivo core.

1. Assicurati che il AWS IoT Greengrass core sia in funzione. Esegui i seguenti comandi nel terminale di Raspberry Pi in base alle esigenze.
 - a. Per controllare se il daemon è in esecuzione:

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

Se l'output contiene una voce root per `/greengrass/ggc/packages/ggc-version/bin/daemon`, allora il daemon è in esecuzione.

 Note

La versione indicata nel percorso dipende dalla versione del software AWS IoT Greengrass Core installata sul dispositivo principale.

- b. Per avviare il demone:

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

2. Nella pagina di configurazione del gruppo, scegli Deploy.
3.
 - a. Nella scheda Funzioni Lambda, nella sezione Funzioni System Lambda, seleziona Rilevatore IP e scegli Modifica.
 - b. Nella finestra di dialogo Modifica impostazioni del rilevatore IP, seleziona Rileva e sostituisci automaticamente gli endpoint del broker MQTT.
 - c. Seleziona Salva.

Questo consente ai dispositivi di acquisire automaticamente informazioni di base sulla connettività, come, ad esempio indirizzo IP, DNS e numero della porta. Il rilevamento automatico è consigliato, ma supporta AWS IoT Greengrass anche gli endpoint specificati manualmente. Ti viene chiesto il metodo di individuazione solo la prima volta che il gruppo viene distribuito.

 Note

Se richiesto, concedi l'autorizzazione a creare il ruolo di [servizio Greengrass](#) e associarlo al Account AWS tuo ruolo attuale. Regione AWS Questo ruolo consente di accedere AWS IoT Greengrass alle tue risorse nei AWS servizi.

Nella pagina Deployments (Distribuzioni) vengono visualizzati il timestamp della distribuzione, l'ID versione e lo stato. Una volta completata, lo stato visualizzato per la distribuzione dovrebbe essere Completato.

Per la risoluzione dei problemi, consultare [Risoluzione dei problemi](#).

Fase 7: test dell'applicazione

La funzione `TransferStream` Lambda genera dati simulati del dispositivo. Scrive i dati in un flusso che viene esportato da stream manager nel flusso di dati Kinesis target.

1. Nella console Amazon Kinesis, in Kinesis data stream, scegli. `MyKinesisStream`

Note

Se il tutorial è stato eseguito senza un flusso di dati Kinesis di destinazione, [controllare il file di log](#) per stream manager (`GGStreamManager`). Se contiene `export stream MyKinesisStream doesn't exist` in un messaggio di errore, il test viene superato. Questo errore indica che il servizio ha tentato di esportare nel flusso ma il flusso non esiste.

2. Nella `MyKinesisStream` pagina, scegli Monitoraggio. Se il test viene superato, i dati sono visibili nei grafici Put Records (Inserisci record). A seconda della connessione, potrebbe essere necessario fino a un minuto prima che i dati vengano visualizzati.

Important

Al termine della sessione di test, eliminare il flusso di dati Kinesis per evitare di incorrere in ulteriori costi.

In alternativa, eseguire i comandi seguenti per arrestare il daemon Greengrass. Ciò impedisce al core di inviare messaggi fino a quando non si è pronti a continuare il test.

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

3. Rimuovi la funzione `TransferStreamLambda` dal core.
 - a. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).
 - b. In Gruppi Greengrass, scegli il tuo gruppo.
 - c. Nella pagina Lambdas, scegli i puntini di sospensione (...) per la `TransferStream` funzione, quindi scegli Rimuovi funzione.
 - d. In Actions (Operazioni), scegliere Deploy (Distribuisci).

Per visualizzare le informazioni di registrazione o risolvere problemi con i flussi, controllare i log per le funzioni `TransferStream` e `GGStreamManager`. È necessario disporre root delle autorizzazioni per leggere i AWS IoT Greengrass log sul file system.

- `TransferStream` scrive le voci di log in `greengrass-root/ggc/var/log/user/region/account-id/TransferStream.log`.
- `GGStreamManager` scrive le voci di log in `greengrass-root/ggc/var/log/system/GGStreamManager.log`.

Se hai bisogno di ulteriori informazioni sulla risoluzione dei problemi, puoi [impostare il livello di registrazione](#) per i log User Lambda su Debug logs e quindi distribuire nuovamente il gruppo.

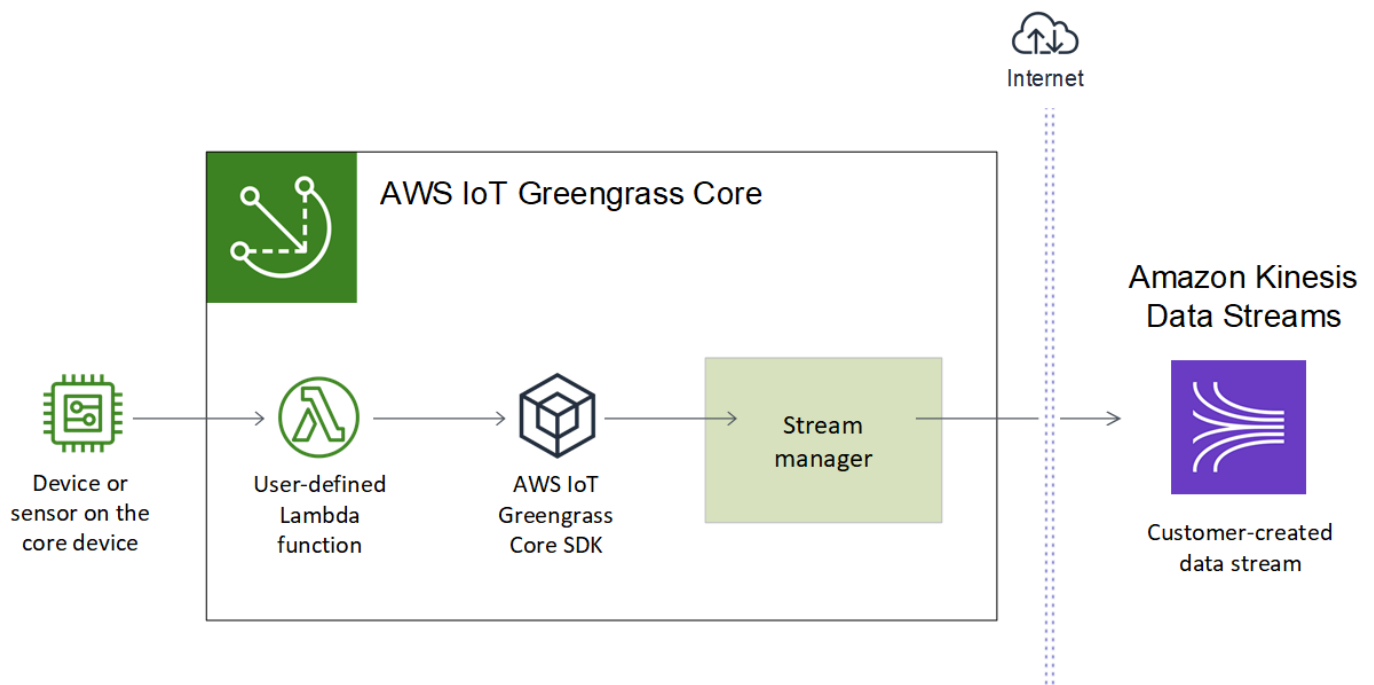
Consulta anche

- [Gestione dei flussi di dati](#)
- [the section called “Configurazione di Stream Manager”](#)
- [the section called “Utilizzalo StreamManagerClient per lavorare con gli stream”](#)
- [the section called “Esporta le configurazioni per le destinazioni supportate Cloud AWS”](#)
- [the section called “Esportazione di flussi di dati \(CLI\)”](#)

Esportazione di flussi di dati su Cloud AWS (CLI)

Questo tutorial mostra come utilizzare il AWS CLI per configurare e distribuire un AWS IoT Greengrass gruppo con lo stream manager abilitato. Il gruppo contiene una funzione Lambda definita dall'utente che scrive su uno stream in stream manager, che viene quindi esportata automaticamente in Cloud AWS

Stream Manager rende più efficienti e affidabili l'acquisizione, l'elaborazione e l'esportazione di flussi di dati ad alto volume. In questo tutorial, crei una funzione `TransferStream` Lambda che utilizza dati IoT. La funzione Lambda utilizza AWS IoT Greengrass Core SDK per creare uno stream in stream manager e quindi leggerlo e scriverlo. Stream Manager esporta quindi il flusso in Kinesis Data Streams. Il diagramma seguente mostra questo flusso di lavoro.



L'obiettivo di questo tutorial è mostrare come le funzioni Lambda definite dall'utente utilizzano `StreamManagerClient` l'oggetto nel Core SDK per interagire con AWS IoT Greengrass lo stream manager. Per semplicità, la funzione Python Lambda creata per questo tutorial genera dati di dispositivo simulati.

Quando si utilizza l' AWS IoT Greengrass API, che include i comandi Greengrass in AWS CLI, per creare un gruppo, lo stream manager è disabilitato per impostazione predefinita. Per abilitare lo stream manager sul tuo core, [crei una versione di definizione di funzione](#) che include la funzione `GGStreamManager` Lambda di sistema e una versione di gruppo che fa riferimento alla nuova versione di definizione della funzione. Quindi, distribuisce il flusso.

Prerequisiti

Per completare questo tutorial, è necessario quanto segue:

- Un gruppo Greengrass e un core Greengrass (v1.10 o successivo). Per informazioni su come creare un gruppo e un core Greengrass, vedere. [Iniziare con AWS IoT Greengrass](#) Il tutorial Getting Started include anche i passaggi per l'installazione del software AWS IoT Greengrass Core.

 Note

Stream manager non è supportato nelle OpenWrt distribuzioni.

- Java 8 Runtime (JDK 8) installato sul dispositivo principale.
- Per distribuzioni basate su Debian (incluso Raspbian) o distribuzioni basate su Ubuntu, eseguire il comando seguente:

```
sudo apt install openjdk-8-jdk
```

- Per distribuzioni basate su Red Hat (incluso Amazon Linux), eseguire il comando seguente:

```
sudo yum install java-1.8.0-openjdk
```

Per ulteriori informazioni, consulta [How to download and install prebuilt OpenJDK packages](#) nella documentazione di OpenJDK.

- AWS IoT Greengrass Core SDK per Python v1.5.0 o successivo. Per utilizzarlo `StreamManagerClient` nel AWS IoT Greengrass Core SDK per Python, devi:
 - Installa Python 3.7 o versione successiva sul dispositivo principale.
 - Includi l'SDK e le sue dipendenze nel pacchetto di distribuzione della funzione Lambda. Le istruzioni vengono fornite in questo tutorial.

 Tip

È possibile utilizzare `StreamManagerClient` con Java o NodeJS. Per esempio di codice, consulta [AWS IoT Greengrass Core SDK per Java](#) [AWS IoT Greengrass e Core SDK per Node.js](#) su GitHub

- Un flusso di destinazione denominato **MyKinesisStream** creato in Amazon Kinesis Data Streams nella Regione AWS stesso gruppo Greengrass. Per ulteriori informazioni, consulta [Create a stream](#) nella Amazon Kinesis Developer Guide.

 Note

In questo tutorial, lo stream manager esporta i dati in Kinesis Data Streams, con conseguente addebito a carico dell'utente. Account AWS Per informazioni sui prezzi, consulta la pagina dei prezzi di [Kinesis Data Streams](#).

Per evitare costi aggiuntivi, puoi eseguire questo tutorial senza creare un flusso di dati Kinesis. In questo caso, controlla i log per vedere che lo stream manager ha tentato di esportare lo stream in Kinesis Data Streams.

- Una policy IAM aggiunta a [the section called “Ruolo del gruppo Greengrass”](#) quella che consente l'`kinesis:PutRecords` sul flusso di dati di destinazione, come mostrato nell'esempio seguente:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kinesis:PutRecords"
      ],
      "Resource": [
        "arn:aws:kinesis:region:account-id:stream/MyKinesisStream"
      ]
    }
  ]
}
```

- AWS CLI Installato e configurato sul tuo computer. Per ulteriori informazioni, vedere [Installazione AWS Command Line Interface e configurazione di AWS CLI nella Guida per l'AWS Command Line Interface](#) utente.

I comandi di esempio in questo tutorial sono scritti per Linux e altri sistemi basati su Unix. Se usi Windows, vedi [Specificazione dei valori dei parametri per l'interfaccia della AWS riga di comando](#) per ulteriori informazioni sulle differenze di sintassi.

Se il comando include una stringa JSON, il tutorial fornisce un esempio che ha JSON in un'unica riga. Su alcuni sistemi, potrebbe essere più efficiente modificare ed eseguire comandi utilizzando questo formato.

Il tutorial include le seguenti fasi di alto livello:

1. [Creare un pacchetto di distribuzione della funzione Lambda](#)
2. [Creazione di una funzione Lambda](#)
3. [Creazione della versione e della definizione della funzione](#)
4. [Creazione di una definizione e di una versione di logger](#)
5. [Ottenimento dell'ARN della versione di definizione del core](#)
6. [Creazione di una versione del gruppo](#)
7. [Crea distribuzione](#)
8. [Eseguire il test dell'applicazione](#)

Il completamento di questo tutorial richiede circa 30 minuti.

Fase 1: Creare un pacchetto di distribuzione della funzione Lambda

In questo passaggio, crei un pacchetto di distribuzione della funzione Lambda che contiene il codice della funzione Python e le dipendenze. Carichi questo pacchetto più tardi quando crei la funzione Lambda in AWS Lambda. La funzione Lambda utilizza AWS IoT Greengrass Core SDK per creare e interagire con flussi locali.

Note

Le funzioni Lambda definite dall'utente devono utilizzare [Core SDK per interagire con AWS IoT Greengrass](#) lo stream manager. Per ulteriori informazioni sui requisiti di Stream Manager di Greengrass, consulta l'argomento relativo ai [requisiti di Stream Manager di Greengrass](#).

1. Scarica il [AWS IoT Greengrass Core SDK per Python](#) v1.5.0 o versione successiva.
2. Decomprimere il pacchetto scaricato per ottenere l'SDK. Il kit SDK è la cartella greengrasssdk.

3. Installa le dipendenze dei pacchetti da includere con l'SDK nel pacchetto di distribuzione delle funzioni Lambda.
 1. Passare alla directory SDK che contiene il file `requirements.txt`. Questo file elenca le dipendenze.
 2. Installare le dipendenze SDK. Ad esempio, eseguire il comando `pip` seguente per installarle nella directory corrente:

```
pip install --target . -r requirements.txt
```

4. Salvare la seguente funzione del codice Python nel file locale `transfer_stream.py`.

 Tip

Per esempio di codice che utilizza Java e NodeJS, consulta AWS IoT Greengrass Core SDK [per AWS IoT Greengrass Java](#) e [Core SDK per Node.js](#) su. GitHub

```
import asyncio
import logging
import random
import time

from greengrasssdk.stream_manager import (
    ExportDefinition,
    KinesisConfig,
    MessageStreamDefinition,
    ReadMessagesOptions,
    ResourceNotFoundException,
    StrategyOnFull,
    StreamManagerClient,
)

# This example creates a local stream named "SomeStream".
# It starts writing data into that stream and then stream manager automatically
# exports
# the data to a customer-created Kinesis data stream named "MyKinesisStream".
# This example runs forever until the program is stopped.
```



```
# The size of the local stream on disk will not exceed the default (which is 256 MB).
# Any data appended after the stream reaches the size limit continues to be
# appended, and
# stream manager deletes the oldest data until the total stream size is back under
# 256 MB.
# The Kinesis data stream in the cloud has no such bound, so all the data from this
# script is
# uploaded to Kinesis and you will be charged for that usage.
```

```
def main(logger):
    try:
        stream_name = "SomeStream"
        kinesis_stream_name = "MyKinesisStream"

        # Create a client for the StreamManager
        client = StreamManagerClient()

        # Try deleting the stream (if it exists) so that we have a fresh start
        try:
            client.delete_message_stream(stream_name=stream_name)
        except ResourceNotFoundException:
            pass

        exports = ExportDefinition(
            kinesis=[KinesisConfig(identifier="KinesisExport" + stream_name,
            kinesis_stream_name=kinesis_stream_name)]
        )
        client.create_message_stream(
            MessageStreamDefinition(
                name=stream_name,
            strategy_on_full=StrategyOnFull.OverwriteOldestData, export_definition=exports
            )
        )

        # Append two messages and print their sequence numbers
        logger.info(
            "Successfully appended message to stream with sequence number %d",
            client.append_message(stream_name, "ABCDEFGHJKLMNOP".encode("utf-8")),
        )
        logger.info(
            "Successfully appended message to stream with sequence number %d",
            client.append_message(stream_name, "PQRSTUVWXYZ".encode("utf-8")),
        )
```

```

    )

    # Try reading the two messages we just appended and print them out
    logger.info(
        "Successfully read 2 messages: %s",
        client.read_messages(stream_name,
            ReadMessagesOptions(min_message_count=2, read_timeout_millis=1000)),
    )

    logger.info("Now going to start writing random integers between 0 and 1000
to the stream")
    # Now start putting in random data between 0 and 1000 to emulate device
sensor input
    while True:
        logger.debug("Appending new random integer to stream")
        client.append_message(stream_name, random.randint(0,
1000).to_bytes(length=4, signed=True, byteorder="big"))
        time.sleep(1)

    except asyncio.TimeoutError:
        logger.exception("Timed out while executing")
    except Exception:
        logger.exception("Exception while running")

def function_handler(event, context):
    return

logging.basicConfig(level=logging.INFO)
# Start up this sample code
main(logger=logging.getLogger())

```

5. Comprimere le voci seguenti nel file `transfer_stream_python.zip`. Questo è il tuo pacchetto di implementazione della funzione Lambda.

- `transfer_stream.py`. La logica dell'app.
- `greengrasssdk`. Libreria richiesta per le funzioni Python Greengrass Lambda che pubblicano messaggi MQTT.

Le [operazioni di Stream Manager](#) sono disponibili nella versione 1.5.0 o successiva di AWS IoT Greengrass Core SDK for Python.

- Le dipendenze che hai installato per AWS IoT Greengrass Core SDK for Python (ad esempio, le directory). cbor2

Quando crei il file zip, includi solo questi elementi, non la cartella che li contiene.

Passaggio 2: creazione di una funzione Lambda

1. Crea un ruolo IAM in modo da poter passare il ruolo ARN quando crei la funzione.

JSON Expanded

```
aws iam create-role --role-name Lambda_empty --assume-role-policy '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "lambda.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}'
```

JSON Single-line

```
aws iam create-role --role-name Lambda_empty --assume-role-policy '{"Version":
"2012-10-17", "Statement": [{"Effect": "Allow", "Principal": {"Service":
"lambda.amazonaws.com"},"Action": "sts:AssumeRole"}]}'
```

Note

AWS IoT Greengrass non utilizza questo ruolo perché le autorizzazioni per le funzioni Greengrass Lambda sono specificate nel ruolo del gruppo Greengrass. Per questo tutorial, viene creato un ruolo vuoto.

2. Copia il valore Arn dall'output.

3. Usa l' AWS Lambda API per creare la funzione. TransferStream Il comando seguente presuppone che il file ZIP si trovi nella directory corrente.

- Sostituisci l'*role-arn* con l'Arn copiato.

```
aws lambda create-function \  
--function-name TransferStream \  
--zip-file fileb://transfer_stream_python.zip \  
--role role-arn \  
--handler transfer_stream.function_handler \  
--runtime python3.7
```

4. Pubblicare una versione della funzione.

```
aws lambda publish-version --function-name TransferStream --description 'First  
version'
```

5. Creare un alias della versione pubblicata.

I gruppi Greengrass possono fare riferimento a una funzione Lambda tramite alias (consigliato) o per versione. L'utilizzo di un alias semplifica la gestione degli aggiornamenti del codice perché non è necessario modificare la tabella di sottoscrizione o la definizione del gruppo quando il codice della funzione viene aggiornato. Invece, è sufficiente indirizzare l'alias alla nuova versione della funzione.

```
aws lambda create-alias --function-name TransferStream --name GG_TransferStream --  
function-version 1
```

Note

AWS IoT Greengrass non supporta gli alias Lambda per le versioni \$LATEST.

6. Copia il valore AliasArn dall'output. Si utilizza questo valore quando si configura la funzione per AWS IoT Greengrass

Ora sei pronto per configurare la funzione per AWS IoT Greengrass.

Fase 3: creazione della versione e della definizione della funzione

Questo passaggio crea una versione della definizione della funzione che fa riferimento alla funzione `GGStreamManager` Lambda di sistema e alla funzione Lambda definita dall'utente `TransferStream`. Per abilitare lo stream manager quando si utilizza l' AWS IoT Greengrass API, la versione della definizione della funzione deve includere la funzione. `GGStreamManager`

1. Crea una definizione di funzione con una versione iniziale che contenga il sistema e le funzioni Lambda definite dall'utente.

La seguente versione di definizione abilita lo stream manager con le impostazioni [dei parametri](#) predefinite. Per configurare le impostazioni personalizzate, è necessario definire le variabili di ambiente per i parametri dello stream manager corrispondenti. Per un esempio, vedi [the section called “Abilitare, disabilitare o configurare lo stream manager”](#). AWS IoT Greengrass utilizza le impostazioni predefinite per i parametri omessi. `MemorySize` dovrebbe esserlo almeno `128000`. `Pinned` deve essere impostato su `true`.

Note

Una funzione Lambda di lunga durata (o bloccata) si avvia automaticamente dopo l'avvio e continua AWS IoT Greengrass a funzionare nel proprio contenitore. Ciò è in contrasto con una funzione Lambda su richiesta, che si avvia quando viene richiamata e si interrompe quando non ci sono più attività da eseguire. Per ulteriori informazioni, consulta [the section called “Configurazione del ciclo di vita”](#).

- `arbitrary-function-id` Sostituiscila con un nome per la funzione, ad esempio. `stream-manager`
- Sostituisci `alias-arn` con `AliasArn` quello che hai copiato quando hai creato l'alias per la funzione `LambdaTransferStream`.

JSON expanded

```
aws greengrass create-function-definition --name MyGreengrassFunctions --  
initial-version '{
```

```

"Functions": [
  {
    "Id": "arbitrary-function-id",
    "FunctionArn": "arn:aws:lambda::function:GGStreamManager:1",
    "FunctionConfiguration": {
      "MemorySize": 128000,
      "Pinned": true,
      "Timeout": 3
    }
  },
  {
    "Id": "TransferStreamFunction",
    "FunctionArn": "alias-arn",
    "FunctionConfiguration": {
      "Executable": "transfer_stream.function_handler",
      "MemorySize": 16000,
      "Pinned": true,
      "Timeout": 5
    }
  }
]
}'

```

JSON single

```

aws greengrass create-function-definition \
--name MyGreengrassFunctions \
--initial-version '{"Functions": [{"Id": "arbitrary-function-id", "FunctionArn": "arn:aws:lambda::function:GGStreamManager:1",
  "FunctionConfiguration": {"Environment": {"Variables":
{"STREAM_MANAGER_STORE_ROOT_DIR": "/data", "STREAM_MANAGER_SERVER_PORT":
"1234", "STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH": "20000"}}}, {"Id": "TransferStreamFunction",
  "FunctionArn": "alias-arn", "FunctionConfiguration": {"Executable":
"transfer_stream.function_handler", "MemorySize": 16000, "Pinned":
true, "Timeout": 5}}}]}'

```

Note

Timeout è richiesto dalla versione di definizione della funzione, ma non viene utilizzato da GGStreamManager. Per ulteriori informazioni Timeout e altre impostazioni a

livello di gruppo, consulta [the section called “Controllo dell'esecuzione della funzione Greengrass Lambda”](#)

2. Copia il valore LatestVersionArn dall'output. È possibile utilizzare questo valore per aggiungere la definizione di funzione alla versione del gruppo distribuita nel core.

Fase 4: creazione di una definizione e di una versione di logger

Configurare le impostazioni di registrazione del gruppo. In questo tutorial, configurerai i componenti di AWS IoT Greengrass sistema, le funzioni Lambda definite dall'utente e i connettori per scrivere i log nel file system del dispositivo principale. Puoi utilizzare i log per risolvere eventuali problemi che potrebbero verificarsi. Per ulteriori informazioni, consulta [the section called “Monitoraggio con i log AWS IoT Greengrass”](#).

1. Creare una definizione di logger che includa una versione iniziale.

JSON Expanded

```
aws greengrass create-logger-definition --name "LoggingConfigs" --initial-  
version '{  
  "Loggers": [  
    {  
      "Id": "1",  
      "Component": "GreengrassSystem",  
      "Level": "INFO",  
      "Space": 10240,  
      "Type": "FileSystem"  
    },  
    {  
      "Id": "2",  
      "Component": "Lambda",  
      "Level": "INFO",  
      "Space": 10240,  
      "Type": "FileSystem"  
    }  
  ]  
}'
```

JSON Single-line

```
aws greengrass create-logger-definition \  
  --name "LoggingConfigs" \  
  --initial-version '{"Loggers":  
[{"Id":"1","Component":"GreengrassSystem","Level":"INFO","Space":10240,"Type":"FileSystem"  
{"Id":"2","Component":"Lambda","Level":"INFO","Space":10240,"Type":"FileSystem"}]}'
```

2. Copiare la `LatestVersionArn` della definizione di logger dall'output. Utilizzare questo valore per aggiungere la versione della definizione di logger alla versione del gruppo distribuita nel core.

Fase 5: ottenimento dell'ARN della versione di definizione del core

Ottieni l'ARN della versione di definizione del core da aggiungere alla nuova versione del gruppo. Per distribuire una versione del gruppo, deve fare riferimento a una versione di definizione del core contenente esattamente un core.

1. Ottieni il gruppo Greengrass IDs di destinazione e la versione del gruppo. Questa procedura presuppone che questa sia la versione più recente del gruppo e del gruppo. La seguente query restituisce il gruppo creato più di recente.

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

In alternativa, puoi eseguire query in base al nome. I nomi dei gruppi non devono essere univoci, pertanto potrebbero essere restituiti più gruppi.

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

Note

Puoi trovare questi valori anche nella AWS IoT console. L'ID gruppo viene visualizzato nella pagina Settings (Impostazioni) del gruppo. IDs Le versioni del gruppo vengono visualizzate nella scheda Distribuzioni del gruppo.

2. Copiare l'Id del gruppo di destinazione dall'output. Questo valore viene utilizzato per ottenere la versione della definizione del core e durante la distribuzione del gruppo.

3. Copiare la `LatestVersion` dall'output, che corrisponde all'ID dell'ultima versione aggiunta al gruppo. Questo valore viene utilizzato per ottenere la versione della definizione del core.
4. Per ottenere l'ARN della versione di definizione del core:
 - a. Ottenere la versione del gruppo.
 - `group-id` Sostituiscila con quella Id che hai copiato per il gruppo.
 - Sostituisci `group-version-id` con `LatestVersion` quello che hai copiato per il gruppo.

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

- b. Copia il valore `CoreDefinitionVersionArn` dall'output. Utilizza questo valore per aggiungere la versione di definizione del core alla versione del gruppo distribuita nel core.

Fase 6: creazione di una versione del gruppo

È ora possibile creare una versione del gruppo che contenga tutti le entità da distribuire. A questo scopo, è necessario creare una versione di gruppo che faccia riferimento alla versione di destinazione di ciascun tipo di componente. Per questo tutorial, includere una versione di definizione del core, una versione di definizione della funzione e una versione di definizione di logger.

1. Creare una versione del gruppo.
 - Sostituisci `group-id` con Id quello che hai copiato per il gruppo.
 - Sostituiscilo `core-definition-version-arn` con `CoreDefinitionVersionArn` quello che hai copiato per la versione di definizione principale.
 - Sostituiscilo `function-definition-version-arn` con `LatestVersionArn` quello che hai copiato per la nuova versione della definizione di funzione.
 - `logger-definition-version-arn` Sostituiscilo con `LatestVersionArn` quello che hai copiato per la nuova versione della definizione del logger.

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

```
--core-definition-version-arn core-definition-version-arn \  
--function-definition-version-arn function-definition-version-arn \  
--logger-definition-version-arn logger-definition-version-arn
```

2. Copia il valore Version dall'output. Questo è l'ID della nuova versione gruppo.

Fase 7: creazione di una distribuzione

Distribuire il gruppo al nuovo dispositivo core.

1. Assicurati che il AWS IoT Greengrass core sia in funzione. Esegui i seguenti comandi nel terminale di Raspberry Pi in base alle esigenze.
 - a. Per controllare se il daemon è in esecuzione:

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

Se l'output contiene una voce root per `/greengrass/ggc/packages/ggc-version/bin/daemon`, allora il daemon è in esecuzione.

Note

La versione indicata nel percorso dipende dalla versione del software AWS IoT Greengrass Core installata sul dispositivo principale.

- b. Per avviare il demone:

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

2. Crea una implementazione .

- *group-id* Sostituiscilo con Id quello che hai copiato per il gruppo.
- Sostituisci *group-version-id* con Version quello che hai copiato per la nuova versione del gruppo.

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

```

```
--group-version-id group-version-id
```

3. Copia il valore DeploymentId dall'output.
4. Ottenere lo stato della distribuzione.
 - Sostituisci *group-id* con Id quello che hai copiato per il gruppo.
 - Sostituiscilo *deployment-id* con DeploymentId quello che hai copiato per la distribuzione.

```
aws greengrass get-deployment-status \  
--group-id group-id \  
--deployment-id deployment-id
```

Se lo stato èSuccess, la distribuzione è avvenuta con successo. Per la risoluzione dei problemi, consultare [Risoluzione dei problemi](#).

Fase 8: test dell'applicazione

La funzione TransferStream Lambda genera dati simulati del dispositivo. Scrive i dati in un flusso che viene esportato da stream manager nel flusso di dati Kinesis target.

1. Nella console Amazon Kinesis, in Kinesis data stream, scegli. MyKinesisStream

Note

Se il tutorial è stato eseguito senza un flusso di dati Kinesis di destinazione, [controllare il file di log](#) per stream manager (GGStreamManager). Se contiene `export stream MyKinesisStream doesn't exist` in un messaggio di errore, il test viene superato. Questo errore indica che il servizio ha tentato di esportare nel flusso ma il flusso non esiste.

2. Nella MyKinesisStreampagina, scegli Monitoraggio. Se il test vien superato, i dati sono visibili nei grafici Put Records (Inserisci record) . A seconda della connessione, potrebbe essere necessario fino a un minuto prima che i dati vengano visualizzati.

 Important

Al termine della sessione di test, eliminare il flusso di dati Kinesis per evitare di incorrere in ulteriori costi.

In alternativa, eseguire i comandi seguenti per arrestare il daemon Greengrass. Ciò impedisce al core di inviare messaggi fino a quando non si è pronti a continuare il test.

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

3. Rimuovi la funzione TransferStreamLambda dal core.
 - a. Seguire [the section called “Creazione di una versione del gruppo”](#) per creare una nuova versione del gruppo, ma rimuovere l'opzione `--function-definition-version-arn` nel comando `create-group-version`. Oppure, crea una versione della definizione di funzione che non includa la funzione TransferStreamLambda.

 Note

Omettendo la funzione GGStreamManager Lambda del sistema dalla versione di gruppo distribuita, si disabilita la gestione del flusso sul core.

- b. Seguire [the section called “Crea distribuzione”](#) per distribuire la nuova versione del gruppo.

Per visualizzare le informazioni di registrazione o risolvere problemi con i flussi, controllare i log per le funzioni TransferStream e GGStreamManager. È necessario disporre root delle autorizzazioni per leggere i AWS IoT Greengrass log sul file system.

- TransferStream scrive le voci di log in `greengrass-root/ggc/var/log/user/region/account-id/TransferStream.log`.
- GGStreamManager scrive le voci di log in `greengrass-root/ggc/var/log/system/GGStreamManager.log`.

Se sono necessarie ulteriori informazioni sulla risoluzione dei problemi, puoi impostare il livello di registrazione Lambda su DEBUG e quindi creare e distribuire una nuova versione del gruppo.

Consulta anche

- [Gestione dei flussi di dati](#)
- [the section called “Utilizzalo StreamManagerClient per lavorare con gli stream”](#)
- [the section called “Esporta le configurazioni per le destinazioni supportate Cloud AWS”](#)
- [the section called “Configurazione di Stream Manager”](#)
- [the section called “Esportazione di flussi di dati \(Console\)”](#)
- [AWS Identity and Access Management Comandi \(IAM\) nel Command AWS CLI Reference](#)
- [AWS Lambda comandi nel AWS CLI Command Reference](#)
- [AWS IoT Greengrass comandi nel AWS CLI Command Reference](#)

Implementa i segreti fino al nucleo AWS IoT Greengrass

Questa funzionalità è disponibile per AWS IoT Greengrass Core v1.7 e versioni successive.

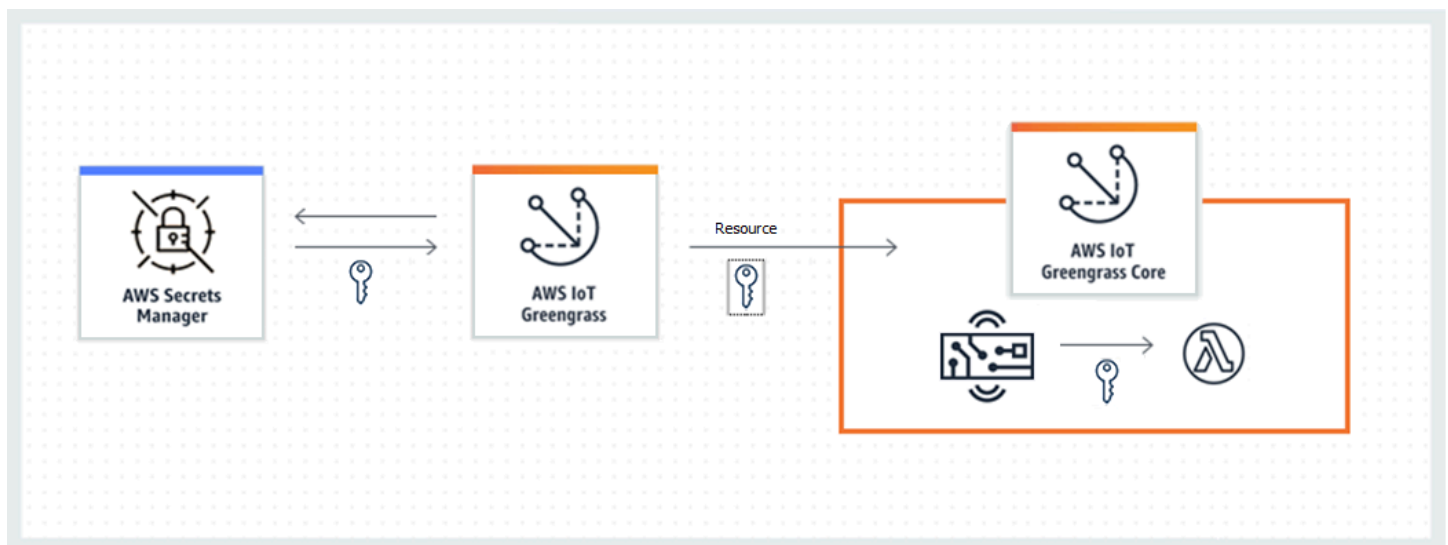
AWS IoT Greengrass ti consente di autenticarti con servizi e applicazioni dei dispositivi Greengrass senza password, token o altri segreti codificati.

AWS Secrets Manager è un servizio che puoi utilizzare per archiviare e gestire in modo sicuro i tuoi segreti nel cloud. AWS IoT Greengrass estende Secrets Manager ai dispositivi principali di Greengrass, in modo che i [connettori](#) e le funzioni Lambda possano utilizzare i segreti locali per interagire con servizi e applicazioni. Ad esempio, il connettore notifiche Twilio utilizza un token di autenticazione memorizzato localmente.

Per integrare un segreto in un gruppo Greengrass, è necessario creare una risorsa di gruppo che faccia riferimento al segreto Secrets Manager. Questa risorsa segreta fa riferimento al segreto del cloud tramite ARN. Per ulteriori informazioni su come creare, gestire e utilizzare le risorse segrete, consulta [the section called “Utilizzo delle risorse segrete”](#).

AWS IoT Greengrass crittografa i tuoi segreti mentre sono in transito e a riposo. Durante l'implementazione di gruppo, AWS IoT Greengrass recupera il segreto da Secrets Manager e crea una copia locale crittografata sul core di Greengrass. Dopo aver ruotato i segreti del cloud in Secrets Manager, ridistribuisce il gruppo per propagare i valori aggiornati al core.

Lo schema seguente mostra il processo di alto livello di distribuzione di un segreto al core. I segreti vengono crittografati mentre sono in transito e quando sono inattivi.



L'utilizzo AWS IoT Greengrass per archiviare i segreti a livello locale offre i seguenti vantaggi:

- Disassociazione dal codice (non hardcoded). Supporta le credenziali gestite a livello centralizzato e consente di proteggere i dati sensibili dal rischio di compromissione.
- Disponibile per gli scenari offline. I connettori e le funzioni sono in grado di accedere in modo sicuro al software e ai servizi locali anche quando non sono connessi a Internet.
- Accesso controllato ai segreti. Solo le funzioni e i connettori autorizzati del gruppo possono accedere ai tuoi segreti. AWS IoT Greengrass utilizza la crittografia con chiave privata per proteggere i segreti. I segreti vengono crittografati mentre sono in transito e quando sono inattivi. Per ulteriori informazioni, consulta [the section called "Crittografia dei segreti"](#).
- Rotazione controllata. Dopo aver ruotato i segreti in Secrets Manager, ridistribuisci il gruppo Greengrass per aggiornare le copie locali dei tuoi segreti. Per ulteriori informazioni, consulta [the section called "Creazione e gestione di segreti"](#).

 Important

AWS IoT Greengrass non aggiorna automaticamente i valori dei segreti locali dopo la rotazione delle versioni cloud. Per aggiornare i valori locali, è necessario distribuire di nuovo il gruppo.

Crittografia dei segreti

AWS IoT Greengrass crittografa i segreti in transito e a riposo.

 Important

Assicurati che le funzioni Lambda definite dall'utente gestiscano i segreti in modo sicuro e non registrino alcun dato sensibile archiviato nel segreto. Per ulteriori informazioni, consulta [Mitigare i rischi di registrazione e debug della funzione Lambda nella Guida per l'utente](#). AWS Secrets Manager. Sebbene questa documentazione si riferisca specificamente alle funzioni di rotazione, la raccomandazione si applica anche alle funzioni Greengrass Lambda.

Crittografia in transito

AWS IoT Greengrass utilizza Transport Layer Security (TLS) per crittografare tutte le comunicazioni su Internet e sulla rete locale. Ciò protegge i segreti durante il transito, il che

si verifica quando i segreti vengono recuperati da Secrets Manager e distribuiti al core. Per informazioni sulle suite di cifratura TLS supportate, consulta [the section called “Supporto TLS per le suite di cifratura”](#).

Crittografia dei dati inattivi

AWS IoT Greengrass utilizza la chiave privata specificata in [config.json](#) per la crittografia dei segreti archiviati nel core. Pertanto, per la protezione dei segreti locali è fondamentale l'archiviazione sicura della chiave privata. Nel [modello di responsabilità AWS condivisa](#), è responsabilità del cliente garantire l'archiviazione sicura della chiave privata sul dispositivo principale.

AWS IoT Greengrass supporta due modalità di archiviazione delle chiavi private:

- Utilizzo dei moduli di sicurezza hardware Per ulteriori informazioni, consulta [the section called “Integrazione della sicurezza hardware”](#).

Note

Attualmente, AWS IoT Greengrass supporta solo il meccanismo di riempimento [PKCS #1 v1.5](#) per la crittografia e la decrittografia dei segreti locali quando si utilizzano chiavi private basate su hardware. Se stai seguendo le istruzioni fornite dal fornitore per generare manualmente chiavi private basate su hardware, assicurati di scegliere PKCS #1 v1.5. AWS IoT Greengrass non supporta Optimal Asymmetric Encryption Padding (OAEP).

- Utilizzo delle autorizzazioni del file system (impostazione predefinita).

La chiave privata viene utilizzata per proteggere la chiave di dati, utilizzate per crittografare i segreti locali. La chiave di dati viene ruotata con ciascuna distribuzione del gruppo.

Il AWS IoT Greengrass core è l'unica entità che ha accesso alla chiave privata. I connettori Greengrass o le funzioni Lambda che sono affiliati a una risorsa segreta ottengono il valore del segreto dal core.

Requisiti

Di seguito sono indicati i requisiti per il supporto dei segreti locali:

- È necessario utilizzare AWS IoT Greengrass Core v1.7 o versione successiva.

- Per ottenere i valori dei segreti locali, le funzioni Lambda definite dall'utente devono AWS IoT Greengrass utilizzare Core SDK v1.3.0 o versione successiva.
- La chiave privata utilizzata per la crittografia dei segreti locali deve essere specificata nel file di configurazione di Greengrass. Per impostazione predefinita, AWS IoT Greengrass utilizza la chiave privata principale memorizzata nel file system. Per fornire una chiave privata personalizzata, consulta [the section called “Specificare la chiave privata per la crittografia dei segreti”](#). Solo il tipo di chiave RSA è supportato.

Note

Attualmente, AWS IoT Greengrass supporta solo il meccanismo di riempimento [PKCS #1 v1.5](#) per la crittografia e la decrittografia dei segreti locali quando si utilizzano chiavi private basate su hardware. Se stai seguendo le istruzioni fornite dal fornitore per generare manualmente chiavi private basate su hardware, assicurati di scegliere PKCS #1 v1.5. AWS IoT Greengrass non supporta Optimal Asymmetric Encryption Padding (OAEP).

- AWS IoT Greengrass devi avere l'autorizzazione per ottenere i tuoi valori segreti. Ciò consente di AWS IoT Greengrass recuperare i valori durante la distribuzione di gruppo. Se utilizzi il ruolo di servizio Greengrass predefinito, hai AWS IoT Greengrass già accesso ai segreti con nomi che iniziano con greengrass -. Per personalizzare l'accesso, consulta [the section called “AWS IoT Greengrass Consenti di ottenere valori segreti”](#).

Note

Ti consigliamo di utilizzare questa convenzione di denominazione per identificare i segreti a cui AWS IoT Greengrass è consentito accedere, anche se personalizzi le autorizzazioni. La console utilizza diverse autorizzazioni per leggere i tuoi segreti, quindi è possibile che tu possa selezionare i segreti nella console che AWS IoT Greengrass non sono autorizzati a recuperare. L'utilizzo di una convenzione di denominazione può aiutarti a evitare un conflitto di autorizzazioni, che potrebbe causare un errore di distribuzione.

Specificare la chiave privata per la crittografia dei segreti

In questa procedura, specifichi il percorso di una chiave privata utilizzata per la crittografia segreta locale. Deve essere una chiave RSA con una lunghezza minima di 2048 bit. Per ulteriori informazioni

sulle chiavi private utilizzate nel AWS IoT Greengrass core, consulta [the section called “Principal di sicurezza”](#)

AWS IoT Greengrass supporta due modalità di archiviazione delle chiavi private: basata su hardware o basata su file system (impostazione predefinita). Per ulteriori informazioni, consulta [the section called “Crittografia dei segreti”](#).

Segui questa procedura solo se desideri modificare la configurazione predefinita, che utilizza la chiave privata del core nel file system. Queste fasi presuppongono che il gruppo e il core siano stati creati come descritto nel [Modulo 2](#) del tutorial Nozioni di base.

1. Apri il file [config.json](#) disponibile nella directory `/greengrass-root/config`.

 Note

greengrass-root rappresenta il percorso in cui il software AWS IoT Greengrass Core è installato sul dispositivo. In genere, questa è la directory `/greengrass`.

2. Nell'oggetto `crypto.principals.SecretsManager`, per la proprietà `privateKeyPath`, immettere il percorso della chiave privata:
 - Se la chiave privata viene archiviata nel file system, specifica il percorso assoluto della chiave. Per esempio:

```
"SecretsManager" : {  
  "privateKeyPath" : "file:///somepath/hash.private.key"  
}
```

- Se la chiave privata è archiviata in un modulo di sicurezza hardware (HSM), specifica il percorso utilizzando lo schema URI [RFC 7512 PKCS#11](#). Per esempio:

```
"SecretsManager" : {  
  "privateKeyPath" : "pkcs11:object=private-key-label;type=private"  
}
```

Per ulteriori informazioni, consulta [the section called “Configurazione della sicurezza hardware”](#).

 Note

Attualmente, AWS IoT Greengrass supporta solo il meccanismo di riempimento [PKCS #1 v1.5](#) per la crittografia e la decrittografia dei segreti locali quando si utilizzano chiavi private basate su hardware. Se stai seguendo le istruzioni fornite dal fornitore per generare manualmente chiavi private basate su hardware, assicurati di scegliere PKCS #1 v1.5. AWS IoT Greengrass non supporta Optimal Asymmetric Encryption Padding (OAEP).

AWS IoT Greengrass Consenti di ottenere valori segreti

In questa procedura, aggiungi una politica in linea al ruolo del servizio Greengrass che AWS IoT Greengrass consente di ottenere i valori dei tuoi segreti.

Segui questa procedura solo se desideri concedere autorizzazioni AWS IoT Greengrass personalizzate ai tuoi segreti o se il tuo ruolo di servizio Greengrass non include `AWSGreengrassResourceAccessRolePolicy` la policy gestita. `AWSGreengrassResourceAccessRolePolicy` concede l'accesso ai segreti con nomi che iniziano con `greengrass-`.

1. Eseguire i seguenti comandi della CLI per ottenere l'ARN del ruolo del servizio Greengrass:

```
aws greengrass get-service-role-for-account --region region
```

L'ARN restituito include il nome del ruolo.

```
{
  "AssociatedAt": "time-stamp",
  "RoleArn": "arn:aws:iam::account-id:role/service-role/role-name"
}
```

Puoi utilizzare l'ARN o il nome nella fase seguente.

2. Aggiungere una policy inline che autorizza l'azione `secretsmanager:GetSecretValue`. Per istruzioni, consulta [Aggiungere e rimuovere le politiche IAM nella Guida](#) per l'utente IAM.

È possibile concedere l'accesso granulare elencando esplicitamente i segreti oppure utilizzando uno schema di denominazione con il carattere jolly *. In alternativa, è possibile concedere l'accesso condizionale ai segreti con versioni o tag. Ad esempio, la seguente politica consente di AWS IoT Greengrass leggere solo i segreti specificati.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Resource": [
        "arn:aws:secretsmanager:region:account-id:secret:greenrass-SecretA-abc",
        "arn:aws:secretsmanager:region:account-id:secret:greenrass-SecretB-xyz"
      ]
    }
  ]
}
```

Note

Se utilizzi una AWS KMS chiave gestita dal cliente per crittografare i segreti, anche il tuo ruolo di servizio Greengrass deve consentire l'azione. `kms:Decrypt`

Per ulteriori informazioni sulle policy IAM per Secrets Manager, consulta [Authentication AWS Secrets Manager and access control for](#) and [Actions, resources and context keys che puoi utilizzare in una policy IAM o in una policy segreta AWS Secrets Manager nella Guida per l'AWS Secrets Manager utente](#).

Consulta anche

- [Che cos'è AWS Secrets Manager?](#) nella Guida per l'AWS Secrets Manager utente
- [PKCS #1: RSA Encryption versione 1.5](#)

Utilizzo delle risorse segrete

AWS IoT Greengrass utilizza risorse segrete per integrare i segreti AWS Secrets Manager in un gruppo Greengrass. Una risorsa segreta è un riferimento a un segreto di Secrets Manager. Per ulteriori informazioni, consulta [Distribuzione dei segreti nel core](#).

Sul dispositivo AWS IoT Greengrass principale, i connettori e le funzioni Lambda possono utilizzare la risorsa segreta per autenticarsi con servizi e applicazioni, senza password, token o altre credenziali di codifica.

Creazione e gestione di segreti

In un gruppo Greengrass, una risorsa segreta fa riferimento all'ARN di un segreto di Secrets Manager. Quando la risorsa segreta viene distribuita nel core, il valore del segreto viene crittografato e reso disponibile ai connettori affiliati e alle funzioni Lambda. Per ulteriori informazioni, consulta [the section called "Crittografia dei segreti"](#).

Utilizzi Secrets Manager per creare e gestire le versioni cloud dei tuoi segreti. Lo usi AWS IoT Greengrass per creare, gestire e distribuire le tue risorse segrete.

Important

Ti consigliamo di seguire la migliore pratica di ruotare i tuoi segreti in Secrets Manager. Quindi, distribuirai il gruppo Greengrass per aggiornare le copie locali dei segreti. Per ulteriori informazioni, consulta [Ruotare AWS Secrets Manager i propri segreti nella Guida](#) per l'AWS Secrets Manager utente.

Per rendere un segreto disponibile nel core Greengrass

1. Creazione di un segreto in Secrets Manager. Questa è la versione cloud del tuo segreto, archiviato e gestito centralmente in Secrets Manager. Le attività di gestione includono la rotazione dei valori dei segreti e l'applicazione di policy delle risorse.
2. Crea una risorsa segreta in AWS IoT Greengrass. Si tratta di un tipo di risorsa di gruppo che fa riferimento al segreto nel cloud di ARN. È possibile fare riferimento a un segreto solo una volta per ogni gruppo.
3. Configura il connettore o la funzione Lambda. È necessario affiliare la risorsa a un connettore o a una funzione specificando i parametri corrispondenti o le proprietà. In questo modo otterranno

il valore della risorsa segreta distribuita in locale. Per ulteriori informazioni, consulta [the section called “Utilizzo dei segreti locali”](#).

4. Distribuire il gruppo Greengrass. Durante la distribuzione, AWS IoT Greengrass recupera il valore del segreto nel cloud e crea (o aggiorna) il segreto locale nel core.

Secrets Manager registra un evento AWS CloudTrail ogni volta che AWS IoT Greengrass recupera un valore segreto. AWS IoT Greengrass non registra alcun evento relativo alla distribuzione o all'utilizzo di segreti locali. Per ulteriori informazioni sulla registrazione di Secrets Manager, consulta [Monitora l'uso dei tuoi AWS Secrets Manager segreti](#) nella Guida per l'AWS Secrets Manager utente.

Inclusione delle etichette temporanee nelle risorse segrete

Secrets Manager utilizza le etichette temporanee per identificare versioni specifiche di un valore segreto. Le etichette di gestione temporanea possono essere definite dal sistema o dall'utente. Secrets Manager assegna l'AWSCURRENT etichetta alla versione più recente del valore segreto. Le etichette di gestione temporanea vengono in genere utilizzate per gestire la rotazione dei segreti. Per ulteriori informazioni sul controllo delle versioni di Secrets Manager, vedere [Termini e concetti chiave AWS Secrets Manager nella Guida per l'AWS Secrets Manager utente](#).

Le risorse segrete includono sempre l'etichetta AWSCURRENT di staging e, facoltativamente, possono includere altre etichette di staging se richieste da una funzione o da un connettore Lambda. Durante la distribuzione di gruppo, AWS IoT Greengrass recupera i valori delle etichette di gestione temporanea a cui si fa riferimento nel gruppo, quindi crea o aggiorna i valori corrispondenti nel core.

Creazione e gestione delle risorse segrete (console)

Creazione delle risorse segrete (console)

Nella AWS IoT Greengrass console, puoi creare e gestire risorse segrete dalla scheda Segreti della pagina Risorse del gruppo. Per i tutorial che illustrano la creazione di una risorsa segreta e la relativa aggiunta a un gruppo, consulta [the section called “Come creare una risorsa segreta \(console\)”](#) e [the section called “Nozioni di base sui connettori \(console\)”](#).

Deployments	Resources			
Subscriptions				
Cores	Local	Machine Learning	Secret	
Devices				
Lambdas				
Resources	Resource Name ▾	Secret Name	Status	Labels
Connectors	MyTwilioAuthToken	greengrass-TwilioAuthTo...	● Unaffiliated	AWSCURRENT ...
Tags				
Settings				

[Add secret resource](#)

Note

In alternativa, la console consente di creare una risorsa segreta e segreta quando si configura un connettore o una funzione Lambda. È possibile eseguire questa operazione dalla pagina Configura parametri del connettore o dalla pagina Risorse della funzione Lambda.

Gestione delle risorse segrete (console)

Le attività di gestione delle risorse segrete del tuo gruppo Greengrass includono l'aggiunta di risorse segrete al gruppo, la rimozione di risorse segrete dal gruppo e la modifica del set di [etichette di gestione temporanea](#) incluse in una risorsa segreta.

Se indichi un segreto diverso da Secrets Manager, devi anche modificare tutti i connettori che utilizzano il segreto:

1. Nella pagina di configurazione del gruppo, scegliere Connettori.
2. Dal menu contestuale del connettore, selezionare Modifica.
3. Nella pagina Edit parameters (Modifica parametri) viene visualizzato un messaggio che informa che l'ARN del segreto è stato modificato. Per confermare la modifica, selezionare Save (Salva).

Se elimini un segreto in Secrets Manager, rimuovi la risorsa segreta corrispondente dal gruppo e dai connettori e dalle funzioni Lambda che vi fanno riferimento. Altrimenti, durante la distribuzione di gruppo, AWS IoT Greengrass restituisce un errore indicante che il segreto non può essere trovato. Aggiorna anche il codice della funzione Lambda secondo necessità.

Creazione e gestione delle risorse segrete (CLI)

Creazione delle risorse segrete (CLI)

Nell' AWS IoT Greengrass API, un segreto è un tipo di risorsa di gruppo. L'esempio seguente crea una definizione di risorsa con una versione iniziale che include la risorsa segreta `MySecretResource`. Per un tutorial che illustra la creazione di una risorsa segreta e la relativa aggiunta a un versione del gruppo, consulta [the section called “Nozioni di base sui connettori \(CLI\)”](#).

La risorsa segreta fa riferimento all'ARN del segreto corrispondente di Secrets Manager e include due etichette di staging in aggiunta a `AWSCURRENT`, che è sempre inclusa.

```
aws greengrass create-resource-definition --name MyGreengrassResources --initial-  
version '{  
  "Resources": [  
    {  
      "Id": "my-resource-id",  
      "Name": "MySecretResource",  
      "ResourceDataContainer": {  
        "SecretsManagerSecretResourceData": {  
          "ARN": "arn:aws:secretsmanager:us-  
west-2:123456789012:secret:greenrass-SomeSecret-KUj89s",  
          "AdditionalStagingLabelsToDownload": [  
            "Label1",  
            "Label2"  
          ]  
        }  
      }  
    ]  
  }  
}'
```

Gestione delle risorse segrete (CLI)

Le attività di gestione delle risorse segrete del tuo gruppo Greengrass includono l'aggiunta di risorse segrete al gruppo, la rimozione di risorse segrete dal gruppo e la modifica del set di [etichette di gestione temporanea](#) incluse in una risorsa segreta.

Nell' AWS IoT Greengrass API, queste modifiche vengono implementate utilizzando le versioni.

L' AWS IoT Greengrass API utilizza versioni per gestire i gruppi. Le versioni sono immutabili, quindi per aggiungere o modificare i componenti del gruppo, ad esempio i dispositivi client, le funzioni e le

risorse del gruppo, è necessario creare versioni di componenti nuovi o aggiornati. Quindi, si crea e si distribuisce una versione di gruppo che contiene la versione di destinazione di ciascun componente. Per ulteriori informazioni sui gruppi, consultate [the section called “AWS IoT Greengrass gruppi”](#).

Ad esempio, per modificare la serie di etichette di gestione temporanea di una risorsa segreta:

1. Creare una versione della definizione di risorsa che contenga la risorsa segreta aggiornata. L'esempio seguente aggiunge una terza etichetta di gestione temporanea alla risorsa segreta della sezione precedente.

Note

Per aggiungere altre risorse alla versione, includerle nella matrice `Resources`.

```
aws greengrass create-resource-definition --name MyGreengrassResources --initial-
version '{
  "Resources": [
    {
      "Id": "my-resource-id",
      "Name": "MySecretResource",
      "ResourceDataContainer": {
        "SecretsManagerSecretResourceData": {
          "ARN": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:greengrass-SomeSecret-KUj89s",
          "AdditionalStagingLabelsToDownload": [
            "Label1",
            "Label2",
            "Label3"
          ]
        }
      }
    }
  ]
}'
```

2. Se l'ID della risorsa segreta viene modificato, aggiornare i connettori e le funzioni che utilizzano la risorsa segreta. Nelle nuove versioni, aggiornare il parametro o la proprietà corrispondente all'ID della risorsa. Se l'ARN del segreto viene modificato, è necessario aggiornare anche il parametro corrispondente per qualsiasi connettore che utilizza il segreto.

 Note

L'ID della risorsa è un identificativo arbitrario fornito dal cliente.

3. Creare una versione del gruppo contenente la versione di destinazione di ciascun componente da inviare al core.
4. Distribuire la versione del gruppo.

Per un tutorial che mostra come creare e distribuire le risorse segrete, i connettori e le funzioni, consulta [the section called “Nozioni di base sui connettori \(CLI\)”](#).

Se elimini un segreto in Secrets Manager, rimuovi la risorsa segreta corrispondente dal gruppo e dai connettori e dalle funzioni Lambda che vi fanno riferimento. Altrimenti, durante la distribuzione di gruppo, AWS IoT Greengrass restituisce un errore indicante che il segreto non può essere trovato. Aggiorna anche il codice della funzione Lambda secondo necessità. Puoi rimuovere un segreto locale distribuendo una versione di definizione delle risorse che non contenga la risorsa segreta corrispondente.

Utilizzo dei segreti locali nei connettori e nelle funzioni Lambda

I connettori Greengrass e le funzioni Lambda utilizzano segreti locali per interagire con servizi e applicazioni. Il valore `AWSCURRENT` viene utilizzato per impostazione predefinita, ma sono anche disponibili i valori delle altre [etichette temporanee](#) incluse nella risorsa segreta.

Per consentirne l'accesso ai segreti locali, è necessario configurare i connettori e le funzioni. La risorsa viene così affiliata al connettore o alla funzione.

Connectors (Connettori)

Se un connettore richiede l'accesso a un segreto locale, fornisce i parametri da configurare con le informazioni necessarie per accedere al segreto.

- Per informazioni su come eseguire questa operazione nella AWS IoT Greengrass console, consulta [the section called “Nozioni di base sui connettori \(console\)”](#)
- Per informazioni su come eseguire questa operazione con la AWS IoT Greengrass CLI, consulta [the section called “Nozioni di base sui connettori \(CLI\)”](#)

Per informazioni sui requisiti dei singoli connettori, consulta [the section called “AWS-connettori Greengrass forniti”](#).

La logica per accedere e utilizzare il segreto è integrata nel connettore.

Funzioni Lambda

Per consentire a una funzione Greengrass Lambda di accedere a un segreto locale, è necessario configurare le proprietà della funzione.

- Per informazioni su come eseguire questa operazione nella AWS IoT Greengrass console, consulta [the section called “Come creare una risorsa segreta \(console\)”](#)
- A tale scopo nell' AWS IoT Greengrass API, fornisci le seguenti informazioni nella `ResourceAccessPolicies` proprietà.
 - `ResourceId`: l'ID della risorsa segreta nel gruppo Greengrass. Questa è la risorsa che fa riferimento all'ARN del segreto Secrets Manager corrispondente.
 - `Permission`: il tipo di accesso della funzione rispetto alla risorsa. Per le risorse segrete è supportata solo l'autorizzazione `ro` (sola lettura).

L'esempio seguente crea una funzione Lambda che può accedere alla risorsa `MyApiKey` segreta.

```
aws greengrass create-function-definition --name MyGreengrassFunctions --initial-  
version '{  
  "Functions": [  
    {  
      "Id": "MyLambdaFunction",  
      "FunctionArn": "arn:aws:lambda:us-  
west-2:123456789012:function:myFunction:1",  
      "FunctionConfiguration": {  
        "Pinned": false,  
        "MemorySize": 16384,  
        "Timeout": 10,  
        "Environment": {  
          "ResourceAccessPolicies": [  
            {  
              "ResourceId": "MyApiKey",  
              "Permission": "ro"  
            }  
          ],  
          "AccessSysfs": true  
        }  
      }  
    }  
  ]  
}
```

```
}'
```

Per accedere ai segreti locali in fase di esecuzione, le funzioni di Greengrass Lambda richiamano la `get_secret_value` funzione dal `secretsmanager` client nel AWS IoT Greengrass Core SDK (v1.3.0 o successivo).

L'esempio seguente mostra come usare AWS IoT Greengrass Core SDK per Python per ottenere un segreto. Passa il nome del segreto alla `get_secret_value` funzione. `SecretId` può essere il nome o l'ARN del segreto di Secrets Manager (non la risorsa segreta).

```
import greengrasssdk

secrets_client = greengrasssdk.client("secretsmanager")
secret_name = "greengrass-MySecret-abc"

def function_handler(event, context):
    response = secrets_client.get_secret_value(SecretId=secret_name)
    secret = response.get("SecretString")
```

Per i segreti di tipo testo, la funzione `get_secret_value` restituisce una stringa. Per i segreti di tipo binario, restituisce una stringa con codifica base64.

Important

Assicurati che le funzioni Lambda definite dall'utente gestiscano i segreti in modo sicuro e non registrino alcun dato sensibile archiviato nel segreto. Per ulteriori informazioni, consulta [Mitigare i rischi di registrazione e debug della funzione Lambda nella Guida per l'utente](#). AWS Secrets Manager. Sebbene questa documentazione si riferisca specificamente alle funzioni di rotazione, la raccomandazione si applica anche alle funzioni Greengrass Lambda.

Il valore corrente del segreto viene restituito per impostazione predefinita. Si tratta della versione a cui è collegata l'etichetta temporanea `AWSCURRENT`. Per accedere a una versione

diversa, passa il nome dell'etichetta temporanea corrispondente per l'argomento facoltativo `VersionStage`. Per esempio:

```
import greengrasssdk

secrets_client = greengrasssdk.client("secretsmanager")
secret_name = "greengrass-TestSecret"
secret_version = "MyTargetLabel"

# Get the value of a specific secret version
def function_handler(event, context):
    response = secrets_client.get_secret_value(
        SecretId=secret_name, VersionStage=secret_version
    )
    secret = response.get("SecretString")
```

Per un'altra funzione di esempio che chiama `get_secret_value`, consulta [Creare un pacchetto di distribuzione della funzione Lambda](#).

Come creare una risorsa segreta (console)

Questa funzionalità è disponibile per AWS IoT Greengrass Core v1.7 e versioni successive.

Questo tutorial mostra come usare per AWS Management Console aggiungere una risorsa segreta a un gruppo Greengrass. Una risorsa segreta è un riferimento a una forma segreta. AWS Secrets Manager Per ulteriori informazioni, consulta [Distribuzione dei segreti nel core](#).

Sul dispositivo AWS IoT Greengrass principale, i connettori e le funzioni Lambda possono utilizzare la risorsa segreta per autenticarsi con servizi e applicazioni, senza password, token o altre credenziali di codifica.

In questo tutorial, inizierai creando un segreto nella console. AWS Secrets Manager Quindi, nella AWS IoT Greengrass console, aggiungi una risorsa segreta a un gruppo Greengrass dalla pagina Risorse del gruppo. Questa risorsa segreta fa riferimento al segreto di Secrets Manager. Successivamente, si collega la risorsa segreta a una funzione Lambda, che consente alla funzione di ottenere il valore del segreto locale.

Note

In alternativa, la console consente di creare una risorsa segreta e segreta quando si configura un connettore o una funzione Lambda. È possibile eseguire questa operazione dalla pagina Configura parametri del connettore o dalla pagina Risorse della funzione Lambda.

Solo i connettori contenenti parametri per i segreti possono accedere ai segreti. Per un tutorial che mostra come il connettore Twilio Notifications utilizza un token di autenticazione memorizzato localmente, consulta [the section called “Nozioni di base sui connettori \(console\)”](#)

Il tutorial include le seguenti fasi di alto livello:

1. [Crea un segreto di Secrets Manager](#)
2. [Aggiunta di una risorsa segreta a un gruppo](#)
3. [Creare un pacchetto di distribuzione della funzione Lambda](#)
4. [Creazione di una funzione Lambda](#)
5. [Aggiunta della funzione al gruppo](#)
6. [Collegamento della risorsa segreta alla funzione](#)
7. [Aggiunta di sottoscrizioni al gruppo](#)
8. [Distribuzione del gruppo.](#)
9. [the section called “Test della funzione Lambda”](#)

Il completamento di questo tutorial richiede circa 20 minuti.

Prerequisiti

Per completare questo tutorial, è necessario quanto segue:

- Un gruppo Greengrass e un core Greengrass (v1.7 o successivo). Per informazioni su come creare un gruppo e un core Greengrass, consulta [Iniziare con AWS IoT Greengrass](#). Il tutorial Getting Started include anche i passaggi per l'installazione del AWS IoT Greengrass software Core.
- AWS IoT Greengrass deve essere configurato per supportare i segreti locali. Per ulteriori informazioni, consulta [Requisiti dei segreti](#).

 Note

Questo requisito include l'autorizzazione all'accesso ai segreti di Secrets Manager. Se utilizzi il ruolo di servizio Greengrass predefinito, Greengrass è autorizzato a ottenere i valori dei segreti con nomi che iniziano con greengrass-.

- Per ottenere i valori dei segreti locali, le funzioni Lambda definite dall'utente devono AWS IoT Greengrass utilizzare Core SDK v1.3.0 o versione successiva.

Fase 1: Creare un segreto di Secrets Manager

In questo passaggio, si utilizza la AWS Secrets Manager console per creare un segreto.

1. Accedere alla [console AWS Secrets Manager](#).

 Note

Per ulteriori informazioni su questo processo, consulta [Fase 1: Creare e archiviare il segreto AWS Secrets Manager nella Guida per l'AWS Secrets Manager utente](#).

2. Scegli Archivia un nuovo segreto.
3. In Scegli il tipo di segreto, scegli Altro tipo di segreto.
4. In Specify the key-value pairs to be stored for this secret (Specifica le coppie chiave-valore da archiviare per questo segreto):
 - In Chiave, inserire **test**.
 - In Valore, specifica **abcdefghi**.
5. Mantieni aws/secretsmanager selezionato per la chiave di crittografia, quindi scegli Avanti.

 Note

Non ti viene addebitato alcun costo AWS KMS se utilizzi la chiave AWS gestita predefinita creata da Secrets Manager nel tuo account.

6. In Secret name (Nome segreto), immetti **greengrass-TestSecret**, quindi seleziona Next (Avanti).

 Note

Per impostazione predefinita, il ruolo di servizio Greengrass consente di AWS IoT Greengrass ottenere il valore dei segreti con nomi che iniziano con greengrass -. Per ulteriori informazioni, consulta [Requisiti dei segreti](#).

7. Questo tutorial non richiede la rotazione, quindi scegli disabilita la rotazione automatica, quindi scegli Avanti.
8. Nella pagina Review (Revisione), rivedi le impostazioni e quindi scegli Store (Archivia).

Dovrai quindi creare una risorsa segreta nel gruppo Greengrass che faccia riferimento al segreto.

Fase 2: aggiunta di una risorsa segreta a un gruppo Greengrass

In questo passaggio, configurate una risorsa di gruppo che fa riferimento al segreto di Secrets Manager.

1. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).
2. Scegli il gruppo a cui aggiungere la risorsa segreta.
3. Nella pagina di configurazione del gruppo, scegli la scheda Risorse, quindi scorri verso il basso fino alla sezione Segreti. La sezione Segreti mostra le risorse segrete che appartengono al gruppo. È possibile aggiungere, modificare e rimuovere risorse segrete da questa sezione.

 Note

In alternativa, la console consente di creare una risorsa segreta e segreta quando si configura un connettore o una funzione Lambda. È possibile eseguire questa operazione dalla pagina Configura parametri del connettore o dalla pagina Risorse della funzione Lambda.

4. Scegli Aggiungi nella sezione Segreti.
5. Nella pagina Aggiungi una risorsa segreta, inserisci **MyTestSecret** il nome della risorsa.
6. In Segreto, scegli greengrass-. TestSecret
7. Nella sezione Seleziona etichette (opzionale), l'etichetta AWSCURRENT staging rappresenta la versione più recente del segreto. Questa etichetta è sempre inclusa in una risorsa segreta.

 Note

Questo tutorial richiede solo l' **AWSCURRENT** etichetta. Facoltativamente, puoi includere le etichette richieste dalla funzione o dal connettore Lambda.

8. Scegliere Add resource (Aggiungi risorsa).

Fase 3: Creare un pacchetto di distribuzione della funzione Lambda

Per creare una funzione Lambda, devi prima creare un pacchetto di distribuzione della funzione Lambda che contenga il codice della funzione e le dipendenze. Le funzioni Greengrass Lambda richiedono il [AWS IoT Greengrass Core SDK](#) per attività come la comunicazione con i messaggi MQTT nell'ambiente principale e l'accesso ai segreti locali. Questo tutorial crea una funzione Python, quindi utilizzi la versione Python dell'SDK nel pacchetto di distribuzione.

 Note

Per ottenere i valori dei segreti locali, le funzioni Lambda definite dall'utente devono AWS IoT Greengrass utilizzare Core SDK v1.3.0 o versione successiva.

1. Dalla pagina dei download di [AWS IoT Greengrass Core SDK](#), scarica AWS IoT Greengrass Core SDK per Python sul tuo computer.
2. Decomprimere il pacchetto scaricato per ottenere l'SDK. Il kit SDK è la cartella greengrasssdk.
3. Salvare la seguente funzione del codice Python nel file locale `secret_test.py`.

```
import greengrasssdk

secrets_client = greengrasssdk.client("secretsmanager")
iot_client = greengrasssdk.client("iot-data")
secret_name = "greengrass-TestSecret"
send_topic = "secrets/output"

def function_handler(event, context):
    """
    Gets a secret and publishes a message to indicate whether the secret was
    successfully retrieved.
```

```
"""
response = secrets_client.get_secret_value(SecretId=secret_name)
secret_value = response.get("SecretString")
message = (
    f"Failed to retrieve secret {secret_name}."
    if secret_value is None
    else f"Successfully retrieved secret {secret_name}."
)
iot_client.publish(topic=send_topic, payload=message)
print("Published: " + message)
```

La `get_secret_value` funzione supporta il nome o l'ARN del segreto di Secrets Manager per il `SecretId` valore. Questo esempio utilizza il nome segreto. Per questo esempio secret, AWS IoT Greengrass restituisce la coppia chiave-valore: `{"test": "abcdefghi"}`

Important

Assicurati che le funzioni Lambda definite dall'utente gestiscano i segreti in modo sicuro e non registrino alcun dato sensibile archiviato nel segreto. Per ulteriori informazioni, consulta [Mitigare i rischi di registrazione e debug della funzione Lambda nella Guida per l'utente](#). AWS Secrets Manager. Sebbene questa documentazione si riferisca specificamente alle funzioni di rotazione, la raccomandazione si applica anche alle funzioni Greengrass Lambda.

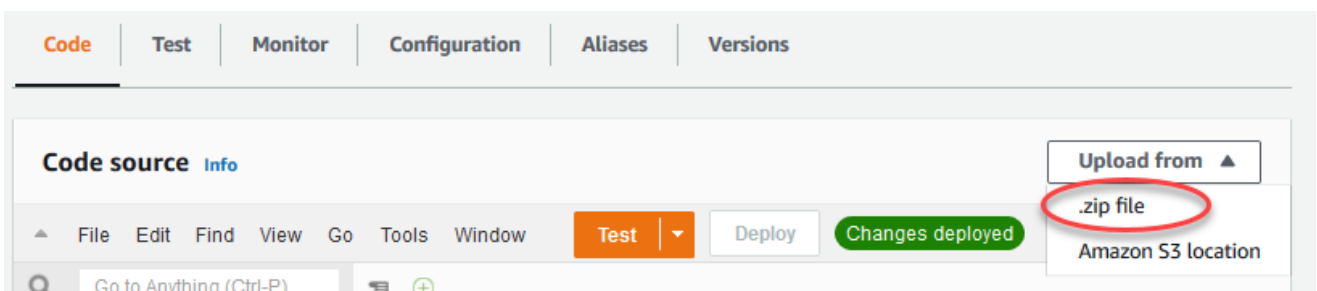
4. Comprimere le voci seguenti nel file `secret_test_python.zip`. Durante la creazione del file ZIP, includi solo il codice e le dipendenze e non la cartella che li contiene.
 - `secret_test.py`. La logica dell'app.
 - `greengrasssdk`. Libreria richiesta per tutte le funzioni Lambda di Python Greengrass.

Questo è il tuo pacchetto di implementazione della funzione Lambda.

Fase 4: Creazione di una funzione Lambda

In questo passaggio, si utilizza la AWS Lambda console per creare una funzione Lambda e configurarla per utilizzare il pacchetto di distribuzione. In seguito, pubblicherai una versione della funzione e creerai un alias.

1. Innanzitutto, crea la funzione Lambda.
 - a. In AWS Management Console, scegli Servizi e apri la AWS Lambda console.
 - b. Scegli Crea funzione, quindi scegli Autore da zero.
 - c. Nella sezione Basic information (Informazioni di base), specifica i seguenti valori:
 - Nel campo Function name (Nome funzione), immettere **SecretTest**.
 - In Runtime, scegliere Python 3.7.
 - Per le autorizzazioni, mantieni l'impostazione predefinita. Questo crea un ruolo di esecuzione che concede le autorizzazioni Lambda di base. Questo ruolo non viene utilizzato da AWS IoT Greengrass
 - d. Nella parte inferiore della pagina, scegli Crea funzione.
2. Quindi, registra il gestore e carica il pacchetto di distribuzione della funzione Lambda.
 - a. Nella scheda Codice, in Codice sorgente, scegli Carica da. Dal menu a discesa, scegli un file.zip.



- b. Scegli Carica, quindi scegli il pacchetto di `secret_test_python.zip` distribuzione. Quindi, scegliere Save (Salva).
- c. Nella scheda Codice della funzione, in Impostazioni di runtime, scegli Modifica, quindi inserisci i seguenti valori.
 - In Runtime, scegliere Python 3.7.
 - In Handler (Gestore), immetti **secret_test.function_handler**
- d. Seleziona Salva.

 Note

Il pulsante Test sulla AWS Lambda console non funziona con questa funzione. Il AWS IoT Greengrass Core SDK non contiene moduli necessari per eseguire le funzioni Greengrass Lambda in modo indipendente nella console. AWS Lambda Questi moduli (ad esempio, `greengrass_common`) vengono forniti alle funzioni dopo essere stati distribuiti nel core Greengrass.

3. Ora, pubblica la prima versione della tua funzione Lambda e crea un [alias per la](#) versione.

 Note

I gruppi Greengrass possono fare riferimento a una funzione Lambda tramite alias (consigliato) o per versione. L'utilizzo di un alias semplifica la gestione degli aggiornamenti del codice perché non è necessario modificare la tabella di sottoscrizione o la definizione del gruppo quando il codice della funzione viene aggiornato. Invece, è sufficiente indirizzare l'alias alla nuova versione della funzione.

- a. Nel menu Actions (Operazioni), seleziona Publish new version (Pubblica nuova versione).
- b. Per Version description (Descrizione versione), immettere **First version**, quindi scegliere Publish (Pubblica).
- c. Nella pagina di configurazione SecretTest: 1, dal menu Azioni, scegli Crea alias.
- d. Nella pagina Create a new alias (Crea un nuovo alias), utilizza i seguenti valori:
 - Per Nome, immetti **GG_SecretTest**.
 - In Version (Versione), selezionare 1.

 Note

AWS IoT Greengrass non supporta gli alias Lambda per le versioni \$LATEST.

- e. Scegli Create (Crea) .

Ora sei pronto per aggiungere la funzione Lambda al tuo gruppo Greengrass e allegare la risorsa segreta.

Fase 5: Aggiungere la funzione Lambda al gruppo Greengrass

In questo passaggio, aggiungi la funzione Lambda al gruppo Greengrass nella console. AWS IoT

1. Nella pagina di configurazione del gruppo, scegli la scheda Funzioni Lambda.
2. Nella sezione Funzioni My Lambda, scegli Aggiungi.
3. Per la funzione Lambda, scegli. SecretTest
4. Per la versione della funzione Lambda, scegli l'alias della versione che hai pubblicato.

Quindi, configura il ciclo di vita della funzione Lambda.

1. Nella sezione di configurazione della funzione Lambda, apporta i seguenti aggiornamenti.

Note

Ti consigliamo di eseguire la funzione Lambda senza containerizzazione a meno che il tuo business case non lo richieda. Ciò consente l'accesso alla GPU e alla fotocamera del dispositivo senza configurare le risorse del dispositivo. Se esegui senza containerizzazione, devi anche concedere l'accesso root alle tue funzioni Lambda AWS IoT Greengrass .

a. Per eseguire senza containerizzazione:

- Per Utente e gruppo di sistema, scegli. **Another user ID/group ID** Per ID utente di sistema, immettere0. Per ID del gruppo di sistema, immettere0.

Ciò consente alla funzione Lambda di funzionare come root. Per ulteriori informazioni sull'esecuzione come root, consulta [the section called “Impostazione dell'identità di accesso predefinita per le funzioni Lambda in un gruppo”](#).

 Tip

È inoltre necessario aggiornare il `config.json` file per concedere l'accesso root alla funzione Lambda. Per la procedura, vedi [the section called “Esecuzione di una funzione Lambda come utente root”](#).

- Per la containerizzazione della funzione Lambda, scegli Nessun contenitore.

Per ulteriori informazioni sull'esecuzione senza containerizzazione, consulta [the section called “Considerazioni sulla scelta della containerizzazione delle funzioni Lambda”](#)

- Per Timeout, immettere **10 seconds**.
- Per Pinned, scegli True.

Per ulteriori informazioni, consulta [the section called “Configurazione del ciclo di vita”](#).

- In Parametro aggiuntivo, per Accesso in lettura alla directory `/sys`, scegli Abilitato.

b. Per eseguire invece in modalità containerizzata:

 Note

Si sconsiglia l'esecuzione in modalità containerizzata a meno che il business case non lo richieda.

- Per Utente e gruppo di sistema, scegli Usa i valori predefiniti del gruppo.
- Per la containerizzazione delle funzioni Lambda, scegli Usa default di gruppo.
- Per Memory limit (Limite memoria), immettere **1024 MB**.
- Per Timeout, immettere **10 seconds**.
- Per Pinned, scegli True.

Per ulteriori informazioni, consulta [the section called “Configurazione del ciclo di vita”](#).

- In Parametri aggiuntivi, per Accesso in lettura alla directory `/sys`, scegli Abilitato.

2. Scegli Aggiungi funzione Lambda.

Quindi, associa la risorsa segreta alla funzione.

Passaggio 6: collegare la risorsa segreta alla funzione Lambda

In questo passaggio, associ la risorsa segreta alla funzione Lambda nel tuo gruppo Greengrass. Ciò associa la risorsa alla funzione, che consente alla funzione di ottenere il valore del segreto locale.

1. Nella pagina di configurazione del gruppo, scegli la scheda Funzioni Lambda.
2. Scegli la SecretTestfunzione.
3. Nella pagina dei dettagli della funzione, scegli Risorse.
4. Scorri fino alla sezione Segreti e scegli Associa.
5. Scegli MyTestSecret, quindi scegli Associa.

Fase 7: aggiunta di sottoscrizioni al gruppo Greengrass

In questo passaggio, aggiungi abbonamenti che consentono lo scambio di AWS IoT messaggi e la funzione Lambda. Un abbonamento consente di AWS IoT richiamare la funzione e l'altro consente alla funzione di inviare dati di output a. AWS IoT

1. Nella pagina di configurazione del gruppo, scegli la scheda Abbonamenti, quindi scegli Aggiungi abbonamento.
2. Crea un abbonamento che AWS IoT consenta di pubblicare messaggi sulla funzione.

Nella pagina di configurazione del gruppo, scegli la scheda Abbonamenti, quindi scegli Aggiungi abbonamento.

3. Nella pagina Crea un abbonamento, configura l'origine e la destinazione come segue:
 - a. In Tipo di sorgente, scegli la funzione Lambda, quindi scegli IoT Cloud.
 - b. In Tipo di destinazione, scegli Servizio, quindi scegli SecretTest.
 - c. Nel filtro Argomento **secrets/input**, inserisci, quindi scegli Crea abbonamento.
4. Aggiungere un secondo abbonamento. Scegli la scheda Abbonamenti, scegli Aggiungi abbonamento e configura l'origine e la destinazione come segue:
 - a. In Tipo di origine, scegli Servizi, quindi scegli SecretTest.
 - b. Nel tipo di Target, scegli la funzione Lambda, quindi scegli IoT Cloud.
 - c. Nel filtro Argomento, inserisci **secrets/output**, quindi scegli Crea abbonamento.

Fase 8: distribuzione del gruppo Greengrass

Distribuire il gruppo al nuovo dispositivo core. Durante la distribuzione, AWS IoT Greengrass recupera il valore del segreto da Secrets Manager e crea una copia locale crittografata sul core.

1. Assicurati che il AWS IoT Greengrass core sia in esecuzione. Esegui i seguenti comandi nel terminale di Raspberry Pi in base alle esigenze.

- a. Per controllare se il daemon è in esecuzione:

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

Se l'output contiene una voce root per `/greengrass/ggc/packages/ggc-version/bin/daemon`, allora il daemon è in esecuzione.

Note

La versione indicata nel percorso dipende dalla versione del software AWS IoT Greengrass Core installata sul dispositivo principale.

- b. Per avviare il demone:

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

2. Nella pagina di configurazione del gruppo, scegli Deploy.
3.
 - a. Nella scheda Funzioni Lambda, nella sezione Funzioni System Lambda, seleziona Rilevatore IP e scegli Modifica.
 - b. Nella finestra di dialogo Modifica impostazioni del rilevatore IP, seleziona Rileva e sostituisci automaticamente gli endpoint del broker MQTT.
 - c. Seleziona Salva.

Questo consente ai dispositivi di acquisire automaticamente informazioni di base sulla connettività, come, ad esempio indirizzo IP, DNS e numero della porta. Il rilevamento automatico è consigliato, ma supporta AWS IoT Greengrass anche gli endpoint specificati manualmente. Ti viene chiesto il metodo di individuazione solo la prima volta che il gruppo viene distribuito.

 Note

Se richiesto, concedi l'autorizzazione a creare il ruolo di [servizio Greengrass](#) e associarlo al Account AWS tuo ruolo attuale. Regione AWS Questo ruolo consente di accedere AWS IoT Greengrass alle tue risorse nei AWS servizi.

Nella pagina Deployments (Distribuzioni) vengono visualizzati il timestamp della distribuzione, l'ID versione e lo stato. Una volta completata, lo stato visualizzato per la distribuzione dovrebbe essere Completato.

Per la risoluzione dei problemi, consultare [Risoluzione dei problemi](#).

Test della funzione Lambda

1. Nella home page della AWS IoT console, scegli Test.
2. Per Iscriviti all'argomento, utilizza i seguenti valori, quindi scegli Iscriviti.

Proprietà	Valore
Argomento sottoscrizione	secrets/output
Visualizzazione payload MQTT	Visualizza i payload come stringhe

3. Per Pubblica su argomento, utilizzate i seguenti valori, quindi scegliete Pubblica per richiamare la funzione.

Proprietà	Valore
Argomento	secrets/input
Messaggio	Mantenere il messaggio predefinito. La pubblicazione di un messaggio richiama la funzione Lambda, ma la funzione in questo tutorial non elabora il corpo del messaggio.

In caso di esito positivo, la funzione pubblica il messaggio "Success" ("Successo").

Consulta anche

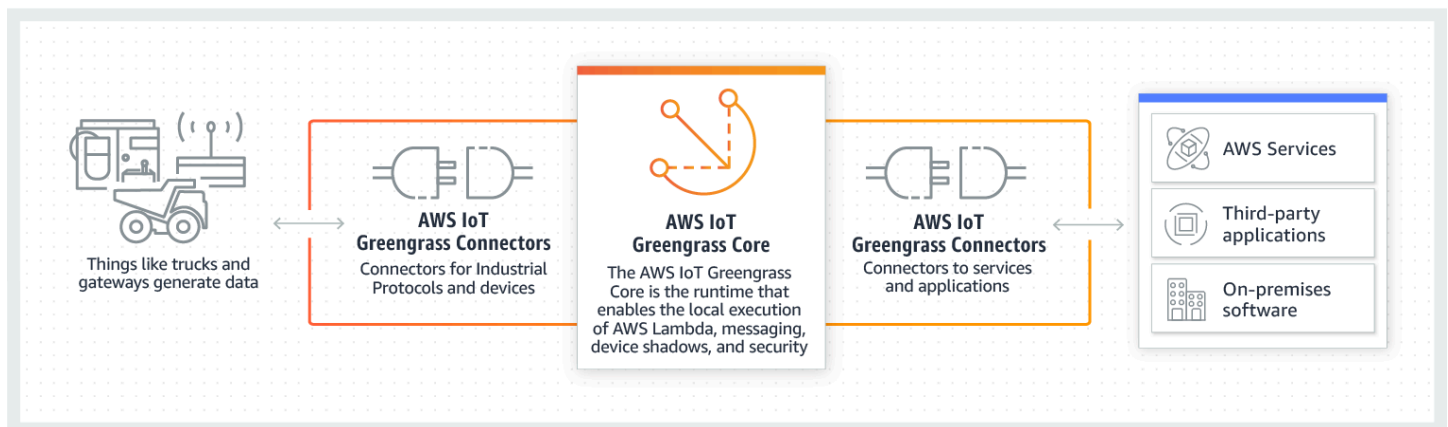
- [Distribuzione dei segreti nel core](#)

Integrazione con servizi e protocolli tramite i connettori Greengrass

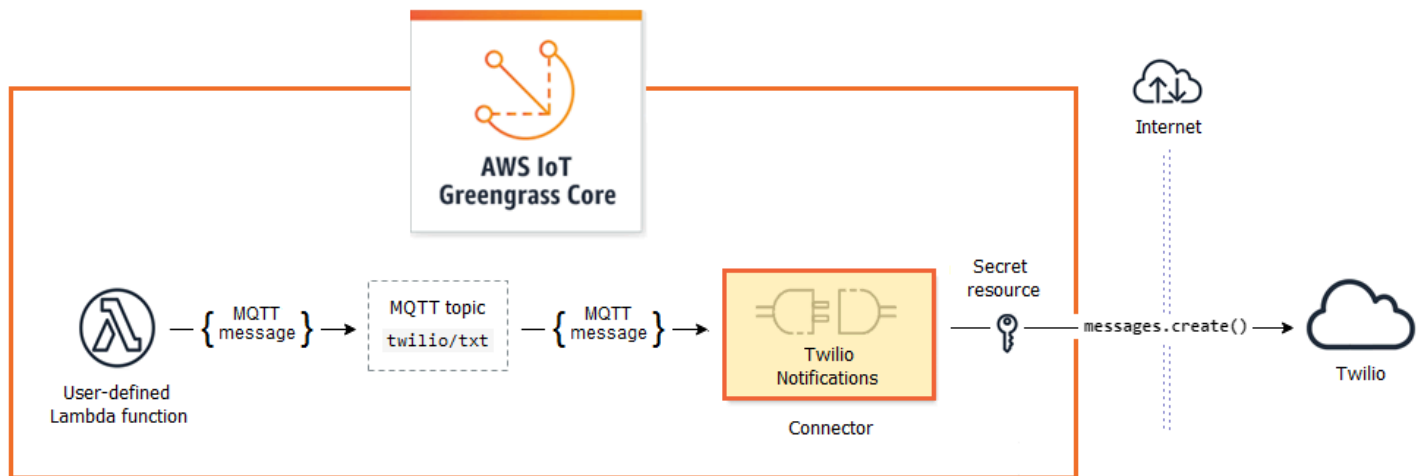
Questa funzionalità è disponibile per AWS IoT Greengrass Core v1.7 e versioni successive.

I connettori AWS IoT Greengrass sono moduli predefiniti che rendono più efficiente l'interazione con l'infrastruttura locale, i protocolli dei dispositivi e altri servizi cloud. AWS Utilizzando i connettori, puoi dedicare meno tempo all'apprendimento di nuovi protocolli APIs e dedicare più tempo alla logica che conta per la tua azienda.

Il diagramma seguente mostra dove i connettori possono inserirsi nel AWS IoT Greengrass panorama.



Molti connettori utilizzano messaggi MQTT per comunicare con i dispositivi client e le funzioni Greengrass Lambda del gruppo AWS IoT o con il servizio shadow locale. Nell'esempio seguente, il connettore Twilio Notifications riceve messaggi MQTT da una funzione Lambda definita dall'utente, utilizza un riferimento locale di un segreto AWS Secrets Manager da e chiama l'API Twilio.



Per i tutorial sulla creazione di questa soluzione, consulta [the section called “Nozioni di base sui connettori \(console\)”](#) e [the section called “Nozioni di base sui connettori \(CLI\)”](#).

I connettori Greengrass possono aiutarti a estendere le funzionalità dei dispositivi o a creare dispositivi monouso. Utilizzando i connettori, è possibile:

- L'implementazione di una logica di business riutilizzabile.
- Interagisci con servizi cloud e locali, inclusi AWS servizi di terze parti.
- L'integrazione e l'elaborazione dei dati del dispositivo.
- Abilita device-to-device le chiamate utilizzando gli abbonamenti agli argomenti MQTT e le funzioni Lambda definite dall'utente.

AWS fornisce una serie di connettori Greengrass che semplificano le interazioni con servizi e fonti di dati comuni. Questi moduli precostituiti attivano gli scenari di logging e diagnostica, rifornimento, elaborazione dei dati industriali, allarmi e messaggistica. Per ulteriori informazioni, consulta [the section called “AWS-connettori Greengrass forniti”](#).

Requisiti

Per utilizzare i connettori, tieni presente questi punti:

- Ogni connettore che utilizzi ha dei requisiti che devi soddisfare. Questi requisiti possono includere la versione minima del software AWS IoT Greengrass Core, i prerequisiti del dispositivo, le

autorizzazioni e i limiti richiesti. Per ulteriori informazioni, consulta [the section called “AWS-connettori Greengrass forniti”](#).

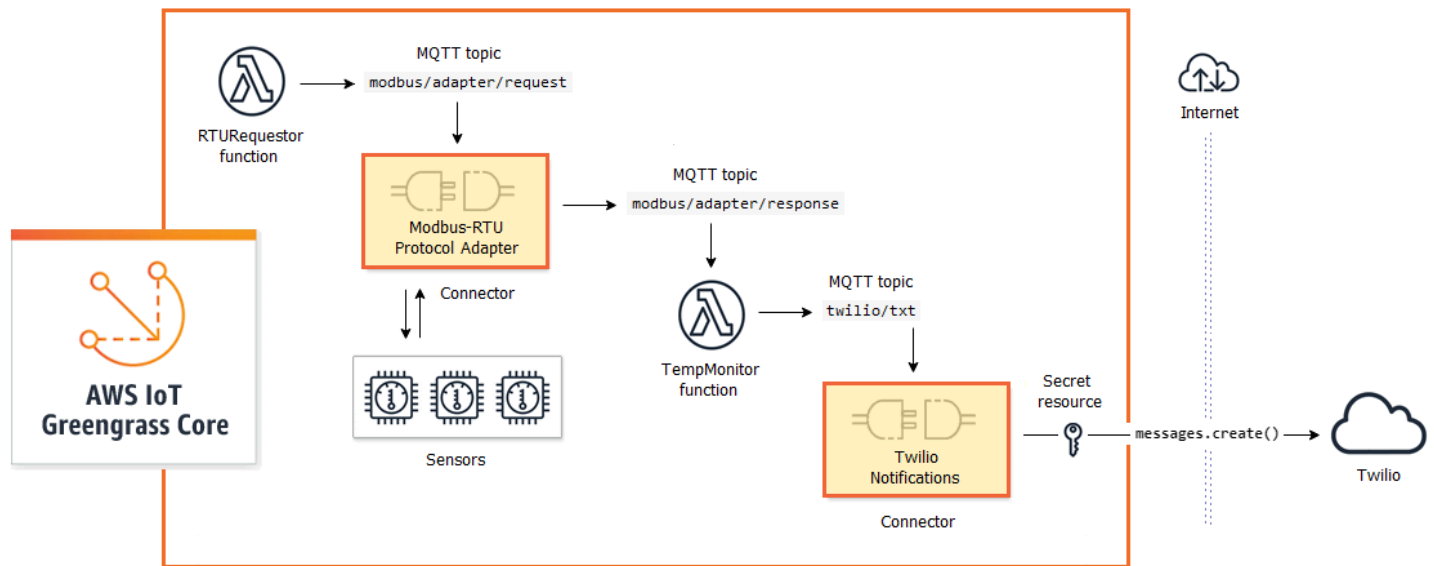
- Un gruppo Greengrass può contenere solo un'istanza configurata di un determinato connettore. Tuttavia, è possibile utilizzare l'istanza in più abbonamenti. Per ulteriori informazioni, consulta [the section called “Parametri di configurazione”](#).
- Quando la [containerizzazione predefinita](#) per il gruppo Greengrass è impostata su Nessun container, i connettori del gruppo devono essere eseguiti senza containerizzazione. Per trovare i connettori che supportano la modalità Nessun container consulta [the section called “AWS-connettori Greengrass forniti”](#).

Utilizzo dei connettori Greengrass

Un connettore è un tipo di componente di gruppo. Come altri componenti del gruppo, come i dispositivi client e le funzioni Lambda definite dall'utente, puoi aggiungere connettori ai gruppi, configurarne le impostazioni e distribuirli nel core. AWS IoT Greengrass I connettori vengono eseguiti nell'ambiente core.

È possibile distribuire alcuni connettori come semplici applicazioni autonome. Ad esempio, il connettore Device Defender legge le metriche di sistema dal dispositivo principale e le invia per l'analisi. AWS IoT Device Defender

È possibile aggiungere altri connettori come elementi costitutivi in soluzioni più grandi. La seguente soluzione di esempio utilizza il connettore Modbus-RTU Protocol Adapter per elaborare i messaggi provenienti dai sensori e il connettore Twilio Notifications per avviare i messaggi Twilio.



Le soluzioni spesso includono funzioni Lambda definite dall'utente che si trovano accanto ai connettori ed elaborano i dati che il connettore invia o riceve. In questo esempio, la TempMonitor funzione riceve dati dall'adattatore di protocollo Modbus-RTU, esegue alcune logiche aziendali e quindi invia i dati a Twilio Notifications.

Di seguito è riportata la procedura generica per creare e distribuire una soluzione:

1. Definire il flusso di dati di alto livello. Identificare le origini dati, i canali dei dati, i servizi, i protocolli e le risorse di cui hai bisogno. In questa soluzione di esempi, sono inclusi dati sul protocollo RTU Modbus, la porta seriale Modbus fisica e Twilio.
2. Individuare i connettori da includere nella soluzione e aggiungerli al gruppo. La soluzione di esempio utilizza l'adattatore di protocollo Modbus-RTU e le notifiche Twilio. Per aiutarti a trovare i connettori adeguati al tuo scenario e per ulteriori informazioni sui singoli requisiti, consulta [the section called “AWS-connettori Greengrass forniti”](#).
3. Identifica se sono necessarie funzioni Lambda, dispositivi client o risorse definite dall'utente, quindi creali e aggiungili al gruppo. Questo passaggio potrebbe includere funzioni che contengono logica di business o l'elaborazione di dati in un formato richiesto da un'altra entità della soluzione. La soluzione di esempio utilizza funzioni per inviare richieste Modbus RTU e avviare notifiche Twilio. Include anche una risorsa dispositivo locale per la porta seriale RTU Modbus e una risorsa segreta per il token di autenticazione Twilio.

 Note

Le risorse segrete fanno riferimento a password, token e altri segreti di AWS Secrets Manager. I segreti possono essere utilizzati dai connettori e dalle funzioni Lambda per l'autenticazione con servizi e applicazioni. Per impostazione predefinita, AWS IoT Greengrass può accedere ai segreti con nomi che iniziano con "greengrass-». Per ulteriori informazioni, consulta [Distribuzione dei segreti nel core](#).

4. Creazione di sottoscrizioni che consentono alle entità della soluzione di scambiarsi messaggi MQTT. Se un connettore viene utilizzato in una sottoscrizione, il connettore e il messaggio di origine o di destinazione devono utilizzare la sintassi dell'argomento predefinita supportata dal connettore. Per ulteriori informazioni, consulta [the section called "Input e output"](#).
5. Distribuzione del gruppo nel core Greengrass

Per informazioni sulla creazione e la distribuzione di un connettore, consulta i seguenti tutorial:

- [the section called "Nozioni di base sui connettori \(console\)"](#)
- [the section called "Nozioni di base sui connettori \(CLI\)"](#)

Parametri di configurazione

Molti connettori forniscono parametri che consentono di personalizzarne il funzionamento o l'output. Questi parametri vengono utilizzati durante l'inizializzazione, in fase di runtime o in altri momenti del ciclo di vita del connettore.

I tipi di parametri e il relativo utilizzo possono variare in base al connettore. Ad esempio, il connettore SNS ha un parametro che configura l'argomento SNS predefinito e Device Defender ha un parametro che configura la frequenza di campionamento dei dati.

Una versione di gruppo può contenere vari connettori, ma una sola istanza alla volta di un determinato connettore. Questo significa che ciascun connettore nel gruppo può avere solo una configurazione attiva. Tuttavia, l'istanza del connettore può essere utilizzata in più sottoscrizioni nel gruppo. Ad esempio, è possibile creare abbonamenti che consentono a molti dispositivi di inviare dati al connettore Kinesis Firehose.

Parametri utilizzati per l'accesso alle risorse di gruppo

I connettori Greengrass utilizzano le risorse di gruppo per accedere al file system, alle porte, alle periferiche e ad altre risorse locali sul dispositivo core. Se un connettore richiede l'accesso a una risorsa di gruppo, fornirà i relativi parametri di configurazione.

Le risorse di gruppo includono:

- [Risorse locali](#). Directory, file, porte, pin e periferiche presenti sul dispositivo core Greengrass.
- [Risorse di Machine Learning](#). Modelli Machine Learning che ricevono formazione nel cloud e vengono distribuiti al core per un'inferenza locale.
- [Risorse segrete](#). Copie locali crittografate di password, chiavi, token o testo arbitrario da AWS Secrets Manager. I connettori possono accedere in modo sicuro a questi segreti locali e utilizzarli per autenticare i servizi o l'infrastruttura locale.

Ad esempio, i parametri per Device Defender consentono l'accesso alle metriche di sistema nella `/proc` directory host e i parametri per le notifiche Twilio consentono l'accesso a un token di autenticazione Twilio memorizzato localmente.

Aggiornamento dei parametri del connettore

I parametri sono configurati quando il connettore viene aggiunto a un gruppo Greengrass. È possibile modificare i valori dei parametri dopo l'aggiunta del connettore.

- Nella console: dalla pagina di configurazione del gruppo, apri Connettori e dal menu contestuale del connettore scegli Modifica.

Note

Se il connettore utilizza una risorsa segreta che successivamente verrà modificata per fare riferimento a un altro segreto, è necessario modificare i parametri del connettore e confermare la modifica.

- Nell'API: crea un'altra versione del connettore che definisca la nuova configurazione.

L'AWS IoT Greengrass API utilizza versioni per gestire i gruppi. Le versioni sono immutabili, quindi per aggiungere o modificare i componenti del gruppo, ad esempio i dispositivi client, le funzioni e le risorse del gruppo, è necessario creare versioni di componenti nuovi o aggiornati. Quindi, si

crea e si distribuisce una versione di gruppo che contiene la versione di destinazione di ciascun componente.

Dopo aver modificato la configurazione del connettore, è necessario distribuire il gruppo per propagare le modifiche apportate al core.

Input e output

Molti connettori Greengrass possono comunicare con altre entità inviando e ricevendo messaggi MQTT. La comunicazione MQTT è controllata da abbonamenti che consentono a un connettore di scambiare dati con funzioni Lambda, dispositivi client e altri connettori del gruppo Greengrass o con AWS IoT e il servizio shadow locale. Per consentire questa comunicazione, devi creare sottoscrizioni nel gruppo a cui appartiene il connettore. Per ulteriori informazioni, consulta [the section called “Sottoscrizioni gestite nel flusso di lavoro di messaggistica MQTT”](#).

I connettori possono essere autori di messaggi, abbonati ai messaggi o entrambi. Ogni connettore definisce gli argomenti MQTT che pubblica o a cui è abbonato. Questi argomenti predefiniti devono essere utilizzati nelle sottoscrizioni in cui il connettore è l'origine o la destinazione di un messaggio. Per i tutorial che includono le fasi di configurazione delle sottoscrizioni a un connettore, consulta [the section called “Nozioni di base sui connettori \(console\)”](#) e [the section called “Nozioni di base sui connettori \(CLI\)”](#).

Note

Molti connettori dispongono anche di modalità di comunicazione integrate per interagire con il cloud o i servizi locali. Queste variano in base al connettore e potrebbero richiedere la configurazione di parametri o l'aggiunta di autorizzazioni al [ruolo del gruppo](#). Per informazioni sui requisiti del connettore, consultare [the section called “AWS-connettori Greengrass forniti”](#).

Argomenti di input

La maggior dei connettori ricevono dati su argomenti MQTT. Alcuni connettori effettuano la sottoscrizione a più argomenti per i dati di input. Ad esempio, il connettore Serial Stream supporta due argomenti:

- `serial/+/read/#`

- `serial/+/write/#`

Per questo connettore, le richieste di lettura e scrittura vengono inviate all'argomento corrispondente. Quando crei gli abbonamenti, assicurati di utilizzare l'argomento in linea con l'implementazione.

I caratteri `+` e `#` negli esempi precedenti sono caratteri jolly. Questi caratteri jolly consentono agli abbonati di ricevere messaggi su più argomenti e agli autori di personalizzare gli argomenti di destinazione.

- Il carattere jolly `+` appare ovunque nella gerarchia degli argomenti. Può essere sostituito da una voce della gerarchia.

Ad esempio, per l'argomento `sensor/+/input`, i messaggi possono essere pubblicati negli argomenti `sensor/id-123/input`, ma non in `sensor/group-a/id-123/input`.

- Il carattere jolly `#` può comparire solo alla fine della gerarchia dell'argomento. Può essere sostituito da zero o più elementi della gerarchia.

Ad esempio, per l'argomento `sensor/#`, i messaggi possono essere pubblicati in `sensor/`, `sensor/id-123` e `sensor/group-a/id-123`, ma non in `sensor.`

I caratteri jolly sono validi solo per l'abbonamento agli argomenti. Non è possibile pubblicare messaggi in argomenti contenenti caratteri jolly. Consulta la documentazione del connettore per ulteriori informazioni sui requisiti degli argomenti di input o output. Per ulteriori informazioni, consulta [the section called “AWS-connettori Greengrass forniti”](#).

Supporto per la containerizzazione

Per impostazione predefinita, la maggior parte dei connettori viene eseguita sul core di Greengrass in un ambiente di runtime isolato gestito da AWS IoT Greengrass. Questi ambienti di runtime, denominati container, forniscono l'isolamento tra i connettori e il sistema host, offrendo maggiore sicurezza per l'host e il connettore.

Tuttavia, questa containerizzazione Greengrass non è supportata in alcuni ambienti, ad esempio quando si esegue AWS IoT Greengrass in un contenitore Docker o su vecchi kernel Linux senza cgroup. In questi ambienti, i connettori devono essere eseguiti in modalità No container (Nessun container). Per trovare i connettori che supportano la modalità Nessun container consulta [the section](#)

called [“AWS-connettori Greengrass forniti”](#). Alcuni connettori vengono eseguiti in questa modalità in modo nativo e alcuni connettori consentono di impostare la modalità di isolamento.

Puoi inoltre impostare la modalità di isolamento su No container (Nessun container) in ambienti che supportano la containerizzazione di Greengrass, ma consigliamo di utilizzare la modalità Greengrass container (Container Greengrass) quando possibile.

Note

L'impostazione predefinita della containerizzazione per il gruppo Greengrass non si applica ai [connettori](#).

Aggiornamento delle versioni dei connettori

I provider di connettori potrebbero rilasciare nuove versioni di un connettore che aggiungono funzionalità, risolvono problemi o migliorano le prestazioni. Per informazioni sulle versioni disponibili e sulle modifiche correlate, consulta la [documentazione relativa a ciascun connettore](#).

Nella AWS IoT console, puoi verificare la presenza di nuove versioni per i connettori del tuo gruppo Greengrass.

1. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).
2. In Gruppi Greengrass, scegli il tuo gruppo.
3. Scegli Connectors (Connettori) per visualizzare i connettori nel gruppo.

Se il connettore ha una nuova versione, viene visualizzato un pulsante Disponibile nella colonna Aggiorna.

4. Per aggiornare la versione del connettore:
 - a. Nella pagina Connettori, nella colonna Aggiornamento, scegli Disponibile. Viene visualizzata la pagina Upgrade connector (Aggiorna connettore) in cui vengono visualizzate le impostazioni dei parametri correnti, se applicabili.

Scegli la nuova versione del connettore, definisci i parametri in base alle esigenze, quindi scegli Upgrade (Aggiorna).

- b. Nella pagina Subscriptions (Sottoscrizioni) aggiungi nuove sottoscrizioni nel gruppo per sostituire quelle che utilizzano il connettore come origine o destinazione. Quindi, rimuovi le vecchie sottoscrizioni.

Le sottoscrizioni fanno riferimento ai connettori in base alla versione, in modo che diventino non validi se si modifica la versione del connettore nel gruppo.

- c. Dal menu Azioni, scegli Distribuisci per distribuire le modifiche al core.

Per aggiornare un connettore dall' AWS IoT Greengrass API, crea e distribuisci una versione di gruppo che includa il connettore e gli abbonamenti aggiornati. Utilizza lo stesso processo seguito per aggiungere un connettore a un gruppo. Per i passaggi dettagliati che mostrano come utilizzare il connettore di notifica Twilio di esempio AWS CLI per configurare e distribuire un esempio di connettore di notifica Twilio, consulta. [the section called “Nozioni di base sui connettori \(CLI\)”](#)

Registrazione per connettori

I connettori Greengrass contengono funzioni Lambda che scrivono eventi ed errori nei log Greengrass. A seconda delle impostazioni del gruppo, i log vengono scritti in CloudWatch Logs, nel file system locale o in entrambi. I log dei connettori includono gli ARN della funzione corrispondente. L'esempio ARN seguente proviene dal connettore Kinesis Firehose:

```
arn:aws:lambda:aws-region:account-id:function:KinesisFirehoseClient:1
```

La configurazione di logging predefinita scrive log a livello di informazioni nel file system con la seguente struttura di directory:

```
greengrass-root/ggc/var/log/user/region/aws/function-name.log
```

Per ulteriori informazioni sulla registrazione di Greengrass, vedere. [the section called “Monitoraggio con i log AWS IoT Greengrass ”](#)

AWS-connettori Greengrass forniti

AWS fornisce i seguenti connettori che supportano AWS IoT Greengrass scenari comuni. Per ulteriori informazioni sul funzionamento dei connettori, consulta la documentazione seguente:

- [Integrazione con servizi e protocolli tramite i connettori](#)

- [Nozioni di base sui connettori \(console\)](#) o [Nozioni di base sui connettori \(CLI\)](#)

Connector	Descrizione	Runtime Lambda supportati	Supporta la modalità Nessun container
CloudWatch Metriche	Pubblica metriche personalizzate su Amazon. CloudWatch	• Python 3.8* • Python 3.7 • Python 2.7	Sì
Dispositivo Defender	Invia le metriche di sistema a. AWS IoT Device Defender	• Python 3.8* • Python 3.7 • Python 2.7	No
Distribuzione delle applicazioni Docker	Esegue un file di Docker Compose per avviare un'applicazione Docker sul dispositivo core.	• Python 3.8 • Python 3.7	Sì
IoT Analytics	Invia dati da dispositivi e sensori a AWS IoT Analytics.	• Python 3.8* • Python 3.7 • Python 2.7	Sì
Adattatore di protocollo IP Ethernet IoT	Raccoglie dati dai dispositivi EtherNet/IP.	• Java 8	Sì
IoT SiteWise	Invia i dati provenienti da dispositivi e sensori a proprietà degli asset in AWS IoT SiteWise.	• Java 8	Sì
Kinesis Firehose	Invia dati ai flussi di distribuzione di Amazon Data Firehose.	• Python 3.8* • Python 3.7 • Python 2.7	Sì

Connector	Descrizione	Runtime Lambda supportati	Supporta la modalità Nessun container
Feedback ML	Pubblica l'input del modello di machine learning nel cloud e l'output in un argomento MQTT.	• Python 3.8 • Python 3.7	No
Classificazione delle immagini ML	Esegue un servizio locale di inferenza di classificazione delle immagini. Questo connettore fornisce le versioni per diverse piattaforme.	• Python 3.8* • Python 3.7 • Python 2.7	No
Rilevamento di oggetti ML	Esegue un servizio di inferenza di rilevamento oggetti locale. Questo connettore fornisce le versioni per diverse piattaforme.	• Python 3.8 • Python 3.7	No
Adattatore di protocollo Modbus-RTU	Invia le richieste ai dispositivi RTU Modbus.	• Python 3.8* • Python 3.7 • Python 2.7	No
Adattatore di protocollo Modbus-TCP	Raccoglie dati dai dispositivi ModbusTCP.	• Java 8	Sì
Raspberry Pi GPIO	Controlla i pin GPIO in un dispositivo core Raspberry Pi.	• Python 3.8* • Python 3.7 • Python 2.7	No
Flusso seriale	Effettua operazioni di lettura e scrittura in una porta seriale del dispositivo core.	• Python 3.8* • Python 3.7 • Python 2.7	No
ServiceNow MetricBase Integration	Pubblica le metriche delle serie temporali su ServiceNow MetricBase	• Python 3.8* • Python 3.7 • Python 2.7	Sì

Connector	Descrizione	Runtime Lambda supportati	Supporta la modalità Nessun container
SNS	Invia messaggi a un argomento di Amazon SNS.	• Python 3.8* • Python 3.7 • Python 2.7	Sì
Integrazione con Splunk	Pubblica dati su Splunk HEC.	• Python 3.8* • Python 3.7 • Python 2.7	Sì
Notifiche Twilio	Avvia un messaggio di testo o vocale Twilio.	• Python 3.8* • Python 3.7 • Python 2.7	Sì

* Per utilizzare i runtime di Python 3.8, è necessario creare un collegamento simbolico dalla cartella di installazione predefinita di Python 3.7 ai binari Python 3.8 installati. Per ulteriori informazioni, consulta i requisiti specifici del connettore.

Note

Si consiglia di [aggiornare le versioni dei connettori](#) da Python 2.7 a Python 3.7. Il supporto continuo per i connettori Python 2.7 dipende dal supporto in AWS Lambda fase di esecuzione. Per ulteriori informazioni, consulta la [politica di supporto di Runtime](#) nella AWS Lambda Developer Guide.

CloudWatch Connettore Metrics

Il [connettore CloudWatch Metrics](#) pubblica metriche personalizzate dai dispositivi Greengrass su Amazon. CloudWatch Il connettore fornisce un'infrastruttura centralizzata per la pubblicazione delle CloudWatch metriche, che puoi utilizzare per monitorare e analizzare l'ambiente principale di Greengrass e agire sugli eventi locali. Per ulteriori informazioni, consulta [Using Amazon CloudWatch metrics](#) nella Amazon CloudWatch User Guide.

Questo connettore riceve i dati dei parametri come messaggi MQTT. Il connettore raggruppa i parametri che si trovano nello stesso namespace e li pubblica a intervalli regolari. CloudWatch

Questo connettore ha le seguenti versioni.

Versione	ARN
5	arn:aws:greengrass: <i>region</i> ::/connectors/CloudWatchMetrics/versions/5
4	arn:aws:greengrass: <i>region</i> ::/connectors/CloudWatchMetrics/versions/4
3	arn:aws:greengrass: <i>region</i> ::/connectors/CloudWatchMetrics/versions/3
2	arn:aws:greengrass: <i>region</i> ::/connectors/CloudWatchMetrics/versions/2
1	arn:aws:greengrass: <i>region</i> ::/connectors/CloudWatchMetrics/versions/1

Per informazioni sulle modifiche di ogni versione, consulta [Changelog](#).

Requisiti

Questo connettore presenta i seguenti requisiti:

Version 3 - 5

- AWS IoT Greengrass Software principale v1.9.3 o successivo.
- [Python](#) versione 3.7 o 3.8 installata sul dispositivo principale e aggiunta alla variabile di ambiente PATH.

 Note

Per usare Python 3.8, esegui il seguente comando per creare un collegamento simbolico dalla cartella di installazione predefinita di Python 3.7 ai binari Python 3.8 installati.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Questo configura il dispositivo in modo che soddisfi il requisito Python per AWS IoT Greengrass.

- Il [ruolo del gruppo Greengrass](#) è configurato per consentire l'azione `cloudwatch:PutMetricData`, come illustrato nel seguente esempio di policy AWS Identity and Access Management (IAM).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1528133056761",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

Per il requisito del ruolo di gruppo, è necessario configurare il ruolo in modo da concedere le autorizzazioni necessarie e assicurarsi che il ruolo sia stato aggiunto al gruppo. Per ulteriori informazioni, consulta [the section called “Gestione del ruolo del gruppo \(console\)”](#) o [the section called “Gestire il ruolo del gruppo \(CLI\)”](#).

Per ulteriori informazioni sulle CloudWatch autorizzazioni, consulta [Amazon CloudWatch permissions reference](#) nella IAM User Guide.

Versions 1 - 2

- AWS IoT Greengrass Software di base v1.7 o successivo.
- [Python](#) versione 2.7 installato sul dispositivo principale e aggiunto alla variabile di ambiente PATH.
- Il [ruolo del gruppo Greengrass](#) è configurato per consentire l'azione `cloudwatch:PutMetricData`, come illustrato nel seguente esempio di policy AWS Identity and Access Management (IAM).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1528133056761",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

Per il requisito del ruolo di gruppo, è necessario configurare il ruolo in modo da concedere le autorizzazioni necessarie e assicurarsi che il ruolo sia stato aggiunto al gruppo. Per ulteriori informazioni, consulta [the section called “Gestione del ruolo del gruppo \(console\)”](#) o [the section called “Gestire il ruolo del gruppo \(CLI\)”](#).

Per ulteriori informazioni sulle CloudWatch autorizzazioni, consulta [Amazon CloudWatch permissions reference](#) nella IAM User Guide.

Parametri del connettore

Questo connettore fornisce i seguenti parametri:

Versions 4 - 5

PublishInterval

Il numero massimo di secondi di attesa prima di pubblicare i parametri raggruppati per un dato spazio dei nomi. Il valore massimo è 900. Per configurare il connettore in modo da pubblicare i parametri man mano che vengono ricevuti (senza raggruppamento), specificare 0.

Il connettore pubblica su CloudWatch dopo aver ricevuto 20 metriche nello stesso namespace o dopo l'intervallo specificato.

Note

Il connettore non garantisce l'ordine degli eventi di pubblicazione.

Nome visualizzato nella console: Intervallo di pubblicazione AWS IoT

Obbligatorio: true

Tipo: string

Valori validi: 0 - 900

Schema valido: [0-9] | [1-9]\d | [1-9]\d\d | 900

PublishRegion

La Regione AWS CloudWatch metrica su cui pubblicare. Questo valore sostituisce la regione predefinita dei parametri Greengrass. È obbligatorio solo durante la pubblicazione di parametri tra regioni.

Nome visualizzato nella AWS IoT console: regione di pubblicazione

Obbligatorio: false

Tipo: string

Schema valido: ^\$|([a-z]{2}-[a-z]+-\d{1})

MemorySize

La memoria (in KB) da allocare al connettore.

Nome visualizzato nella AWS IoT console: dimensione della memoria

Obbligatorio: `true`

Tipo: `string`

Schema valido: `^[0-9]+$`

`MaxMetricsToRetain`

Il numero massimo di parametri in tutti gli spazi dei nomi da salvare in memoria prima che vengano sostituiti da nuovi parametri. Il valore minimo è 2000.

Questo limite è valido quando non è presente una connessione a Internet e il connettore inizia il buffering dei parametri da pubblicare successivamente. Quando il buffer è pieno, i parametri meno recenti vengono sostituiti da quelli nuovi. I parametri in un determinato spazio dei nomi vengono sostituiti solo da quelli dello stesso spazio dei nomi.

Note

I parametri non vengono salvati se si interrompe il processo host del connettore. Ad esempio, questa interruzione potrebbe verificarsi durante la distribuzione dei gruppi o al riavvio del dispositivo.

Nome visualizzato nella AWS IoT console: numero massimo di metriche da conservare

Obbligatorio: `true`

Tipo: `string`

Schema valido: `^([2-9]\d{3}|[1-9]\d{4,})$`

`IsolationMode`

Modalità di [containerizzazione](#) per questo connettore. L'impostazione predefinita è `GreengrassContainer`, il che significa che il connettore viene eseguito in un ambiente di runtime isolato all'interno del AWS IoT Greengrass contenitore.

Note

L'impostazione predefinita della containerizzazione per il gruppo non si applica ai connettori.

Nome visualizzato nella AWS IoT console: modalità di isolamento del contenitore

Obbligatorio: false

Tipo: string

Valori validi: GreengrassContainer o NoContainer

Schema valido: ^NoContainer\$|^GreengrassContainer\$

Versions 1 - 3

PublishInterval

Il numero massimo di secondi di attesa prima di pubblicare i parametri raggruppati per un dato spazio dei nomi. Il valore massimo è 900. Per configurare il connettore in modo da pubblicare i parametri man mano che vengono ricevuti (senza raggruppamento), specificare 0.

Il connettore pubblica su CloudWatch dopo aver ricevuto 20 metriche nello stesso namespace o dopo l'intervallo specificato.

Note

Il connettore non garantisce l'ordine degli eventi di pubblicazione.

Nome visualizzato nella console: Intervallo di pubblicazione AWS IoT

Obbligatorio: true

Tipo: string

Valori validi: 0 - 900

Schema valido: [0-9] | [1-9]\d | [1-9]\d\d | 900

PublishRegion

La Regione AWS CloudWatch metrica su cui pubblicare. Questo valore sostituisce la regione predefinita dei parametri Greengrass. È obbligatorio solo durante la pubblicazione di parametri tra regioni.

Nome visualizzato nella AWS IoT console: regione di pubblicazione

Obbligatorio: `false`

Tipo: `string`

Schema valido: `^[a-z]{2}-[a-z]+\d{1}`

MemorySize

La memoria (in KB) da allocare al connettore.

Nome visualizzato nella AWS IoT console: dimensione della memoria

Obbligatorio: `true`

Tipo: `string`

Schema valido: `^[0-9]+$`

MaxMetricsToRetain

Il numero massimo di parametri in tutti gli spazi dei nomi da salvare in memoria prima che vengano sostituiti da nuovi parametri. Il valore minimo è 2000.

Questo limite è valido quando non è presente una connessione a Internet e il connettore inizia il buffering dei parametri da pubblicare successivamente. Quando il buffer è pieno, i parametri meno recenti vengono sostituiti da quelli nuovi. I parametri in un determinato spazio dei nomi vengono sostituiti solo da quelli dello stesso spazio dei nomi.

Note

I parametri non vengono salvati se si interrompe il processo host del connettore. Ad esempio, questa interruzione potrebbe verificarsi durante la distribuzione dei gruppi o al riavvio del dispositivo.

Nome visualizzato nella AWS IoT console: numero massimo di metriche da conservare

Obbligatorio: `true`

Tipo: `string`

Schema valido: `^([2-9]\d{3}|[1-9]\d{4,})$`

Esempio di creazione di un connettore (AWS CLI)

Il seguente comando CLI crea un `ConnectorDefinition` con una versione iniziale che contiene il connettore CloudWatch Metrics.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-  
version '{  
  "Connectors": [  
    {  
      "Id": "MyCloudWatchMetricsConnector",  
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/CloudWatchMetrics/  
versions/4",  
      "Parameters": {  
        "PublishInterval" : "600",  
        "PublishRegion" : "us-west-2",  
        "MemorySize" : "16",  
        "MaxMetricsToRetain" : "2500",  
        "IsolationMode" : "GreengrassContainer"  
      }  
    }  
  ]  
}'
```

Nella AWS IoT Greengrass console, puoi aggiungere un connettore dalla pagina Connettori del gruppo. Per ulteriori informazioni, consulta [the section called “Nozioni di base sui connettori \(console\)”](#).

Dati di input

Questo connettore accetta metriche su un argomento MQTT e le pubblica su. CloudWatch I messaggi di input devono essere in formato JSON.

Filtro argomento in sottoscrizione

`cloudwatch/metric/put`

Proprietà dei messaggi

`request`

Informazioni sui parametri di questo messaggio.

L'oggetto della richiesta contiene i dati dei parametri da pubblicare in CloudWatch. I valori delle metriche devono soddisfare le specifiche dell'[PutMetricDataAPI](#). Sono obbligatorie solo le proprietà `namespace`, `metricData.metricName` e `metricData.value`.

Obbligatorio: `true`

Tipo: `object` che include le seguenti proprietà:

`namespace`

Lo spazio dei nomi definito dall'utente per i dati metrici in questa richiesta. CloudWatch utilizza i namespace come contenitori per i punti dati metrici.

 Note

Non è possibile specificare uno spazio dei nomi che inizi con la stringa riservata.
`AWS/`

Obbligatorio: `true`

Tipo: `string`

Schema valido: `^[^:]*`

`metricData`

I dati del parametro.

Richiesto: `true`

Tipo: `object` che include le seguenti proprietà:

`metricName`

Nome del parametro.

Obbligatorio: `true`

Tipo: `string`

`dimensions`

Le dimensioni associate al parametro. Le dimensioni forniscono ulteriori informazioni sul parametro e sui relativi dati. Un parametro è in grado di definire fino a 10 dimensioni.

Questo connettore include automaticamente una dimensione denominata `coreName`, dove il valore è il nome del core.

Obbligatorio: `false`

Tipo: array di oggetti dimensionali che includono le seguenti proprietà:

`name`

Il nome della dimensione.

Obbligatorio: `false`

Tipo: `string`

`value`

Il valore della dimensione.

Richiesto: `false`

Tipo: `string`

`timestamp`

L'ora in cui i dati della metrica sono stati ricevuti, espressa come numero di secondi trascorsi da alloraJan 1, 1970 00:00:00 UTC. Se questo valore viene omissso, il connettore utilizza l'ora di ricezione del messaggio.

Obbligatorio: `false`

Tipo: `timestamp`

 Note

Se utilizzi tra le versioni 1 e 4 di questo connettore, ti consigliamo di recuperare il timestamp separatamente per ogni metrica quando invii più metriche da un'unica fonte. Non utilizzare una variabile per memorizzare il timestamp.

`value`

Il valore del parametro.

 Note

CloudWatch rifiuta valori troppo piccoli o troppo grandi. I valori devono essere compresi nell'intervallo da $8.515920e-109$ a $1.174271e+108$ (Base 10) o da $2e-360$ a $2e360$ (Base 2). I valori speciali (ad esempio, NaN, +Infinity, -Infinity) non sono supportati.

Obbligatorio: true

Tipo: double

unit

Unità del parametro.

Richiesto: false

Tipo: string

Valori validi: Seconds, Microseconds, Milliseconds, Bytes, Kilobytes, Megabytes, Gigabytes, Terabytes, Bits, Kilobits, Megabits, Gigabits, Terabits, Percent, Count, Bytes/Second, Kilobytes/Second, Megabytes/Second, Gigabytes/Second, Terabytes/Second, Bits/Second, Kilobits/Second, Megabits/Second, Gigabits/Second, Terabits/Second, Count/Second, None

Limiti

Tutti i limiti imposti dall' CloudWatch [PutMetricData](#) API si applicano alle metriche quando si utilizza questo connettore. Di seguito sono riportati i limiti particolarmente importanti:

- Limite di 40 KB sul payload dell'API
- 20 parametri per ciascuna richiesta API
- 150 transazioni al secondo (TPS) per l'API PutMetricData

Per ulteriori informazioni, consulta [CloudWatch i limiti](#) nella Amazon CloudWatch User Guide.

Input di esempio

```
{
```

```
{
  "request": {
    "namespace": "Greengrass",
    "metricData": {
      {
        "metricName": "latency",
        "dimensions": [
          {
            "name": "hostname",
            "value": "test_hostname"
          }
        ],
        "timestamp": 1539027324,
        "value": 123.0,
        "unit": "Seconds"
      }
    }
  }
}
```

Dati di output

Questo connettore pubblica le informazioni di stato come dati di output su un argomento MQTT.

Filtro argomento in sottoscrizione

cloudwatch/metric/put/status

Output di esempio: Operazione riuscita

La risposta include lo spazio dei nomi dei dati metrici e il RequestId campo della risposta.
CloudWatch

```
{
  "response": {
    "cloudwatch_rid": "70573243-d723-11e8-b095-75ff2EXAMPLE",
    "namespace": "Greengrass",
    "status": "success"
  }
}
```

Esempio di output: Errore

```
{
```

```
"response" : {  
    "namespace": "Greengrass",  
    "error": "InvalidInputException",  
    "error_message": "cw metric is invalid",  
    "status": "fail"  
}  
}
```

Note

Se il connettore rileva un errore ripetibile (ad esempio errori di connessione), riprova la pubblicazione nel batch successivo.

Esempio di utilizzo

Usa i seguenti passaggi di alto livello per configurare una funzione Lambda di esempio di Python 3.7 che puoi usare per provare il connettore.

Note

- Se usi altri runtime Python, puoi creare un collegamento simbolico da Python3.x a Python 3.7.
- Gli argomenti [Nozioni di base sui connettori \(console\)](#) e [Nozioni di base sui connettori \(CLI\)](#) contengono passaggi dettagliati che illustrano come configurare e distribuire un connettore Twilio Notifications di esempio.

1. Assicurarsi di soddisfare i [requisiti](#) per il connettore.

Per il requisito del ruolo di gruppo, è necessario configurare il ruolo in modo da concedere le autorizzazioni necessarie e assicurarsi che il ruolo sia stato aggiunto al gruppo. Per ulteriori informazioni, consulta [the section called “Gestione del ruolo del gruppo \(console\)”](#) o [the section called “Gestire il ruolo del gruppo \(CLI\)”](#).

2. Crea e pubblica una funzione Lambda che invia dati di input al connettore.

Salvare il [codice di esempio](#) come file PY. Scarica e decomprimi il [AWS IoT Greengrass Core SDK per Python](#). Quindi, crea un pacchetto zip che contiene il file PY e la cartella

greengrasssdk a livello root. Questo pacchetto zip è il pacchetto di distribuzione in cui carichi. AWS Lambda

Dopo aver creato la funzione Python 3.7 Lambda, pubblica una versione della funzione e crea un alias.

3. Configurare il gruppo Greengrass.

- a. Aggiungi la funzione Lambda tramite il relativo alias (consigliato). Configura il ciclo di vita Lambda come longevo (o nella "Pinned": true CLI).
- b. Aggiungere il connettore e configurarne i relativi [parametri](#).
- c. Aggiungere sottoscrizioni che consentono al connettore di ricevere [i dati di input](#) e inviare [i dati di output](#) nei filtri degli argomenti supportati.
 - Imposta la funzione Lambda come origine, il connettore come destinazione e utilizza un filtro per argomenti di input supportato.
 - Imposta il connettore come origine, AWS IoT Core come destinazione e utilizza un filtro per l'argomento di output supportato. Utilizzi questo abbonamento per visualizzare i messaggi di stato nella AWS IoT console.

4. Distribuisci il gruppo.

5. Nella AWS IoT console, nella pagina Test, sottoscrivi l'argomento relativo ai dati di output per visualizzare i messaggi di stato dal connettore. La funzione Lambda di esempio è di lunga durata e inizia a inviare messaggi subito dopo l'implementazione del gruppo.

Al termine del test, puoi impostare il ciclo di vita Lambda su richiesta (o nella CLI) e "Pinned": false distribuire il gruppo. Ciò impedisce alla funzione di inviare messaggi.

Esempio

L'esempio seguente della funzione Lambda invia un messaggio di input al connettore.

```
import greengrasssdk
import time
import json

iot_client = greengrasssdk.client('iot-data')
send_topic = 'cloudwatch/metric/put'

def create_request_with_all_fields():
```

```
    return {
        "request": {
            "namespace": "Greengrass_CW_Connector",
            "metricData": {
                "metricName": "Count1",
                "dimensions": [
                    {
                        "name": "test",
                        "value": "test"
                    }
                ],
                "value": 1,
                "unit": "Seconds",
                "timestamp": time.time()
            }
        }
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
                       payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

Licenze

Il connettore CloudWatch Metrics include i seguenti software/licenze di terze parti:

- [AWS SDK per Python \(Boto3\)](#)/Apache License 2.0
- [botocore](#)/Apache License 2.0
- [dateutil](#)/PSF License
- [docutils](#)/BSD License, GNU General Public License (GPL), Python Software Foundation License, Public Domain
- [jmespath](#)/MIT License
- [s3transfer](#)/Apache License 2.0
- [urllib3](#)/MIT License

Questo connettore è rilasciato ai sensi del contratto di [licenza del software Greengrass Core](#).

Changelog

La tabella seguente descrive le modifiche apportate a ciascuna versione del connettore.

Versione	Modifiche
5	Correzione dell'aggiunta del supporto per timestamp duplicati nei dati di input.
4	Aggiunto il parametro <code>IsolationMode</code> per configurare la modalità di containerizzazione per il connettore.
3	È stato aggiornato il runtime Lambda a Python 3.7, che modifica i requisiti di runtime.
2	Correggere per ridurre l'eccessiva registrazione di log.
1	Versione iniziale.

Un gruppo Greengrass può contenere una sola versione del connettore alla volta. Per informazioni sull'aggiornamento di una versione del connettore, consulta [the section called “Aggiornamento delle versioni dei connettori”](#).

Consulta anche

- [Integrazione con servizi e protocolli tramite i connettori](#)
- [the section called “Nozioni di base sui connettori \(console\)”](#)
- [the section called “Nozioni di base sui connettori \(CLI\)”](#)
- [Utilizzo dei CloudWatch parametri di Amazon](#) nella Amazon CloudWatch User Guide
- [PutMetricData](#) nell'Amazon CloudWatch API Reference

Connettore Device Defender

Il [connettore](#) Device Defender notifica agli amministratori le modifiche allo stato di un dispositivo core Greengrass. Ciò consente di identificare un comportamento anomalo che potrebbe indicare la compromissione del dispositivo.

Questo connettore legge le metriche di sistema dalla `/proc` directory sul dispositivo principale, quindi le pubblica su AWS IoT Device Defender. Per i dettagli sui rapporti sulle metriche, consulta le specifiche del [documento sulle metriche del dispositivo](#) nella Guida per gli sviluppatori AWS IoT.

Questo connettore ha le seguenti versioni.

Versione	ARN
3	<code>arn:aws:greengrass: <i>region</i>::/connectors/DeviceDefender/versions/3</code>
2	<code>arn:aws:greengrass: <i>region</i>::/connectors/DeviceDefender/versions/2</code>
1	<code>arn:aws:greengrass: <i>region</i>::/connectors/DeviceDefender/versions/1</code>

Per informazioni sulle modifiche di ogni versione, consulta [Changelog](#).

Requisiti

Questo connettore presenta i seguenti requisiti:

Version 3

- AWS IoT Greengrass Software principale v1.9.3 o successivo.
- [Python](#) versione 3.7 o 3.8 installata sul dispositivo principale e aggiunta alla variabile di ambiente PATH.

 Note

Per usare Python 3.8, esegui il seguente comando per creare un collegamento simbolico dalla cartella di installazione predefinita di Python 3.7 ai binari Python 3.8 installati.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Questo configura il dispositivo in modo che soddisfi il requisito Python per AWS IoT Greengrass.

- AWS IoT Device Defender configurato per utilizzare la funzione Detect per tenere traccia delle violazioni. Per ulteriori informazioni, consulta [Detect](#) nella AWS IoT Developer Guide.
- Una [risorsa di volume locale](#) nel gruppo Greengrass che punta alla /proc directory. La risorsa deve utilizzare le seguenti proprietà:
 - Percorso di origine: /proc
 - Percorso di destinazione: /host_proc (o un valore che corrisponda al [modello valido](#))
 - AutoAddGroupOwner: true
- La libreria [psutil](#) installata sul core Greengrass. La versione 5.7.0 è la versione più recente verificata per funzionare con il connettore.
- La libreria [cbor](#) installata sul core Greengrass. La versione 1.0.0 è l'ultima versione verificata per il funzionamento con il connettore.

Versions 1 - 2

- AWS IoT Greengrass Software principale v1.7 o successivo.
- [Python](#) versione 2.7 installato sul dispositivo principale e aggiunto alla variabile di ambiente PATH.
- AWS IoT Device Defender configurato per utilizzare la funzione Detect per tenere traccia delle violazioni. Per ulteriori informazioni, consulta [Detect](#) nella AWS IoT Developer Guide.
- Una [risorsa di volume locale](#) nel gruppo Greengrass che punta alla /proc directory. La risorsa deve utilizzare le seguenti proprietà:
 - Percorso di origine: /proc
 - Percorso di destinazione: /host_proc (o un valore che corrisponda al [modello valido](#))

- `AutoAddGroupOwner`: `true`
- La libreria [psutil](#) installata sul core Greengrass.
- La libreria [cbor](#) installata sul core Greengrass.

Parametri del connettore

Questo connettore fornisce i seguenti parametri:

`SampleIntervalSeconds`

Il numero di secondi tra ciascun ciclo di raccolta e produzione del rapporto dei parametri. Il valore minimo è di 300 secondi (5 minuti).

Nome visualizzato nella AWS IoT console: Intervallo di segnalazione delle metriche

Obbligatorio: `true`

Tipo: `string`

Schema valido: `^[0-9]*(?:3[0-9][0-9]|[4-9][0-9]{2}|[1-9][0-9]{3,})$`

`ProcDestinationPath-ResourceId`

L'ID della risorsa del volume `/proc`.

Note

Al connettore è concesso l'accesso in sola lettura alla risorsa.

Nome visualizzato nella AWS IoT console: Resource for `/proc` directory

Obbligatorio: `true`

Tipo: `string`

Schema valido: `[a-zA-Z0-9_-]+`

`ProcDestinationPath`

Il percorso di destinazione della risorsa del volume `/proc`.

Nome visualizzato nella AWS IoT console: percorso di destinazione della risorsa /proc

Obbligatorio: true

Tipo: string

Schema valido: `\/[a-zA-Z0-9_-]+`

Esempio di creazione di un connettore (AWS CLI)

Il seguente comando CLI crea un `ConnectorDefinition` con una versione iniziale che contiene il connettore Device Defender.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-  
version '{  
  "Connectors": [  
    {  
      "Id": "MyDeviceDefenderConnector",  
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/DeviceDefender/  
versions/3",  
      "Parameters": {  
        "SampleIntervalSeconds": "600",  
        "ProcDestinationPath": "/host_proc",  
        "ProcDestinationPath-ResourceId": "my-proc-resource"  
      }  
    }  
  ]  
}'
```

Note

La funzione Lambda in questo connettore ha un ciclo di vita di [lunga durata](#).

Nella AWS IoT Greengrass console, puoi aggiungere un connettore dalla pagina Connettori del gruppo. Per ulteriori informazioni, consulta [the section called “Nozioni di base sui connettori \(console\)”](#).

Dati di input

Questo connettore non accetta messaggi MQTT come dati di input.

Dati di output

Questo connettore pubblica le metriche di sicurezza AWS IoT Device Defender come dati di output.

Filtro argomento in sottoscrizione

```
$aws/things/+/defender/metrics/json
```

Note

Questa è la sintassi dell'argomento prevista. AWS IoT Device Defender Il connettore sostituisce il carattere jolly + con il nome del dispositivo (ad esempio, \$aws/things/*thing-name*/defender/metrics/json).

Output di esempio

Per i dettagli relativi ai rapporti sulle metriche, consulta le [specifiche del documento sulle metriche dei dispositivi](#) nella Guida per gli sviluppatori.AWS IoT

```
{
  "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
      }
    }
  }
}

```

Licenze

Questo connettore è rilasciato ai sensi del contratto di [licenza del software Greengrass Core](#).

Changelog

La tabella seguente descrive le modifiche apportate a ciascuna versione del connettore.

Versione	Modifiche
3	È stato aggiornato il runtime Lambda a Python 3.7, che modifica i requisiti di runtime.
2	Correggere per ridurre l'eccessiva registrazione di log.
1	Versione iniziale.

Un gruppo Greengrass può contenere una sola versione del connettore alla volta. Per informazioni sull'aggiornamento di una versione del connettore, consulta [the section called “Aggiornamento delle versioni dei connettori”](#).

Consulta anche

- [Integrazione con servizi e protocolli tramite i connettori](#)
- [the section called “Nozioni di base sui connettori \(console\)”](#)
- [the section called “Nozioni di base sui connettori \(CLI\)”](#)
- [Device Defender](#) nella Guida per gli sviluppatori AWS IoT

Connettore di distribuzione dell'applicazione Docker

Il connettore di distribuzione delle applicazioni Greengrass Docker semplifica l'esecuzione delle immagini Docker su un core. AWS IoT Greengrass Il connettore utilizza Docker Compose per avviare un'applicazione Docker multi-container da un file `docker-compose.yml`. In particolare, il connettore esegue i comandi `docker-compose` per gestire container Docker su un singolo dispositivo core. Per ulteriori informazioni, consulta [Overview of Docker Compose](#) nella documentazione di Docker. Il connettore può accedere alle immagini Docker archiviate nei registri di container Docker, come Amazon Elastic Container Registry (Amazon ECR), Docker Hub e registri affidabili Docker privati.

Dopo aver distribuito il gruppo Greengrass, il connettore recupera le immagini più recenti e avvia i contenitori Docker. Eseguire il comando `and. docker-compose pull docker-compose up`

Quindi, il connettore pubblica lo stato del comando in un argomento [MQTT di output](#). Inoltre, registra le informazioni di stato relative all'esecuzione dei container Docker. In questo modo è possibile monitorare i log delle applicazioni in Amazon CloudWatch. Per ulteriori informazioni, consulta [the section called “Monitoraggio con i log AWS IoT Greengrass”](#). Il connettore avvia inoltre container Docker ad ogni riavvio del daemon Greengrass. Il numero di container Docker che possono essere eseguiti sul core dipende dall'hardware in uso.

I container Docker vengono eseguiti all'esterno del dominio Greengrass sul dispositivo core, pertanto non possano accedere alla comunicazione tra processi (IPC) del core. Tuttavia, è possibile configurare alcuni canali di comunicazione con componenti Greengrass, come le funzioni Lambda locali. Per ulteriori informazioni, consulta [the section called “Comunicazione con i container Docker”](#).

Puoi utilizzare il connettore per scenari come l'hosting di un server Web o di un server MySQL sul dispositivo core. I servizi locali nelle applicazioni Docker possono comunicare tra loro, con altri processi nell'ambiente locale e con i servizi cloud. Ad esempio, puoi eseguire un server Web sul core che invia le richieste dalle funzioni Lambda a un servizio Web nel cloud.

Questo connettore viene eseguito in modalità di isolamento [Nessun container](#), quindi è possibile distribuirlo in un gruppo Greengrass che viene eseguito senza la containerizzazione di Greengrass.

Questo connettore ha le seguenti versioni.

Versione	ARN
7	arn:aws:greengrass: <i>region</i> ::/connectors/DockerApplicationDeployment/versions/7
6	arn:aws:greengrass: <i>region</i> ::/connectors/DockerApplicationDeployment/versions/6
5	arn:aws:greengrass: <i>region</i> ::/connectors/DockerApplicationDeployment/versions/5
4	arn:aws:greengrass: <i>region</i> ::/connectors/DockerApplicationDeployment/versions/4

Versione	ARN
3	arn:aws:greengrass: <i>region</i> ::/connectors/DockerApplicationDeployment/versions/3
2	arn:aws:greengrass: <i>region</i> ::/connectors/DockerApplicationDeployment/versions/2
1	arn:aws:greengrass: <i>region</i> ::/connectors/DockerApplicationDeployment/versions/1

Per informazioni sulle modifiche di ogni versione, consulta [Changelog](#).

Requisiti

Questo connettore presenta i seguenti requisiti:

- AWS IoT Greengrass Software principale v1.10 o successivo.

Note

Questo connettore non è supportato nelle distribuzioni. OpenWrt

- [Python](#) versione 3.7 o 3.8 installata sul dispositivo principale e aggiunta alla variabile di ambiente PATH.

Note

Per usare Python 3.8, esegui il seguente comando per creare un collegamento simbolico dalla cartella di installazione predefinita di Python 3.7 ai binari Python 3.8 installati.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Questo configura il dispositivo in modo che soddisfi il requisito Python per AWS IoT Greengrass.

- Un minimo di 36 MB di RAM sul core Greengrass per consentire al connettore per monitorare i container Docker in esecuzione. Il requisito di memoria totale dipende dal numero di container Docker in esecuzione sul core.
- [Docker Engine](#) v1.1.9.1 o versione successiva installato sul core Greengrass. La versione 19.0.3 è l'ultima versione verificata per il funzionamento con il connettore.

L'eseguibile `docker` deve trovarsi nella directory `/usr/bin` o `/usr/local/bin`.

 Important

Ti consigliamo di installare un archivio delle credenziali per proteggere le copie locali delle credenziali Docker. Per ulteriori informazioni, consulta [the section called “Note sulla sicurezza”](#).

Per informazioni sull'installazione di Docker su distribuzioni Amazon Linux, consulta [Docker basics for Amazon ECS nella Amazon](#) Elastic Container Service Developer Guide.

- [Docker Compose](#) installato sul core Greengrass. L'eseguibile `docker-compose` deve trovarsi nella directory `/usr/bin` o `/usr/local/bin`.

Le seguenti versioni di Docker Compose sono state verificate per il funzionamento con il connettore.

Versione connettore	Versione Docker Compose verificata
7	1.25.4
6	1.25.4
5	1.25.4
4	1.25.4
3	1.25.4
2	1.25.1
1	1.24.1

- Un singolo file Docker Compose (ad esempio, `docker-compose.yml`), archiviato in Amazon Simple Storage Service (Amazon S3). Il formato deve essere compatibile con la versione di Docker Compose installata sul core. È opportuno testare il file prima di utilizzarlo sul core. Se modifichi il file dopo la distribuzione del gruppo Greengrass, devi ridistribuire il gruppo per aggiornare la copia locale sul core.
- Un utente Linux che dispone dell'autorizzazione per chiamare il daemon Docker locale e scrivere nella directory che archivia la copia locale del file Compose. Per ulteriori informazioni, consulta [Configurazione dell'utente Docker sul core](#).
- Il [ruolo del gruppo Greengrass](#) configurato per consentire l'operazione `s3:GetObject` sul bucket S3 che contiene il file Compose. Questa autorizzazione è mostrata nel seguente esempio di politica IAM.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAccessToComposeFileS3Bucket",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::bucket-name/*"
    }
  ]
}
```

Note

Se il tuo bucket S3 è abilitato al controllo delle versioni, allora il ruolo deve essere configurato anche per consentire l'azione. `s3:GetObjectVersion` Per ulteriori informazioni, consulta [Using versioning](#) nella Amazon Simple Storage Service User Guide.

Per il requisito del ruolo di gruppo, è necessario configurare il ruolo in modo da concedere le autorizzazioni necessarie e assicurarsi che il ruolo sia stato aggiunto al gruppo. Per ulteriori informazioni, consulta [the section called “Gestione del ruolo del gruppo \(console\)”](#) o [the section called “Gestire il ruolo del gruppo \(CLI\)”](#).

- Se il file Docker Compose fa riferimento a un'immagine Docker archiviata in Amazon ECR, il [ruolo del gruppo Greengrass](#) è configurato per consentire quanto segue:
 - `ecr:GetDownloadUrlForLayer` `ecr:BatchGetImage` azioni sui tuoi repository Amazon ECR che contengono le immagini Docker.
 - Operazione `ecr:GetAuthorizationToken` sulle risorse.

I repository devono trovarsi nello stesso Account AWS e Regione AWS nel connettore.

Important

Le autorizzazioni nel ruolo di gruppo possono essere assunte da tutte le funzioni e i connettori Lambda del gruppo Greengrass. Per ulteriori informazioni, consulta [the section called "Note sulla sicurezza"](#).

Queste autorizzazioni vengono mostrate nella policy di esempio riportata di seguito.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowGetEcrRepositories",
      "Effect": "Allow",
      "Action": [
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage"
      ],
      "Resource": [
        "arn:aws:ecr:region:account-id:repository/repository-name"
      ]
    },
    {
      "Sid": "AllowGetEcrAuthToken",
      "Effect": "Allow",
      "Action": "ecr:GetAuthorizationToken",
      "Resource": "*"
    }
  ]
}
```

Per ulteriori informazioni, consulta gli [esempi di policy dei repository di Amazon ECR](#) nella Amazon ECR User Guide.

Per il requisito del ruolo di gruppo, è necessario configurare il ruolo in modo da concedere le autorizzazioni necessarie e assicurarsi che il ruolo sia stato aggiunto al gruppo. Per ulteriori informazioni, consulta [the section called “Gestione del ruolo del gruppo \(console\)”](#) o [the section called “Gestire il ruolo del gruppo \(CLI\)”](#).

- Se il file di Docker Compose fa riferimento a un'immagine Docker di [Marketplace AWS](#), anche il connettore dispone dei seguenti requisiti:
 - Devi essere abbonato ai prodotti container. Marketplace AWS Per ulteriori informazioni, consulta la sezione relativa all'[individuazione e alla sottoscrizione ai prodotti container](#) nella Marketplace AWS Subscribers Guide.
 - AWS IoT Greengrass deve essere configurato per supportare i segreti locali, come descritto in [Secrets Requirements](#). Il connettore utilizza questa funzionalità solo per recuperare i segreti da AWS Secrets Manager, non per archivarli.
 - È necessario creare un segreto in Secrets Manager per ogni Marketplace AWS registro che memorizza un'immagine Docker a cui si fa riferimento nel file Compose. Per ulteriori informazioni, consulta [the section called “Accesso alle immagini Docker da repository privati”](#).
- Se il tuo file Docker Compose fa riferimento a un'immagine Docker da archivi privati in registri diversi da Amazon ECR, come Docker Hub, il connettore ha anche i seguenti requisiti:
 - AWS IoT Greengrass [deve essere configurato per supportare i segreti locali, come descritto in Secrets Requirements](#). Il connettore utilizza questa funzionalità solo per recuperare i segreti da AWS Secrets Manager, non per archivarli.
 - È necessario creare un segreto in Secrets Manager per ogni repository privato che memorizza un'immagine Docker a cui si fa riferimento nel file Compose. Per ulteriori informazioni, consulta [the section called “Accesso alle immagini Docker da repository privati”](#).
- Il daemon Docker deve essere in esecuzione quando si distribuisce un gruppo Greengrass contenente questo connettore.

Accesso alle immagini Docker da repository privati

Se utilizzi le credenziali per accedere alle immagini Docker, il connettore deve poter accedere ad esse. La modalità utilizzata dipende dalla posizione dell'immagine Docker.

Per le immagini Docker archiviate in Amazon ECR, concedi l'autorizzazione a ottenere il token di autorizzazione nel ruolo del gruppo Greengrass. Per ulteriori informazioni, consulta [the section called "Requisiti"](#).

Per le immagini Docker archiviate in altri archivi o registri privati, è necessario creare un account segreto per archiviare le informazioni di accesso. AWS Secrets Manager Sono incluse le immagini Docker a cui ti sei abbonato. Marketplace AWS Crea un segreto per ogni repository. Se aggiorni i tuoi segreti in Secrets Manager, le modifiche si propagano al core la prossima volta che distribuisce il gruppo.

Note

Secrets Manager è un servizio che puoi utilizzare per archiviare e gestire in modo sicuro credenziali, chiavi e altri segreti in. Cloud AWS Per ulteriori informazioni, consulta [Cos'è? AWS Secrets Manager](#) nella Guida AWS Secrets Manager per l'utente.

Ogni segreto deve contenere le seguenti chiavi:

Chiave	Valore
username	Il nome utente utilizzato per accedere al repository o al registro.
password	La password utilizzata per accedere al repository o al registro.
registryUrl	L'endpoint del registro. Questo deve corrispondere all'URL del registro corrispondente nel file Compose.

Note

Per consentire l'accesso AWS IoT Greengrass a un segreto per impostazione predefinita, il nome del segreto deve iniziare con greengrass-. In caso contrario, il ruolo del servizio Greengrass deve concedere l'accesso. Per ulteriori informazioni, consulta [the section called "AWS IoT Greengrass Consenti di ottenere valori segreti"](#).

Per ottenere le informazioni di accesso per le immagini Docker da Marketplace AWS

1. Ottieni la password per le immagini Docker Marketplace AWS da usando il `aws ecr get-login-password` comando. Per ulteriori informazioni, consulta la sezione [get-login-password](#) nella Documentazione di riferimento della AWS CLI .

```
aws ecr get-login-password
```

2. Recupera l'URL del registro per l'immagine Docker. Apri il Marketplace AWS sito Web e apri la pagina di lancio del prodotto container. In Immagini del contenitore, scegli Visualizza i dettagli dell'immagine del contenitore per individuare il nome utente e l'URL del registro.

Usa il nome utente, la password e l'URL del registro recuperati per creare un segreto per ogni Marketplace AWS registro che memorizza le immagini Docker a cui si fa riferimento nel file Compose.

Per creare segreti (console)

Nella AWS Secrets Manager console, scegli Altro tipo di segreti. In Specify the key-value pairs to be stored for this secret (Specifica le coppie chiave-valore da archiviare per questo segreto), aggiungi righe per username, password e registryUrl. Per ulteriori informazioni, consulta [Creazione di un segreto di base](#) nella Guida AWS Secrets Manager per l'utente.

Specify the key/value pairs to be stored in this secret [Info](#)

Secret key/value	Plaintext	
username	Mary_Major	Remove
password	abc123xyz456	Remove
registryUrl	https://docker.io	Remove

[+ Add row](#)

Per creare segreti (CLI)

In AWS CLI, utilizzare il `create-secret` comando Secrets Manager, come illustrato nell'esempio seguente. Per ulteriori informazioni, vedere [create-secret](#) nel AWS CLI Command Reference.

```
aws secretsmanager create-secret --name greengrass-MySecret --secret-string [{"username":"Mary_Major"}, {"password":"abc123xyz456"}, {"registryUrl":"https://docker.io"}]
```

Important

È responsabilità dell'utente proteggere la directory `DockerComposeFileDestinationPath` che archivia il file di Docker Compose e le credenziali per le immagini Docker di repository privati. Per ulteriori informazioni, consulta [the section called "Note sulla sicurezza"](#).

Parametri

Questo connettore fornisce i seguenti parametri:

Version 7

`DockerComposeFileS3Bucket`

Il nome del bucket S3 che contiene il file di Docker Compose. Quando crei il bucket, assicurati di seguire [le regole per i nomi dei bucket](#) descritte nella Amazon Simple Storage Service User Guide.

Nome visualizzato nella AWS IoT console: file Docker Compose in S3

Note

Nella console, la proprietà Docker Compose file in S3 (File di Docker Compose in S3) combina i parametri `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` e `DockerComposeFileS3Version`.

Obbligatorio: true

Tipo: string

Schema valido [a-zA-Z0-9\\-\\.]{3,63}

DockerComposeFileS3Key

La chiave oggetto per il tuo file Docker Compose in Amazon S3. Per ulteriori informazioni, incluse le linee guida per la denominazione delle chiavi degli oggetti, consulta [Chiave oggetto e metadati](#) nella Guida per l'utente di Amazon Simple Storage Service.

Note

Nella console, la proprietà Docker Compose file in S3 (File di Docker Compose in S3) combina i parametri DockerComposeFileS3Bucket, DockerComposeFileS3Key e DockerComposeFileS3Version.

Richiesto: true

Tipo: string

Schema valido .+

DockerComposeFileS3Version

La versione dell'oggetto per il tuo file Docker Compose in Amazon S3. Per ulteriori informazioni, incluse le linee guida per la denominazione delle chiavi degli oggetti, consulta [Using versioning](#) nella Amazon Simple Storage Service User Guide.

Note

Nella console, la proprietà Docker Compose file in S3 (File di Docker Compose in S3) combina i parametri DockerComposeFileS3Bucket, DockerComposeFileS3Key e DockerComposeFileS3Version.

Richiesto: false

Tipo: string

Schema valido . +

DockerComposeFileDestinationPath

Il percorso assoluto della directory locale utilizzata per archiviare una copia del file di Docker Compose. Deve essere una directory esistente. L'utente specificato per `DockerUserId` deve disporre dell'autorizzazione per creare un file in questa directory. Per ulteriori informazioni, consulta [the section called “Configurazione dell'utente Docker sul core”](#).

Important

Questa directory archivia il file di Docker Compose e le credenziali per le immagini Docker di repository privati. È responsabilità dell'utente proteggere questa directory. Per ulteriori informazioni, consulta [the section called “Note sulla sicurezza”](#).

Nome visualizzato nella AWS IoT console: percorso della directory per il file Compose locale

Obbligatorio: true

Tipo: string

Schema valido `\/.*\`

Esempio: `/home/username/myCompose`

DockerUserId

L'UID dell'utente Linux con cui viene eseguito il connettore. Questo utente deve appartenere al gruppo Linux `docker` sul dispositivo core e disporre delle autorizzazioni in scrittura per la directory `DockerComposeFileDestinationPath`. Per ulteriori informazioni, consulta [Configurazione dell'utente Docker sul core](#).

Note

L'esecuzione come root deve essere utilizzata solo quando strettamente necessario. Se specifichi l'utente root, devi consentire alle funzioni Lambda di essere eseguite come root sul AWS IoT Greengrass core. Per ulteriori informazioni, consulta [the section called “Esecuzione di una funzione Lambda come utente root”](#).

Nome visualizzato nella AWS IoT console: ID utente Docker

Obbligatorio: `false`

Tipo: `string`

Schema valido: `^[0-9]{1,5}$`

`AWSecretsArnList`

I nomi delle risorse Amazon (ARNs) dei segreti AWS Secrets Manager che contengono le informazioni di accesso utilizzate per accedere alle immagini Docker in repository privati. Per ulteriori informazioni, consulta [the section called “Accesso alle immagini Docker da repository privati”](#).

Nome visualizzato nella AWS IoT console: Credentials for private repository

Obbligatorio: `false`. Questo parametro è obbligatorio per accedere alle immagini Docker archiviate nei repository privati.

Tipo: di array `string`

Schema valido: `[( ?,? ?"(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\\]+)/[a-zA-Z0-9/_+=, .@-]+-[a-zA-Z0-9]+)")]`

`DockerContainerStatusLogFrequency`

La frequenza (in secondi) di registrazione delle informazioni sullo stato relative ai container Docker in esecuzione sul core. Il valore predefinito è 300 secondi (5 minuti).

Nome visualizzato nella AWS IoT console: frequenza di registrazione

Obbligatorio: `false`

Tipo: `string`

Schema valido: `^[1-9]{1}[0-9]{0,3}$`

`ForceDeploy`

Indica se forzare la distribuzione di Docker se fallisce a causa della pulizia impropria dell'ultima distribuzione. Il valore predefinito è `False`.

Nome visualizzato nella AWS IoT console: Force deployment

Obbligatorio: `false`

Tipo: `string`

Schema valido: `^(true|false)$`

`DockerPullBeforeUp`

Indica se il deployer deve essere eseguito `docker-compose pull` prima di eseguire un `docker-compose up` determinato pull-down-up comportamento. Il valore predefinito è `True`.

Nome visualizzato nella AWS IoT console: Docker Pull Before Up

Obbligatorio: `false`

Tipo: `string`

Schema valido: `^(true|false)$`

`StopContainersOnNewDeployment`

Indica se il connettore deve interrompere i contenitori docker gestiti da Docker Deployer quando GGC viene interrotto (GGC si interrompe quando viene distribuito un nuovo gruppo o il kernel viene spento). Il valore predefinito è `True`.

Nome visualizzato nella console: Docker stop in caso di nuova distribuzione AWS IoT

Note

Consigliamo di mantenere questo parametro impostato sul `True` valore predefinito. Il parametro che `False` fa sì che il contenitore Docker continui a funzionare anche dopo l'interruzione del AWS IoT Greengrass core o l'avvio di una nuova distribuzione. Se imposti questo parametro su `False`, devi assicurarti che i contenitori Docker vengano mantenuti come necessario in caso di modifica o aggiunta del nome del `docker-compose` servizio.

Per ulteriori informazioni, consulta la documentazione del file di `docker-compose` composizione.

Obbligatorio: `false`

Tipo: `string`

Schema valido: `^(true|false)$`

DockerOfflineMode

Indica se utilizzare il file Docker Compose esistente all' AWS IoT Greengrass avvio offline. Il valore predefinito è `False`.

Obbligatorio: `false`

Tipo: `string`

Schema valido: `^(true|false)$`

Version 6

DockerComposeFileS3Bucket

Il nome del bucket S3 che contiene il file di Docker Compose. Quando crei il bucket, assicurati di seguire [le regole per i nomi dei bucket](#) descritte nella Amazon Simple Storage Service User Guide.

Nome visualizzato nella AWS IoT console: file Docker Compose in S3

Note

Nella console, la proprietà Docker Compose file in S3 (File di Docker Compose in S3) combina i parametri `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` e `DockerComposeFileS3Version`.

Obbligatorio: `true`

Tipo: `string`

Schema valido `[a-zA-Z0-9\\-\\.]{3,63}`

DockerComposeFileS3Key

La chiave oggetto per il tuo file Docker Compose in Amazon S3. Per ulteriori informazioni, incluse le linee guida per la denominazione delle chiavi degli oggetti, consulta [Chiave oggetto e metadati](#) nella Guida per l'utente di Amazon Simple Storage Service.

 Note

Nella console, la proprietà Docker Compose file in S3 (File di Docker Compose in S3) combina i parametri `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` e `DockerComposeFileS3Version`.

Richiesto: `true`

Tipo: `string`

Schema valido `.+`

`DockerComposeFileS3Version`

La versione dell'oggetto per il tuo file Docker Compose in Amazon S3. Per ulteriori informazioni, incluse le linee guida per la denominazione delle chiavi degli oggetti, consulta [Using versioning](#) nella Amazon Simple Storage Service User Guide.

 Note

Nella console, la proprietà Docker Compose file in S3 (File di Docker Compose in S3) combina i parametri `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` e `DockerComposeFileS3Version`.

Richiesto: `false`

Tipo: `string`

Schema valido `.+`

`DockerComposeFileDestinationPath`

Il percorso assoluto della directory locale utilizzata per archiviare una copia del file di Docker Compose. Deve essere una directory esistente. L'utente specificato per `DockerUserId` deve disporre dell'autorizzazione per creare un file in questa directory. Per ulteriori informazioni, consulta [the section called “Configurazione dell'utente Docker sul core”](#).

 Important

Questa directory archivia il file di Docker Compose e le credenziali per le immagini Docker di repository privati. È responsabilità dell'utente proteggere questa directory. Per ulteriori informazioni, consulta [the section called “Note sulla sicurezza”](#).

Nome visualizzato nella AWS IoT console: percorso della directory per il file Compose locale

Obbligatorio: `true`

Tipo: `string`

Schema valido `\/.*\/?`

Esempio: `/home/username/myCompose`

DockerUserId

L'UID dell'utente Linux con cui viene eseguito il connettore. Questo utente deve appartenere al gruppo Linux `docker` sul dispositivo core e disporre delle autorizzazioni in scrittura per la directory `DockerComposeFileDestinationPath`. Per ulteriori informazioni, consulta [Configurazione dell'utente Docker sul core](#).

 Note

L'esecuzione come root deve essere utilizzata solo quando strettamente necessario. Se specifichi l'utente root, devi consentire alle funzioni Lambda di essere eseguite come root sul AWS IoT Greengrass core. Per ulteriori informazioni, consulta [the section called “Esecuzione di una funzione Lambda come utente root”](#).

Nome visualizzato nella AWS IoT console: ID utente Docker

Obbligatorio: `false`

Tipo: `string`

Schema valido: `^[0-9]{1,5}$`

AWSecretsArnList

I nomi delle risorse Amazon (ARNs) dei segreti AWS Secrets Manager che contengono le informazioni di accesso utilizzate per accedere alle immagini Docker in repository privati. Per ulteriori informazioni, consulta [the section called “Accesso alle immagini Docker da repository privati”](#).

Nome visualizzato nella AWS IoT console: Credentials for private repository

Obbligatorio: false. Questo parametro è obbligatorio per accedere alle immagini Docker archiviate nei repository privati.

Tipo: di array string

Schema valido: `[( ?,? ?"(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\]+/)[a-zA-Z0-9/_+=, .@-]+-[a-zA-Z0-9]+)"]]`

DockerContainerStatusLogFrequency

La frequenza (in secondi) di registrazione delle informazioni sullo stato relative ai container Docker in esecuzione sul core. Il valore predefinito è 300 secondi (5 minuti).

Nome visualizzato nella AWS IoT console: frequenza di registrazione

Obbligatorio: false

Tipo: string

Schema valido: `^[1-9]{1}[0-9]{0,3}$`

ForceDeploy

Indica se forzare la distribuzione di Docker se fallisce a causa della pulizia impropria dell'ultima distribuzione. Il valore predefinito è False.

Nome visualizzato nella AWS IoT console: Force deployment

Obbligatorio: false

Tipo: string

Schema valido: `^(true|false)$`

DockerPullBeforeUp

Indica se il deployer deve essere eseguito `docker-compose pull` prima di eseguire un `docker-compose up` determinato pull-down-up comportamento. Il valore predefinito è `True`.

Nome visualizzato nella AWS IoT console: Docker Pull Before Up

Obbligatorio: `false`

Tipo: `string`

Schema valido: `^(true|false)$`

StopContainersOnNewDeployment

Indica se il connettore deve interrompere i contenitori docker gestiti da Docker Deployer quando GGC viene interrotto (quando viene effettuata una nuova distribuzione di gruppo o il kernel viene chiuso). Il valore predefinito è `True`.

Nome visualizzato nella console: Docker stop in caso di nuova distribuzione AWS IoT

Note

Consigliamo di mantenere questo parametro impostato sul `True` valore predefinito. Il parametro che `False` fa sì che il contenitore Docker continui a funzionare anche dopo l'interruzione del AWS IoT Greengrass core o l'avvio di una nuova distribuzione. Se imposti questo parametro su `False`, devi assicurarti che i contenitori Docker vengano mantenuti come necessario in caso di modifica o aggiunta del nome del `docker-compose` servizio.

Per ulteriori informazioni, consulta la documentazione del file di `docker-compose` composizione.

Obbligatorio: `false`

Tipo: `string`

Schema valido: `^(true|false)$`

Version 5

DockerComposeFileS3Bucket

Il nome del bucket S3 che contiene il file di Docker Compose. Quando crei il bucket, assicurati di seguire [le regole per i nomi dei bucket](#) descritte nella Amazon Simple Storage Service User Guide.

Nome visualizzato nella AWS IoT console: file Docker Compose in S3

Note

Nella console, la proprietà Docker Compose file in S3 (File di Docker Compose in S3) combina i parametri `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` e `DockerComposeFileS3Version`.

Obbligatorio: `true`

Tipo: `string`

Schema valido `[a-zA-Z0-9\\-\\.]{3,63}`

DockerComposeFileS3Key

La chiave oggetto per il tuo file Docker Compose in Amazon S3. Per ulteriori informazioni, incluse le linee guida per la denominazione delle chiavi degli oggetti, consulta [Chiave oggetto e metadati](#) nella Guida per l'utente di Amazon Simple Storage Service.

Note

Nella console, la proprietà Docker Compose file in S3 (File di Docker Compose in S3) combina i parametri `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` e `DockerComposeFileS3Version`.

Richiesto: `true`

Tipo: `string`

Schema valido `.+`

DockerComposeFileS3Version

La versione dell'oggetto per il tuo file Docker Compose in Amazon S3. Per ulteriori informazioni, incluse le linee guida per la denominazione delle chiavi degli oggetti, consulta [Using versioning](#) nella Amazon Simple Storage Service User Guide.

Note

Nella console, la proprietà Docker Compose file in S3 (File di Docker Compose in S3) combina i parametri `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` e `DockerComposeFileS3Version`.

Richiesto: false

Tipo: string

Schema valido .+

DockerComposeFileDestinationPath

Il percorso assoluto della directory locale utilizzata per archiviare una copia del file di Docker Compose. Deve essere una directory esistente. L'utente specificato per `DockerUserId` deve disporre dell'autorizzazione per creare un file in questa directory. Per ulteriori informazioni, consulta [the section called “Configurazione dell'utente Docker sul core”](#).

Important

Questa directory archivia il file di Docker Compose e le credenziali per le immagini Docker di repository privati. È responsabilità dell'utente proteggere questa directory. Per ulteriori informazioni, consulta [the section called “Note sulla sicurezza”](#).

Nome visualizzato nella AWS IoT console: percorso della directory per il file Compose locale

Obbligatorio: true

Tipo: string

Schema valido `\. * \?`

Esempio: `/home/username/myCompose`

`DockerUserId`

L'UID dell'utente Linux con cui viene eseguito il connettore. Questo utente deve appartenere al gruppo `Linux docker` sul dispositivo core e disporre delle autorizzazioni in scrittura per la directory `DockerComposeFileDestinationPath`. Per ulteriori informazioni, consulta [Configurazione dell'utente Docker sul core](#).

Note

L'esecuzione come root deve essere utilizzata solo quando strettamente necessario. Se specifichi l'utente root, devi consentire alle funzioni Lambda di essere eseguite come root sul AWS IoT Greengrass core. Per ulteriori informazioni, consulta [the section called “Esecuzione di una funzione Lambda come utente root”](#).

Nome visualizzato nella AWS IoT console: ID utente Docker

Obbligatorio: `false`

Tipo: `string`

Schema valido: `^[0-9]{1,5}$`

`AWSecretsArnList`

I nomi delle risorse Amazon (ARNs) dei segreti AWS Secrets Manager che contengono le informazioni di accesso utilizzate per accedere alle immagini Docker in repository privati. Per ulteriori informazioni, consulta [the section called “Accesso alle immagini Docker da repository privati”](#).

Nome visualizzato nella AWS IoT console: Credentials for private repository

Obbligatorio: `false`. Questo parametro è obbligatorio per accedere alle immagini Docker archiviate nei repository privati.

Tipo: `di array string`

Schema valido: `[ ( ?, ? ? "(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\+\/][a-zA-Z0-9/_+=, .@-]+-[a-zA-Z0-9]+))" ]`

DockerContainerStatusLogFrequency

La frequenza (in secondi) di registrazione delle informazioni sullo stato relative ai container Docker in esecuzione sul core. Il valore predefinito è 300 secondi (5 minuti).

Nome visualizzato nella AWS IoT console: frequenza di registrazione

Obbligatorio: false

Tipo: string

Schema valido: `^[1-9]{1}[0-9]{0,3}$`

ForceDeploy

Indica se forzare la distribuzione di Docker se fallisce a causa della pulizia impropria dell'ultima distribuzione. Il valore predefinito è False.

Nome visualizzato nella AWS IoT console: Force deployment

Obbligatorio: false

Tipo: string

Schema valido: `^(true|false)$`

DockerPullBeforeUp

Indica se il deployer deve essere eseguito `docker-compose pull` prima di eseguire un `docker-compose up` determinato pull-down-up comportamento. Il valore predefinito è True.

Nome visualizzato nella AWS IoT console: Docker Pull Before Up

Obbligatorio: false

Tipo: string

Schema valido: `^(true|false)$`

Versions 2 - 4

`DockerComposeFileS3Bucket`

Il nome del bucket S3 che contiene il file di Docker Compose. Quando crei il bucket, assicurati di seguire [le regole per i nomi dei bucket](#) descritte nella Amazon Simple Storage Service User Guide.

Nome visualizzato nella AWS IoT console: file Docker Compose in S3

 Note

Nella console, la proprietà Docker Compose file in S3 (File di Docker Compose in S3) combina i parametri `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` e `DockerComposeFileS3Version`.

Obbligatorio: `true`

Tipo: `string`

Schema valido `[a-zA-Z0-9\\-\\.]{3,63}`

`DockerComposeFileS3Key`

La chiave oggetto per il tuo file Docker Compose in Amazon S3. Per ulteriori informazioni, incluse le linee guida per la denominazione delle chiavi degli oggetti, consulta [Chiave oggetto e metadati](#) nella Guida per l'utente di Amazon Simple Storage Service.

 Note

Nella console, la proprietà Docker Compose file in S3 (File di Docker Compose in S3) combina i parametri `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` e `DockerComposeFileS3Version`.

Richiesto: `true`

Tipo: `string`

Schema valido `.+`

DockerComposeFileS3Version

La versione dell'oggetto per il tuo file Docker Compose in Amazon S3. Per ulteriori informazioni, incluse le linee guida per la denominazione delle chiavi degli oggetti, consulta [Using versioning](#) nella Amazon Simple Storage Service User Guide.

Note

Nella console, la proprietà Docker Compose file in S3 (File di Docker Compose in S3) combina i parametri `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` e `DockerComposeFileS3Version`.

Richiesto: false

Tipo: string

Schema valido .+

DockerComposeFileDestinationPath

Il percorso assoluto della directory locale utilizzata per archiviare una copia del file di Docker Compose. Deve essere una directory esistente. L'utente specificato per `DockerUserId` deve disporre dell'autorizzazione per creare un file in questa directory. Per ulteriori informazioni, consulta [the section called “Configurazione dell'utente Docker sul core”](#).

Important

Questa directory archivia il file di Docker Compose e le credenziali per le immagini Docker di repository privati. È responsabilità dell'utente proteggere questa directory. Per ulteriori informazioni, consulta [the section called “Note sulla sicurezza”](#).

Nome visualizzato nella AWS IoT console: percorso della directory per il file Compose locale

Obbligatorio: true

Tipo: string

Schema valido `\\.*\\?`

Esempio: `/home/username/myCompose`

`DockerUserId`

L'UID dell'utente Linux con cui viene eseguito il connettore. Questo utente deve appartenere al gruppo Linux `docker` sul dispositivo core e disporre delle autorizzazioni in scrittura per la directory `DockerComposeFileDestinationPath`. Per ulteriori informazioni, consulta [Configurazione dell'utente Docker sul core](#).

Note

L'esecuzione come root deve essere utilizzata solo quando strettamente necessario. Se specifichi l'utente root, devi consentire alle funzioni Lambda di essere eseguite come root sul AWS IoT Greengrass core. Per ulteriori informazioni, consulta [the section called "Esecuzione di una funzione Lambda come utente root"](#).

Nome visualizzato nella AWS IoT console: ID utente Docker

Obbligatorio: `false`

Tipo: `string`

Schema valido: `^[0-9]{1,5}$`

`AWSSecretsArnList`

I nomi delle risorse Amazon (ARNs) dei segreti AWS Secrets Manager che contengono le informazioni di accesso utilizzate per accedere alle immagini Docker in repository privati. Per ulteriori informazioni, consulta [the section called "Accesso alle immagini Docker da repository privati"](#).

Nome visualizzato nella AWS IoT console: Credentials for private repository

Obbligatorio: `false`. Questo parametro è obbligatorio per accedere alle immagini Docker archiviate nei repository privati.

Tipo: `di array string`

Schema valido: `[ ( ?, ? ? "(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\+\/][a-zA-Z0-9/_+=, .@-]+-[a-zA-Z0-9]+) )" ) ]`

DockerContainerStatusLogFrequency

La frequenza (in secondi) di registrazione delle informazioni sullo stato relative ai container Docker in esecuzione sul core. Il valore predefinito è 300 secondi (5 minuti).

Nome visualizzato nella AWS IoT console: frequenza di registrazione

Obbligatorio: false

Tipo: string

Schema valido: `^[1-9]{1}[0-9]{0,3}$`

ForceDeploy

Indica se forzare la distribuzione di Docker se fallisce a causa della pulizia impropria dell'ultima distribuzione. Il valore predefinito è False.

Nome visualizzato nella AWS IoT console: Force deployment

Obbligatorio: false

Tipo: string

Schema valido: `^(true|false)$`

Version 1

DockerComposeFileS3Bucket

Il nome del bucket S3 che contiene il file di Docker Compose. Quando crei il bucket, assicurati di seguire [le regole per i nomi dei bucket](#) descritte nella Amazon Simple Storage Service User Guide.

Nome visualizzato nella AWS IoT console: file Docker Compose in S3

Note

Nella console, la proprietà Docker Compose file in S3 (File di Docker Compose in S3) combina i parametri `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` e `DockerComposeFileS3Version`.

Obbligatorio: true

Tipo: string

Schema valido [a-zA-Z0-9\\-\\.]{3,63}

DockerComposeFileS3Key

La chiave oggetto per il tuo file Docker Compose in Amazon S3. Per ulteriori informazioni, incluse le linee guida per la denominazione delle chiavi degli oggetti, consulta [Chiave oggetto e metadati](#) nella Guida per l'utente di Amazon Simple Storage Service.

Note

Nella console, la proprietà Docker Compose file in S3 (File di Docker Compose in S3) combina i parametri DockerComposeFileS3Bucket, DockerComposeFileS3Key e DockerComposeFileS3Version.

Obbligatorio: true

Tipo: string

Schema valido .+

DockerComposeFileS3Version

La versione dell'oggetto per il tuo file Docker Compose in Amazon S3. Per ulteriori informazioni, incluse le linee guida per la denominazione delle chiavi degli oggetti, consulta [Using versioning](#) nella Amazon Simple Storage Service User Guide.

Note

Nella console, la proprietà Docker Compose file in S3 (File di Docker Compose in S3) combina i parametri DockerComposeFileS3Bucket, DockerComposeFileS3Key e DockerComposeFileS3Version.

Richiesto: false

Tipo: string

Schema valido .+

DockerComposeFileDestinationPath

Il percorso assoluto della directory locale utilizzata per archiviare una copia del file di Docker Compose. Deve essere una directory esistente. L'utente specificato per `DockerUserId` deve disporre dell'autorizzazione per creare un file in questa directory. Per ulteriori informazioni, consulta [the section called “Configurazione dell'utente Docker sul core”](#).

Important

Questa directory archivia il file di Docker Compose e le credenziali per le immagini Docker di repository privati. È responsabilità dell'utente proteggere questa directory. Per ulteriori informazioni, consulta [the section called “Note sulla sicurezza”](#).

Nome visualizzato nella AWS IoT console: percorso della directory per il file Compose locale

Obbligatorio: `true`

Tipo: `string`

Schema valido `\. * \/?`

Esempio: `/home/username/myCompose`

DockerUserId

L'UID dell'utente Linux con cui viene eseguito il connettore. Questo utente deve appartenere al gruppo Linux `docker` sul dispositivo core e disporre delle autorizzazioni in scrittura per la directory `DockerComposeFileDestinationPath`. Per ulteriori informazioni, consulta [Configurazione dell'utente Docker sul core](#).

Note

L'esecuzione come `root` deve essere utilizzata solo quando strettamente necessario. Se specifichi l'utente `root`, devi consentire alle funzioni Lambda di essere eseguite come `root` sul AWS IoT Greengrass core. Per ulteriori informazioni, consulta [the section called “Esecuzione di una funzione Lambda come utente root”](#).

Nome visualizzato nella AWS IoT console: ID utente Docker

Obbligatorio: `false`

Tipo: string

Schema valido: `^[0-9]{1,5}$`

AWSecretsArnList

I nomi delle risorse Amazon (ARNs) dei segreti AWS Secrets Manager che contengono le informazioni di accesso utilizzate per accedere alle immagini Docker in repository privati. Per ulteriori informazioni, consulta [the section called “Accesso alle immagini Docker da repository privati”](#).

Nome visualizzato nella AWS IoT console: Credentials for private repository

Obbligatorio: false. Questo parametro è obbligatorio per accedere alle immagini Docker archiviate nei repository privati.

Tipo: di array string

Schema valido: `[( ?, ? ?"(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\\]+)/[a-zA-Z0-9/_+=, .@-]+-[a-zA-Z0-9]+)"]`

DockerContainerStatusLogFrequency

La frequenza (in secondi) di registrazione delle informazioni sullo stato relative ai container Docker in esecuzione sul core. Il valore predefinito è 300 secondi (5 minuti).

Nome visualizzato nella AWS IoT console: frequenza di registrazione

Obbligatorio: false

Tipo: string

Schema valido: `^[1-9]{1}[0-9]{0,3}$`

Esempio di creazione di un connettore (AWS CLI)

Il seguente comando CLI crea una `ConnectorDefinition` versione iniziale che contiene il connettore di distribuzione dell'applicazione Greengrass Docker.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
  "Connectors": [
    {
      "Id": "MyDockerAppplicationDeploymentConnector",
```

```

    "ConnectorArn": "arn:aws:greengrass:region::/connectors/
    DockerApplicationDeployment/versions/5",
    "Parameters": {
        "DockerComposeFileS3Bucket": "amzn-s3-demo-bucket",
        "DockerComposeFileS3Key": "production-docker-compose.yml",
        "DockerComposeFileS3Version": "123",
        "DockerComposeFileDestinationPath": "/home/username/myCompose",
        "DockerUserId": "1000",
        "AWSSecretsArnList": "[\"arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret1-hash\", \"arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret2-hash\"]",
        "DockerContainerStatusLogFrequency": "30",
        "ForceDeploy": "True",
        "DockerPullBeforeUp": "True"
    }
}
]
}'

```

Note

La funzione Lambda in questo connettore ha un ciclo di vita di [lunga durata](#).

Dati di input

Questo connettore non richiede né accetta dati di input.

Dati di output

Questo connettore pubblica lo stato del comando `docker-compose up` come dati di output.

Filtro argomento in sottoscrizione

`dockerapplicationdeploymentconnector/message/status`

Output di esempio: Operazione riuscita

```

{
  "status": "success",
  "GreengrassDockerApplicationDeploymentStatus": "Successfully triggered docker-
compose up",
  "S3Bucket": "amzn-s3-demo-bucket",

```

```
"ComposeFileName": "production-docker-compose.yml",  
"ComposeFileVersion": "123"  
}
```

Esempio di output: Errore

```
{  
  "status": "fail",  
  "error_message": "description of error",  
  "error": "InvalidParameter"  
}
```

Il tipo di errore può essere `InvalidParameter` o `InternalError`.

Configurazione dell'utente Docker sul core AWS IoT Greengrass

Il connettore di distribuzione dell'applicazione Greengrass Docker viene eseguito come utente specificato per il parametro `DockerUserId`. Se non specifichi un valore, il connettore viene eseguito come `ggc_user`, che è l'identità di accesso Greengrass predefinita.

Per consentire al connettore di interagire con il daemon Docker, l'utente Docker deve appartenere al gruppo Linux `docker` sul core. L'utente Docker deve inoltre disporre delle autorizzazioni in scrittura per la directory `DockerComposeFileDestinationPath`. Questa è la posizione in cui il connettore archivia il file `docker-compose.yml` locale e le credenziali Docker.

Note

- Ti consigliamo di creare un utente Linux anziché utilizzare l'impostazione predefinita `ggc_user`. Altrimenti, qualsiasi funzione Lambda nel gruppo Greengrass può accedere al file Compose e alle credenziali Docker.
- L'esecuzione come root deve essere utilizzata solo quando strettamente necessario. Se specifichi l'utente root, devi consentire alle funzioni Lambda di essere eseguite come root sul AWS IoT Greengrass core. Per ulteriori informazioni, consulta [the section called "Esecuzione di una funzione Lambda come utente root"](#).

1. Creare l'utente. È possibile eseguire il comando `useradd` e includere l'opzione `-u` facoltativa per assegnare un UID. Per esempio:

```
sudo useradd -u 1234 user-name
```

2. Aggiungere l'utente al gruppo `docker` sul core. Per esempio:

```
sudo usermod -aG docker user-name
```

Per ulteriori informazioni, inclusa la modalità di creazione del gruppo `docker`, consulta [Manage Docker as a non-root user](#) nella documentazione di Docker.

3. Concedere le autorizzazioni utente per scrivere nella directory specificata per il parametro `DockerComposeFileDestinationPath`. Per esempio:
 - a. Per impostare l'utente come il proprietario della directory. Questo esempio utilizza l'UID della fase 1.

```
chown 1234 docker-compose-file-destination-path
```

- b. Per fornire le autorizzazioni in lettura e scrittura al proprietario.

```
chmod 700 docker-compose-file-destination-path
```

Per ulteriori informazioni, consulta [How To Manage File And Folder Permissions In Linux](#) nella documentazione di Linux Foundation.

- c. Se non è stato assegnato un UID al momento della creazione dell'utente o se è stato utilizzato un utente esistente, eseguire il comando `id` per cercare l'UID.

```
id -u user-name
```

Utilizzare l'UID per configurare il parametro `DockerUserId` per il connettore.

Informazioni di utilizzo

Quando si utilizza il connettore di distribuzione delle applicazioni Greengrass Docker, è necessario conoscere le seguenti informazioni sull'utilizzo specifiche dell'implementazione.

- Prefisso per i nomi dei progetti. Il connettore antepone il prefisso `greengrassdockerapplicationdeployment` ai nomi dei container Docker che avvia. Il

connettore utilizza questo prefisso come il nome del progetto nei comandi `docker-compose` che esegue.

- Comportamento di registrazione. Il connettore scrive informazioni sullo stato e informazioni sulla risoluzione dei problemi in un file di log. È possibile configurare AWS IoT Greengrass l'invio dei log a CloudWatch Logs e la scrittura dei log localmente. Per ulteriori informazioni, consulta [the section called "Registrazione"](#). Questo è il percorso al log locale per il connettore:

```
/greengrass-root/ggc/var/log/user/region/aws/DockerApplicationDeployment.log
```

Devi disporre delle autorizzazioni root per accedere ai log locali.

- Aggiornamento delle immagini Docker. Docker memorizza le immagini nella cache sul dispositivo core. Se aggiorni un'immagine Docker e desideri propagare la modifica al dispositivo core, assicurati di modificare il tag per l'immagine nel file Compose. Le modifiche diventano effettive dopo la distribuzione del gruppo Greengrass.
- Timeout di 10 minuti per le operazioni di pulizia. Quando il demone Greengrass si ferma durante un riavvio, il `docker-compose down` comando viene avviato. Tutti i contenitori Docker dispongono di un massimo di 10 minuti dopo l'avvio per eseguire qualsiasi `docker-compose down` operazione di pulizia. Se la pulizia non viene completata entro 10 minuti, devi pulire manualmente i contenitori rimanenti. Per ulteriori informazioni, consulta [docker rm](#) nella documentazione dell'interfaccia a riga di comandi di Docker.
- Esecuzione di comandi Docker. Per risolvere i problemi, puoi eseguire i comandi Docker in una finestra del terminale sul dispositivo core. Ad esempio, esegui il comando seguente per visualizzare i container Docker avviati dal connettore:

```
docker ps --filter name="greengrassdockerapplicationdeployment"
```

- ID di risorsa riservato. Il connettore utilizza l'ID `DOCKER_DEPLOYER_SECRET_RESOURCE_RESERVED_ID_index` per le risorse Greengrass create nel gruppo Greengrass. La risorsa IDs deve essere univoca nel gruppo, quindi non assegnate un ID di risorsa che potrebbe essere in conflitto con questo ID di risorsa riservato.
- Modalità offline. Quando si imposta il parametro di `DockerOfflineMode` configurazione su `True`, il connettore Docker è in grado di funzionare in modalità offline. Ciò può accadere quando una distribuzione di gruppo Greengrass si riavvia mentre il dispositivo principale è offline e il connettore non riesce a stabilire una connessione ad Amazon S3 o Amazon ECR per recuperare il file Docker Compose.

Con la modalità offline abilitata, il connettore tenta di scaricare il file Compose ed eseguire i `docker login` comandi come farebbe per un normale riavvio. Se questi tentativi falliscono, il connettore cerca un file Compose archiviato localmente nella cartella specificata utilizzando il `DockerComposeFileDestinationPath` parametro. Se esiste un file Compose locale, il connettore segue la normale sequenza di `docker-compose` comandi ed estrae immagini locali. Se il file Compose o le immagini locali non sono presenti, il connettore si guasta. Il comportamento dei `StopContainersOnNewDeployment` parametri `ForceDeploy` and rimane lo stesso in modalità offline.

Comunicazione con i container Docker

AWS IoT Greengrass supporta i seguenti canali di comunicazione tra i componenti Greengrass e i contenitori Docker:

- Le funzioni Greengrass Lambda possono utilizzare REST APIs per comunicare con i processi nei contenitori Docker. È possibile configurare un server in un contenitore Docker che apre una porta. Le funzioni Lambda possono comunicare con il contenitore su questa porta.
- I processi nei container Docker possono scambiare messaggi MQTT tramite il broker messaggi Greengrass locale. È possibile configurare il contenitore Docker come dispositivo client nel gruppo Greengrass e quindi creare abbonamenti per consentire al contenitore di comunicare con le funzioni Greengrass Lambda, i dispositivi client e altri connettori del gruppo o con e il servizio shadow locale. AWS IoT Per ulteriori informazioni, consulta [the section called “Configurazione della comunicazione MQTT con i container Docker”](#).
- Le funzioni Greengrass Lambda possono aggiornare un file condiviso per passare informazioni ai contenitori Docker. Puoi utilizzare il file Compose per eseguire il montaggio vincolato del percorso file condiviso per un container Docker.

Configurazione della comunicazione MQTT con i container Docker

Puoi configurare un contenitore Docker come dispositivo client e aggiungerlo a un gruppo Greengrass. Quindi, puoi creare sottoscrizioni che consentono la comunicazione MQTT tra il container Docker e i componenti Greengrass o AWS IoT. Nella procedura seguente, viene creata una sottoscrizione che consente al dispositivo del container Docker di ricevere messaggi di aggiornamento shadow dal servizio shadow locale. Puoi seguire questo modello per creare altre sottoscrizioni.

 Note

Questa procedura presuppone che abbiate già creato un gruppo Greengrass e un core Greengrass (v1.10 o successivo). Per informazioni sulla creazione di un gruppo e di un core Greengrass, vedere. [Iniziare con AWS IoT Greengrass](#)

Per configurare un contenitore Docker come dispositivo client e aggiungerlo a un gruppo Greengrass

1. Crea una cartella sul dispositivo principale per archiviare i certificati e le chiavi utilizzati per autenticare il dispositivo Greengrass.

Il percorso del file deve essere montato sul container Docker che si desidera avviare. Il frammento di codice seguente mostra come montare un percorso del file nel file Compose. In questo esempio, *path-to-device-certs* rappresenta la cartella creata in questo passaggio.

```
version: '3.3'
services:
  myService:
    image: user-name/repo:image-tag
    volumes:
      - /path-to-device-certs/:/path-accessible-in-container
```

2. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).
3. Scegliere il gruppo target.
4. Nella pagina di configurazione del gruppo, scegli Dispositivi client, quindi scegli Associa.
5. Nella finestra Modale Associa un dispositivo client a questo gruppo, scegli Crea nuovo AWS IoT elemento.

La pagina Crea oggetti si apre in una nuova scheda.

6. Nella pagina Crea elementi, scegli Crea elemento singolo, quindi scegli Avanti.
7. Nella pagina Specificare le proprietà dell'oggetto, inserisci un nome per il dispositivo, quindi scegli Avanti.
8. Nella pagina Configura il certificato del dispositivo, scegli Avanti.
9. Nella pagina Allega le politiche al certificato, esegui una delle seguenti operazioni:

- Seleziona una politica esistente che conceda le autorizzazioni richieste dai dispositivi client, quindi scegli Crea oggetto.

Si apre una finestra modale in cui è possibile scaricare i certificati e le chiavi utilizzati dal dispositivo per connettersi al core Cloud AWS .

- Crea e allega una nuova policy che conceda le autorizzazioni per i dispositivi client. Esegui questa operazione:
 - a. Scegliere Create Policy (Crea policy).

La pagina Create policy (Crea policy) viene aperta in una nuova scheda.

- b. Nella pagina Create policy (Crea policy), eseguire le operazioni seguenti:
 - i. Per Nome della politica, inserisci un nome che descriva la politica, ad esempio. **GreengrassV1ClientDevicePolicy**
 - ii. Nella scheda Dichiarazioni politiche, in Documento di politica, scegli JSON.
 - iii. Inserisci il seguente documento di policy. Questa politica consente al dispositivo client di scoprire i core Greengrass e comunicare su tutti gli argomenti MQTT. Per informazioni su come limitare l'accesso a questa politica, vedere. [Autenticazione e autorizzazione del dispositivo per AWS IoT Greengrass](#)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Subscribe",
        "iot:Connect",
        "iot:Receive"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "greengrass:*"
      ]
    }
  ]
}
```

```
    ],  
    "Resource": [  
        "*"   
    ]  
  }  
]  
}
```

- iv. Scegliere Crea per creare la policy.
- c. Torna alla scheda del browser con la pagina *Allega politiche al certificato* aperta. Esegui questa operazione:
 - i. Nell'elenco *Politiche*, seleziona la politica che hai creato, ad esempio *GreengrassV1ClientDevicePolicy*.

Se non vedi la politica, scegli il pulsante di aggiornamento.

- ii. Scegli *Create thing* (Crea oggetto).

Si apre una finestra modale in cui puoi scaricare i certificati e le chiavi che il dispositivo utilizza per connettersi al Cloud AWS core.

- 10. Nella modalità *Scarica certificati e chiavi*, scarica i certificati del dispositivo.

 Important

Prima di scegliere *Fine*, scarica le risorse di sicurezza.

Esegui questa operazione:

- a. Per *Certificato del dispositivo*, scegli *Scarica* per scaricare il certificato del dispositivo.
- b. Per il file della chiave pubblica, scegli *Scarica* per scaricare la chiave pubblica per il certificato.
- c. Per *File di chiave privata*, scegli *Scarica* per scaricare il file di chiave privata per il certificato.
- d. Consulta [l'autenticazione del server](#) nella Guida per gli AWS IoT sviluppatori e scegli il certificato CA principale appropriato. Ti consigliamo di utilizzare gli endpoint Amazon Trust Services (ATS) e i certificati CA root ATS. In *Certificati CA root*, scegli *Scarica* per un certificato CA root.
- e. Seleziona *Fatto*.

Prendi nota dell'ID del certificato che è comune nei nomi dei file per il certificato e le chiavi del dispositivo. perché sarà necessaria in seguito.

11. Copia i certificati e le chiavi nella cartella creata nel passaggio 1.

Quindi, creare una sottoscrizione nel gruppo. Per questo esempio, creare una sottoscrizione che consente al dispositivo container Docker di ricevere messaggi MQTT dal servizio shadow locale.

Note

La dimensione massima di un documento shadow è di 8 KB. Per ulteriori informazioni, consulta [Quote di AWS IoT](#) nella Guida per gli sviluppatori di AWS IoT .

Per creare una sottoscrizione che consente al dispositivo container Docker di ricevere messaggi MQTT dal servizio shadow locale

1. Nella pagina di configurazione del gruppo, scegli la scheda Abbonamenti, quindi scegli Aggiungi abbonamento.
2. Nella pagina Select your source and target (Seleziona origine e destinazione), configura l'origine e la destinazione come indicato di seguito:
 - a. Per Select a source (Seleziona un'origine), scegli Services (Servizi), quindi scegli Local Shadow Service (Servizio shadow locale).
 - b. In Select a target (Seleziona una destinazione), scegliere Devices (Dispositivi), quindi selezionare il dispositivo.
 - c. Scegli Next (Successivo).
 - d. Nella pagina Filtra i dati con un argomento, per Filtro argomento scegli **`$aws/things/MyDockerDevice/shadow/update/accepted`**, quindi scegli Avanti.
Sostituiscilo **`MyDockerDevice`** con il nome del dispositivo che hai creato in precedenza.
 - e. Scegli Fine.

Includi il frammento di codice seguente nell'immagine Docker cui fai riferimento nel file Compose. Questo è il codice del dispositivo Greengrass. Inoltre, aggiungi il codice nel container Docker che avvia il dispositivo Greengrass all'interno del container. Può essere eseguito come un processo separato nell'immagine o in un thread separato.

```
import os
import sys
import time
import uuid

from AWSIoTPythonSDK.core.greengrass.discovery.providers import DiscoveryInfoProvider
from AWSIoTPythonSDK.exception.AWSIoTExceptions import DiscoveryInvalidRequestException
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient

# Replace thingName with the name you registered for the Docker device.
thingName = "MyDockerDevice"
clientId = thingName

# Replace host with the IoT endpoint for your &AWS-account;.
host = "myPrefix.iot.region.amazonaws.com"

# Replace topic with the topic where the Docker container subscribes.
topic = "$aws/things/MyDockerDevice/shadow/update/accepted"

# Replace these paths based on the download location of the certificates for the Docker
  container.
rootCAPath = "/path-accessible-in-container/AmazonRootCA1.pem"
certificatePath = "/path-accessible-in-container/certId-certificate.pem.crt"
privateKeyPath = "/path-accessible-in-container/certId-private.pem.key"

# Discover Greengrass cores.
discoveryInfoProvider = DiscoveryInfoProvider()
discoveryInfoProvider.configureEndpoint(host)
discoveryInfoProvider.configureCredentials(rootCAPath, certificatePath, privateKeyPath)
discoveryInfoProvider.configureTimeout(10) # 10 seconds.

GROUP_CA_PATH = "./groupCA/"
MQTT_QOS = 1

discovered = False
groupCA = None
coreInfo = None

try:
    # Get discovery info from AWS IoT.
    discoveryInfo = discoveryInfoProvider.discover(thingName)
    caList = discoveryInfo.getAllCas()
```

```
coreList = discoveryInfo.getAllCores()

# Use first discovery result.
groupId, ca = caList[0]
coreInfo = coreList[0]

# Save the group CA to a local file.
groupCA = GROUP_CA_PATH + groupId + "_CA_" + str(uuid.uuid4()) + ".crt"
if not os.path.exists(GROUP_CA_PATH):
    os.makedirs(GROUP_CA_PATH)
groupCAFile = open(groupCA, "w")
groupCAFile.write(ca)
groupCAFile.close()
discovered = True
except DiscoveryInvalidRequestException as e:
    print("Invalid discovery request detected!")
    print("Type: %s" % str(type(e)))
    print("Error message: %s" % str(e))
    print("Stopping...")
except BaseException as e:
    print("Error in discovery!")
    print("Type: %s" % str(type(e)))
    print("Error message: %s" % str(e))
    print("Stopping...")

myAWSIoTMQTTClient = AWSIoTMQTTClient(clientId)
myAWSIoTMQTTClient.configureCredentials(groupCA, privateKeyPath, certificatePath)

# Try to connect to the Greengrass core.
connected = False
for connectivityInfo in coreInfo.connectivityInfoList:
    currentHost = connectivityInfo.host
    currentPort = connectivityInfo.port
    myAWSIoTMQTTClient.configureEndpoint(currentHost, currentPort)
    try:
        myAWSIoTMQTTClient.connect()
        connected = True
    except BaseException as e:
        print("Error in connect!")
        print("Type: %s" % str(type(e)))
        print("Error message: %s" % str(e))
if connected:
    break
```

```
if not connected:
    print("Cannot connect to core %s. Exiting..." % coreInfo.coreThingArn)
    sys.exit(-2)

# Handle the MQTT message received from GGShadowService.
def customCallback(client, userdata, message):
    print("Received an MQTT message")
    print(message)

# Subscribe to the MQTT topic.
myAWSIoTMQTTClient.subscribe(topic, MQTT_QOS, customCallback)

# Keep the process alive to listen for messages.
while True:
    time.sleep(1)
```

Note sulla sicurezza

Quando utilizzi il connettore di distribuzione delle applicazioni Greengrass Docker, tieni presente le seguenti considerazioni sulla sicurezza.

Storage locale del file di Docker Compose

Il connettore archivia una copia del file Compose nella directory specificata per il parametro `DockerComposeFileDestinationPath`.

È responsabilità dell'utente proteggere questa directory. Utilizzare le autorizzazioni del file system per limitare l'accesso alla directory.

Storage locale delle credenziali Docker

Se le immagini Docker vengono archiviate in repository privati, il connettore archivia le credenziali Docker nella directory specificata per il parametro `DockerComposeFileDestinationPath`.

È responsabilità dell'utente proteggere queste credenziali. Ad esempio, utilizzare [credential-helper](#) sul dispositivo core quando si installa Docker Engine.

Installazione di Docker Engine da un'origine attendibile

È responsabilità dell'utente installare Docker Engine da un'origine attendibile. Questo connettore utilizza il daemon Docker sul dispositivo core per accedere agli asset Docker e gestire i container Docker.

Ambito delle autorizzazioni del ruolo del gruppo Greengrass

Le autorizzazioni aggiunte al ruolo di gruppo Greengrass possono essere assunte da tutte le funzioni e i connettori Lambda del gruppo Greengrass. Questo connettore richiede l'accesso al file di Docker Compose archiviato in un bucket S3. Richiede inoltre l'accesso al token di autorizzazione Amazon ECR se le immagini Docker sono archiviate in un repository privato in Amazon ECR.

Licenze

Il connettore di distribuzione delle applicazioni Greengrass Docker include i seguenti software/licenze di terze parti:

- [AWS SDK per Python \(Boto3\)](#)/Apache License 2.0
- [botocore](#)/Apache License 2.0
- [dateutil](#)/PSF License
- [docutils](#)/BSD License, GNU General Public License (GPL), Python Software Foundation License, Public Domain
- [jmespath](#)/MIT License
- [s3transfer](#)/Apache License 2.0
- [urllib3](#)/MIT License

Questo connettore è rilasciato ai sensi del contratto di [licenza del software Greengrass Core](#).

Changelog

La tabella seguente descrive le modifiche apportate a ciascuna versione del connettore.

Versione	Modifiche
7	DockerOfflineMode Aggiunto per utilizzare un file Docker Compose esistente all' AWS IoT Greengrass avvio offline. Sono stati implementati nuovi tentativi per il comando. docker login Support per 32 bit UIDs.

Versione	Modifiche
6	Aggiunto <code>StopContainersOnNewDeployment</code> per ignorare la pulizia del contenitore quando viene effettuata una nuova implementazione o GGC si interrompe. Meccanismi di spegnimento e avvio più sicuri. Correzione di bug di convalida YAML.
5	Le immagini vengono estratte prima dell'esecuzione. <code>docker-compose down</code>
4	Aggiunto <code>pull-before-up</code> un comportamento per aggiornare le immagini Docker.
3	Risolto un problema con la ricerca di variabili d'ambiente.
2	Aggiunto il parametro <code>ForceDeploy</code> .
1	Versione iniziale.

Un gruppo Greengrass può contenere una sola versione del connettore alla volta. Per informazioni sull'aggiornamento di una versione del connettore, consulta [the section called “Aggiornamento delle versioni dei connettori”](#).

Consulta anche

- [Integrazione con servizi e protocolli tramite i connettori](#)
- [the section called “Nozioni di base sui connettori \(console\)”](#)
- [the section called “Nozioni di base sui connettori \(CLI\)”](#)

Connettore IoT Analytics

Warning

Questo connettore è entrato nella fase di vita prolungata e AWS IoT Greengrass non rilascerà aggiornamenti che forniscano funzionalità, miglioramenti alle funzionalità esistenti, patch di sicurezza o correzioni di bug. Per ulteriori informazioni, consulta [AWS IoT Greengrass Version 1 politica di manutenzione](#).

Il connettore IoT Analytics invia i dati del dispositivo locale a AWS IoT Analytics. È possibile utilizzare questo connettore come hub centrale per raccogliere dati dai sensori sul dispositivo principale Greengrass e dai dispositivi [client collegati](#). Il connettore invia i dati ai AWS IoT Analytics canali della regione corrente Account AWS e della regione. Può inviare i dati a un canale di destinazione predefinito e ai canali specificati in modo dinamico.

Note

AWS IoT Analytics è un servizio completamente gestito che consente di raccogliere, archiviare, elaborare e interrogare dati IoT. Nel AWS IoT Analytics, i dati possono essere ulteriormente analizzati ed elaborati. Ad esempio, possono essere utilizzati per dare forma ai modelli ML per il monitoraggio della salute macchina o per testare nuove strategie di modellazione. Per ulteriori informazioni, consulta [Cos'è AWS IoT Analytics?](#) nella Guida AWS IoT Analytics per l'utente.

Il connettore accetta i dati formattati e non formattati negli [argomenti MQTT di input](#). Supporta due argomenti predefiniti in cui il canale di destinazione viene specificato in linea. Può anche ricevere i messaggi su argomenti definiti dal cliente che vengono [configurati nelle sottoscrizioni](#). Questo può essere usato per indirizzare messaggi da dispositivi client che pubblicano su argomenti fissi o per gestire dati non strutturati o dipendenti dallo stack provenienti da dispositivi con risorse limitate.

Questo connettore utilizza l'[BatchPutMessage](#) API per inviare dati (come stringa con codifica JSON o base64) al canale di destinazione. Il connettore è in grado di eseguire i dati non elaborati in un formato che soddisfa i requisiti delle API. Il connettore memorizza nel buffer i messaggi nelle code per canale ed elabora in modo asincrono i batch. Offre i parametri che consentono di controllare l'accodamento e la creazione di comportamenti in batch, nonché di limitare il consumo della memoria. Ad esempio, è possibile configurare dimensioni massime della coda, intervallo in batch, dimensioni della memoria e numero di canali attivo.

Questo connettore ha le seguenti versioni.

Versione	ARN
4	<code>arn:aws:greengrass: <i>region</i> :/:connectors/IoTAnalytics/versions/4</code>

Versione	ARN
3	arn:aws:greengrass: <i>region</i> ::/connectors/IoTAnalytics/versions/3
2	arn:aws:greengrass: <i>region</i> ::/connectors/IoTAnalytics/versions/2
1	arn:aws:greengrass: <i>region</i> ::/connectors/IoTAnalytics/versions/1

Per informazioni sulle modifiche di ogni versione, consulta [Changelog](#).

Requisiti

Questo connettore presenta i seguenti requisiti:

Version 3 - 4

- AWS IoT Greengrass Software principale v1.9.3 o versione successiva.
- [Python](#) versione 3.7 o 3.8 installata sul dispositivo principale e aggiunta alla variabile di ambiente PATH.

Note

Per usare Python 3.8, esegui il seguente comando per creare un collegamento simbolico dalla cartella di installazione predefinita di Python 3.7 ai binari Python 3.8 installati.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Questo configura il dispositivo in modo che soddisfi il requisito Python per AWS IoT Greengrass.

- Questo connettore può essere utilizzato solo nelle regioni di Amazon Web Services in cui entrambi [AWS IoT Greengrass](#) e [AWS IoT Analytics](#) sono supportati.
- Tutte le AWS IoT Analytics entità e i flussi di lavoro correlati vengono creati e configurati. Le entità includono canali, pipeline, datastore e set di dati. Per ulteriori informazioni, consulta le procedure [AWS CLI](#) o [console](#) nella Guida per l'utente AWS IoT Analytics .

 Note

AWS IoT Analytics I canali di destinazione devono utilizzare lo stesso account e trovarsi nella Regione AWS stesso connettore.

- Il [ruolo del gruppo Greengrass](#) è configurato per consentire l'`iotanalytics:BatchPutMessage` azione sui canali di destinazione, come mostrato nel seguente esempio di politica IAM. I canali devono trovarsi nell'area corrente Account AWS e nella regione.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1528133056761",
      "Action": [
        "iotanalytics:BatchPutMessage"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:iotanalytics:region:account-id:channel/channel_1_name",
        "arn:aws:iotanalytics:region:account-id:channel/channel_2_name"
      ]
    }
  ]
}
```

Per il requisito del ruolo di gruppo, è necessario configurare il ruolo in modo da concedere le autorizzazioni necessarie e assicurarsi che il ruolo sia stato aggiunto al gruppo. Per ulteriori informazioni, consulta [the section called “Gestione del ruolo del gruppo \(console\)”](#) o [the section called “Gestire il ruolo del gruppo \(CLI\)”](#).

Versions 1 - 2

- AWS IoT Greengrass Software principale v1.7 o successivo.
- [Python](#) versione 2.7 installato sul dispositivo principale e aggiunto alla variabile di ambiente PATH.
- Questo connettore può essere utilizzato solo nelle regioni di Amazon Web Services in cui entrambi [AWS IoT Greengrass](#) e [AWS IoT Analytics](#) sono supportati.
- Tutte le AWS IoT Analytics entità e i flussi di lavoro correlati vengono creati e configurati. Le entità includono canali, pipeline, datastore e set di dati. Per ulteriori informazioni, consulta le procedure [AWS CLI](#) o [console](#) nella Guida per l'utente AWS IoT Analytics .

 Note

AWS IoT Analytics I canali di destinazione devono utilizzare lo stesso account e trovarsi nella Regione AWS stesso connettore.

- Il [ruolo del gruppo Greengrass](#) è configurato per consentire l'`iotanalytics:BatchPutMessage` azione sui canali di destinazione, come mostrato nel seguente esempio di politica IAM. I canali devono trovarsi nell'area corrente Account AWS e nella regione.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1528133056761",
      "Action": [
        "iotanalytics:BatchPutMessage"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:iotanalytics:region:account-id:channel/channel_1_name",
        "arn:aws:iotanalytics:region:account-id:channel/channel_2_name"
      ]
    }
  ]
}
```

Per il requisito del ruolo di gruppo, è necessario configurare il ruolo in modo da concedere le autorizzazioni necessarie e assicurarsi che il ruolo sia stato aggiunto al gruppo. Per ulteriori informazioni, consulta [the section called “Gestione del ruolo del gruppo \(console\)”](#) o [the section called “Gestire il ruolo del gruppo \(CLI\)”](#).

Parametri

MemorySize

La quantità di memoria (in KB) da allocare al connettore.

Nome visualizzato nella AWS IoT console: dimensione della memoria

Obbligatorio: true

Tipo: string

Schema valido: `^[0-9]+$`

PublishRegion

Il Regione AWS modo in cui vengono creati i tuoi AWS IoT Analytics canali. Utilizza la stessa regione del connettore.

Note

Questo deve inoltre corrispondere alla regione per i canali specificati nel [ruolo gruppo](#).

Nome visualizzato nella AWS IoT console: regione di pubblicazione

Obbligatorio: false

Tipo: string

Schema valido: `^$|([a-z]{2}-[a-z]+-\\d{1})`

PublishInterval

L'intervallo (in secondi) in cui pubblicare un batch di dati ricevuti. AWS IoT Analytics

Nome visualizzato nella AWS IoT console: Intervallo di pubblicazione

Obbligatorio: false

Tipo: string

Valore predefinito: 1

Schema valido: `$|^[0-9]+$`

`IotAnalyticsMaxActiveChannels`

Il numero massimo di AWS IoT Analytics canali che il connettore monitora attivamente. Questo deve essere maggiore di 0 e almeno pari al numero di canali su cui prevedi che il connettore pubblicherà in un determinato momento.

Puoi utilizzare questo parametro per limitare il consumo della memoria limitando il numero totale di code che il connettore è in grado di gestire in un determinato momento. Una coda viene eliminata quando tutti i messaggi in coda vengono inviati.

Nome visualizzato nella AWS IoT console: numero massimo di canali attivi

Obbligatorio: false

Tipo: string

Valore predefinito: 50

Schema valido: `^[1-9][0-9]*$`

`IotAnalyticsQueueDropBehavior`

Il comportamento per l'eliminazione di messaggi da una coda del canale quando la coda è piena.

Nome visualizzato nella AWS IoT console: comportamento di caduta dalla coda

Obbligatorio: false

Tipo: string

Valori validi: `DROP_NEWEST` o `DROP_OLDEST`

Valore predefinito: `DROP_NEWEST`

Schema valido: `^DROP_NEWEST$|^DROP_OLDEST$`

IotAnalyticsQueueSizePerChannel

Il numero massimo di messaggi da conservare in memoria (per canale) prima che vengano inviati o eliminati. Questo numero deve essere maggiore di 0.

Nome visualizzato nella AWS IoT console: dimensione massima della coda per canale

Obbligatorio: false

Tipo: string

Valore predefinito: 2048

Schema valido: `^[1-9][0-9]*$`

IotAnalyticsBatchSizePerChannel

Il numero massimo di messaggi da inviare a un AWS IoT Analytics canale in una richiesta batch. Questo numero deve essere maggiore di 0.

Nome visualizzato nella AWS IoT console: numero massimo di messaggi da raggruppare in batch per canale

Obbligatorio: false

Tipo: string

Valore predefinito: 5

Schema valido: `^[1-9][0-9]*$`

IotAnalyticsDefaultChannelName

Il nome del AWS IoT Analytics canale utilizzato da questo connettore per i messaggi inviati a un argomento di input definito dal cliente.

Nome visualizzato nella AWS IoT console: nome del canale predefinito

Obbligatorio: false

Tipo: string

Schema valido: `^[a-zA-Z0-9_]$`

IsolationMode

Modalità di [containerizzazione](#) per questo connettore. L'impostazione predefinita è `GreengrassContainer`, il che significa che il connettore viene eseguito in un ambiente di runtime isolato all'interno del AWS IoT Greengrass contenitore.

Note

L'impostazione predefinita della containerizzazione per il gruppo non si applica ai connettori.

Nome visualizzato nella AWS IoT console: modalità di isolamento del contenitore

Obbligatorio: `false`

Tipo: `string`

Valori validi: `GreengrassContainer` o `NoContainer`

Schema valido: `^NoContainer$|^GreengrassContainer$`

Esempio di creazione di un connettore (AWS CLI)

Il seguente comando CLI crea una `ConnectorDefinition` versione iniziale che contiene il connettore IoT Analytics.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-  
version '{  
  "Connectors": [  
    {  
      "Id": "MyIoTAnalyticsApplication",  
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/IoTAnalytics/  
versions/3",  
      "Parameters": {  
        "MemorySize": "65535",  
        "PublishRegion": "us-west-1",  
        "PublishInterval": "2",  
        "IotAnalyticsMaxActiveChannels": "25",  
        "IotAnalyticsQueueDropBehavior": "DROP_OLDEST",  
        "IotAnalyticsQueueSizePerChannel": "1028",  
        "IotAnalyticsBatchSizePerChannel": "5",
```

```
        "IotAnalyticsDefaultChannelName": "my_channel"
    }
}
]
```

Note

La funzione Lambda in questo connettore ha un ciclo di vita di [lunga durata](#).

Nella AWS IoT Greengrass console, puoi aggiungere un connettore dalla pagina Connettori del gruppo. Per ulteriori informazioni, consulta [the section called “Nozioni di base sui connettori \(console\)”](#).

Dati di input

Questo connettore accetta i dati su argomenti MQTT predefiniti e definiti dal cliente. Gli editori possono essere dispositivi client, funzioni Lambda o altri connettori.

Argomenti predefiniti

Il connettore supporta i due seguenti argomenti MQTT strutturati che consentono ai publisher di specificare il nome del canale in linea.

- Un [messaggio formattato](#) sull'argomento `iotanalytics/channels/+/messages/put`. I dati IoT in questi messaggi di input devono essere in formato JSON o una stringa con codifica base64.
- Un messaggio non formattato sull'argomento `iotanalytics/channels/+/messages/binary/put`. I messaggi di input ricevuti su questo argomento vengono trattati come dati binari e possono contenere qualsiasi tipo di dati.

Per pubblicare gli argomenti predefiniti, sostituire il carattere jolly + con il nome del canale. Per esempio:

```
iotanalytics/channels/my_channel/messages/put
```

Argomenti definiti dal cliente

Il connettore supporta la sintassi dell'argomento #, che consente di accettare messaggi di input su qualsiasi argomento MQTT che configuri in una sottoscrizione. Ti consigliamo di specificare il

percorso di un argomento invece di utilizzare solo il carattere # jolly nelle tue sottoscrizioni. Questi messaggi vengono inviati al canale predefinito specificato per il connettore.

I messaggi di input sugli argomenti definiti dal cliente vengono trattati come dati binari. Essi possono usare qualsiasi formato del messaggio e contenere qualsiasi tipo di dati. Puoi utilizzare gli argomenti definiti dal cliente per instradare i messaggi provenienti da dispositivi che pubblicano su argomenti fissi. È inoltre possibile utilizzarli per accettare dati di input da dispositivi client che non sono in grado di elaborare i dati in un messaggio formattato da inviare al connettore.

Per ulteriori informazioni sulle sottoscrizioni e sugli argomenti MQTT, consulta [the section called "Input e output"](#).

Il ruolo del gruppo consente l'operazione `iotanalytics:BatchPutMessage` su tutti i canali di destinazione. Per ulteriori informazioni, consulta [the section called "Requisiti"](#).

Filtro di argomenti: `iotanalytics/channels/+/messages/put`

Utilizzare questo argomento per inviare messaggi formattati al connettore e specificare in modo dinamico un canale di destinazione. Questo argomento consente inoltre di specificare un ID restituito nell'output della risposta. Il connettore verifica che sia IDs univoco per ogni messaggio della `BatchPutMessage` richiesta in uscita a cui viene inviato. AWS IoT Analytics Viene eliminato un messaggio con un ID duplicato.

I dati di input inviati a questo argomento devono utilizzare il seguente formato del messaggio.

Proprietà dei messaggi

`request`

I dati da inviare al canale specificato.

Obbligatorio: `true`

Tipo: `object` che include le seguenti proprietà:

`message`

Il dispositivo o i dati di sensori come JSON o stringa con codifica base64.

Obbligatorio: `true`

Tipo: `string`

id

Un ID arbitrario della richiesta. Questa proprietà viene utilizzata per associare una richiesta di input a una risposta di output. Quando specificato, la proprietà `id` nell'oggetto della risposta è impostata su questo valore. Se si omette questa proprietà, il connettore genera un ID.

Richiesto: `false`

Tipo: `string`

Schema valido: `. *`

Input di esempio

```
{
  "request": {
    "message" : "{\"temp\":23.33}"
  },
  "id" : "req123"
}
```

Filtro di argomenti: `iotanalytics/channels/+/messages/binary/put`

Utilizzare questo argomento per inviare messaggi non formattati al connettore e specificare in modo dinamico un canale di destinazione.

I dati del connettore non analizzano i messaggi di input ricevuti su questo argomento. Vengono trattati come dati binari. Prima di inviare i messaggi a AWS IoT Analytics, il connettore li codifica e li formatta per renderli conformi ai requisiti `BatchPutMessage` dell'API:

- Il connettore codifica su base64 i dati grezzi e include il payload codificato in una richiesta `BatchPutMessage` in uscita.
- Il connettore genera e assegna un ID per ogni messaggio di input.

Note

L'output della risposta del connettore non include una correlazione di ID per questi messaggi di input.

Proprietà dei messaggi

Nessuna.

Filtro di argomenti:

Utilizzare questo argomento per inviare qualsiasi formato del messaggio al canale predefinito. Ciò è particolarmente utile quando i dispositivi client pubblicano su argomenti fissi o quando si desidera inviare dati al canale predefinito da dispositivi client che non sono in grado di elaborare i dati nel formato di [messaggio supportato](#) dal connettore.

La sintassi dell'argomento viene definita nell'abbonamento creato per connettere questo connettore all'origine dati. Ti consigliamo di specificare il percorso di un argomento invece di utilizzare solo il carattere # jolly nelle tue sottoscrizioni.

I dati del connettore non analizzano i messaggi pubblicati su questo argomento di input. Tutti i messaggi di input vengono trattati come dati binari. Prima di inviare i messaggi a AWS IoT Analytics, il connettore li codifica e li formatta per renderli conformi ai requisiti dell'API: `BatchPutMessage`

- Il connettore codifica su base64 i dati grezzi e include il payload codificato in una richiesta `BatchPutMessage` in uscita.
- Il connettore genera e assegna un ID per ogni messaggio di input.



Note

L'output della risposta del connettore non include una correlazione di ID per questi messaggi di input.

Proprietà dei messaggi

Nessuna.

Dati di output

Questo connettore pubblica le informazioni di stato come dati di output su un argomento MQTT. Queste informazioni contengono la risposta restituita da AWS IoT Analytics per ogni messaggio di input ricevuto e inviato. AWS IoT Analytics

Filtro argomento in sottoscrizione

`iotanalytics/messages/put/status`

Output di esempio: Operazione riuscita

```
{
  "response" : {
    "status" : "success"
  },
  "id" : "req123"
}
```

Esempio di output: Errore

```
{
  "response" : {
    "status" : "fail",
    "error" : "ResourceNotFoundException",
    "error_message" : "A resource with the specified name could not be found."
  },
  "id" : "req123"
}
```

Note

Se il connettore rileva un errore ripetibile (ad esempio errori di connessione), riprova la pubblicazione nel batch successivo. Il backoff esponenziale viene gestito dall'SDK. AWS Le richieste con errori non irreversibili vengono aggiunte alla coda del canale per ulteriore pubblicazione in base al parametro `IotAnalyticsQueueDropBehavior`.

Esempio di utilizzo

Usa i seguenti passaggi di alto livello per configurare una funzione Lambda di esempio di Python 3.7 che puoi usare per provare il connettore.

Note

- Se usi altri runtime Python, puoi creare un collegamento simbolico da Python3.x a Python 3.7.

- Gli argomenti [Nozioni di base sui connettori \(console\)](#) e [Nozioni di base sui connettori \(CLI\)](#) contengono passaggi dettagliati che illustrano come configurare e distribuire un connettore Twilio Notifications di esempio.

1. Assicurarsi di soddisfare i [requisiti](#) per il connettore.

Per il requisito del ruolo di gruppo, è necessario configurare il ruolo in modo da concedere le autorizzazioni necessarie e assicurarsi che il ruolo sia stato aggiunto al gruppo. Per ulteriori informazioni, consulta [the section called “Gestione del ruolo del gruppo \(console\)”](#) o [the section called “Gestire il ruolo del gruppo \(CLI\)”](#).

2. Crea e pubblica una funzione Lambda che invia dati di input al connettore.

Salvare il [codice di esempio](#) come file PY. Scarica e decomprimi il [AWS IoT Greengrass Core SDK per Python](#). Quindi, crea un pacchetto zip che contiene il file PY e la cartella greengrasssdk a livello root. Questo pacchetto zip è il pacchetto di distribuzione in cui carichi AWS Lambda

Dopo aver creato la funzione Python 3.7 Lambda, pubblica una versione della funzione e crea un alias.

3. Configurare il gruppo Greengrass.
 - a. Aggiungi la funzione Lambda tramite il relativo alias (consigliato). Configura il ciclo di vita Lambda come longevo (o nella "Pinned": true CLI).
 - b. Aggiungere il connettore e configurarne i relativi [parametri](#).
 - c. Aggiungere sottoscrizioni che consentono al connettore di ricevere [i dati di input](#) e inviare [i dati di output](#) nei filtri degli argomenti supportati.
 - Imposta la funzione Lambda come origine, il connettore come destinazione e utilizza un filtro per argomenti di input supportato.
 - Imposta il connettore come origine, AWS IoT Core come destinazione e utilizza un filtro per l'argomento di output supportato. Utilizzi questo abbonamento per visualizzare i messaggi di stato nella AWS IoT console.
4. Distribuisci il gruppo.
5. Nella AWS IoT console, nella pagina Test, sottoscrivi l'argomento relativo ai dati di output per visualizzare i messaggi di stato dal connettore. La funzione Lambda di esempio è di lunga durata e inizia a inviare messaggi subito dopo l'implementazione del gruppo.

Al termine del test, puoi impostare il ciclo di vita Lambda su richiesta (o nella CLI) e "Pinned": false distribuire il gruppo. Ciò impedisce alla funzione di inviare messaggi.

Esempio

L'esempio seguente della funzione Lambda invia un messaggio di input al connettore.

```
import greengrasssdk
import time
import json

iot_client = greengrasssdk.client('iot-data')
send_topic = 'iotanalytics/channels/my_channel/messages/put'

def create_request_with_all_fields():
    return {
        "request": {
            "message" : "{\"temp\":23.33}"
        },
        "id" : "req_123"
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
        payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

Limiti

Questo connettore è soggetto ai limiti descritti di seguito.

- Tutti i limiti AWS SDK per Python (Boto3) imposti dall' AWS IoT Analytics [batch_put_message](#)azione.
- Tutte le quote imposte dall' AWS IoT Analytics [BatchPutMessage](#)API. Per ulteriori informazioni, vedere [Service Quotas](#) for AWS IoT Analytics in. Riferimenti generali di AWS

- 100.000 messaggi al secondo per canale.
- 100 messaggi per batch.
- 128 KB per messaggio.

Questa API utilizza i nomi dei canali (non i canali ARNs), pertanto l'invio di dati a canali interregionali o tra account non è supportato.

- Tutte le quote imposte dal Core. AWS IoT Greengrass Per ulteriori informazioni, vedere [Service Quotas](#) for the AWS IoT Greengrass core in. Riferimenti generali di AWS

In particolare potrebbero essere applicate le seguenti quote:

- La dimensione massima dei messaggi inviati da un dispositivo è 128 KB.
- La dimensione massima della coda di messaggi nel router core di Greengrass è 2,5 MB.
- La lunghezza massima di una stringa argomento è 256 byte di caratteri con codifica UTF-8.

Licenze

Il connettore IoT Analytics include i seguenti software/licenze di terze parti:

- [AWS SDK per Python \(Boto3\)](#)/Apache License 2.0
- [botocore](#)/Apache License 2.0
- [dateutil](#)/PSF License
- [docutils](#)/BSD License, GNU General Public License (GPL), Python Software Foundation License, Public Domain
- [jmespath](#)/MIT License
- [s3transfer](#)/Apache License 2.0
- [urllib3](#)/MIT License

Questo connettore è rilasciato ai sensi del contratto di [licenza del software Greengrass Core](#).

Changelog

La tabella seguente descrive le modifiche apportate a ciascuna versione del connettore.

Versione	Modifiche
4	Aggiunge il <code>IsolationMode</code> parametro per configurare la modalità di containerizzazione per il connettore.
3	È stato aggiornato il runtime Lambda a Python 3.7, che modifica i requisiti di runtime.
2	Correggere per ridurre l'eccessiva registrazione di log.
1	Versione iniziale.

Un gruppo Greengrass può contenere una sola versione del connettore alla volta. Per informazioni sull'aggiornamento di una versione del connettore, consulta [the section called “Aggiornamento delle versioni dei connettori”](#).

Consulta anche

- [Integrazione con servizi e protocolli tramite i connettori](#)
- [the section called “Nozioni di base sui connettori \(console\)”](#)
- [the section called “Nozioni di base sui connettori \(CLI\)”](#)
- [Che cos'è? AWS IoT Analytics](#) nella Guida per l'AWS IoT Analytics utente

Connettore adattatore per protocollo IP Ethernet IoT

Il [connettore](#) IoT Ethernet IP Protocol Adapter raccoglie i dati dai dispositivi locali utilizzando il protocollo EtherNet/IP. È possibile utilizzare questo connettore per raccogliere dati da più dispositivi e pubblicarli in un flusso di messaggi. `StreamManager`

Puoi utilizzare questo connettore anche con il SiteWise connettore IoT e il tuo SiteWise gateway IoT. Il gateway deve fornire la configurazione per il connettore. Per ulteriori informazioni, consulta [Configurare una sorgente EtherNet/IP \(EIP\) nella](#) guida per l'utente IoT. SiteWise

 Note

Questo connettore funziona in modalità [Nessun contenitore](#) isolato, quindi puoi distribuirlo a un AWS IoT Greengrass gruppo in esecuzione in un contenitore Docker.

Questo connettore ha le seguenti versioni.

Versione	ARN
2 (consigliato)	<code>arn:aws:greengrass: <i>region</i>::/connectors/IoTEIPProtocolAdaptor/versions/2</code>
1	<code>arn:aws:greengrass: <i>region</i>::/connectors/IoTEIPProtocolAdaptor/versions/1</code>

Per informazioni sulle modifiche di ogni versione, consulta [Changelog](#).

Requisiti

Questo connettore presenta i seguenti requisiti:

Version 1 and 2

- AWS IoT Greengrass Software di base v1.10.2 o successivo.
- Stream manager abilitato sul gruppo. AWS IoT Greengrass
- Java 8 installato sul dispositivo principale e aggiunto alla variabile di PATH ambiente.
- Un minimo di 256 MB di RAM aggiuntiva. Questo requisito si aggiunge ai requisiti di memoria AWS IoT Greengrass Core.

 Note

Questo connettore è disponibile solo nelle seguenti regioni:

- cn-north-1

- ap-southeast-1
- ap-southeast-2
- eu-central-1
- eu-west-1
- us-east-1
- us-west-2

Parametri del connettore

Questo connettore supporta i seguenti parametri:

LocalStoragePath

La directory sull' AWS IoT Greengrass host su cui il SiteWise connettore IoT può scrivere dati persistenti. La directory predefinita è `/var/sitewise`.

Nome visualizzato nella AWS IoT console: Percorso di archiviazione locale

Obbligatorio: `false`

Tipo: `string`

Schema valido: `^\s*$|\/`.

ProtocolAdapterConfiguration

L'insieme di configurazioni del collettore EtherNet/IP da cui il connettore raccoglie i dati o a cui si connette. Può trattarsi di un elenco vuoto.

Nome visualizzato nella AWS IoT console: Protocol Adapter Configuration

Richiesto: `true`

Tipo: una stringa JSON ben formata che definisce l'insieme di configurazioni di feedback supportate.

Di seguito è riportato un esempio di: `ProtocolAdapterConfiguration`

```
{
```

```

    "sources": [
      {
        "type": "EIPSource",
        "name": "TestSource",
        "endpoint": {
          "ipAddress": "52.89.2.42",
          "port": 44818
        },
        "destination": {
          "type": "StreamManager",
          "streamName": "MyOutput_Stream",
          "streamBufferSize": 10
        },
        "destinationPathPrefix": "EIPSource_Prefix",
        "propertyGroups": [
          {
            "name": "DriveTemperatures",
            "scanMode": {
              "type": "POLL",
              "rate": 10000
            },
            "tagPathDefinitions": [
              {
                "type": "EIPTagPath",
                "path": "arrayREAL[0]",
                "dstDataType": "double"
              }
            ]
          }
        ]
      }
    ]
  }
}

```

Esempio di creazione di un connettore (AWS CLI)

Il seguente comando CLI crea una `ConnectorDefinition` versione iniziale che contiene il connettore IoT Ethernet IP Protocol Adapter.

```

aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version
'{
  "Connectors": [

```

```

    {
      "Id": "MyIoTEIPProtocolConnector",
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/
IoTEIPProtocolAdaptor/versions/2",
      "Parameters": {
        "ProtocolAdaptorConfiguration": "{ \"sources\": [{ \"type
\": \"EIPSource\", \"name\": \"Source1\", \"endpoint\": { \"ipAddress\":
\"54.245.77.218\", \"port\": 44818 }, \"destinationPathPrefix\": \"EIPConnector_Prefix
\", \"propertyGroups\": [{ \"name\": \"Values\", \"scanMode\": { \"type\": \"POLL\",
\"rate\": 2000 }, \"tagPathDefinitions\": [{ \"type\": \"EIPTagPath\", \"path\":
\"arrayREAL[0]\", \"dstDataType\": \"double\" }]]]]}",
        "LocalStoragePath": "/var/MyIoTEIPProtocolConnectorState"
      }
    }
  ]
}'

```

Note

La funzione Lambda in questo connettore ha un ciclo di vita di [lunga durata](#).

Dati di input

Questo connettore non accetta messaggi MQTT come dati di input.

Dati di output

Questo connettore pubblica dati su `StreamManager`. È necessario configurare il flusso di messaggi di destinazione. I messaggi di output hanno la seguente struttura:

```

{
  "alias": "string",
  "messages": [
    {
      "name": "string",
      "value": boolean|double|integer|string,
      "timestamp": number,
      "quality": "string"
    }
  ]
}

```

Licenze

Il connettore IoT Ethernet IP Protocol Adapter include i seguenti software/licenze di terze parti:

- [Client EtherNet/IP](#)
- [Mappa DB](#)
- [Elsa](#)

Questo connettore è rilasciato ai sensi del contratto di [licenza del software Greengrass Core](#).

Changelog

La tabella seguente descrive le modifiche apportate a ciascuna versione del connettore.

Versione	Modifiche	Data
2	Questa versione contiene correzioni di bug.	23 dicembre 2021
1	Versione iniziale.	15 dicembre 2020

Un gruppo Greengrass può contenere una sola versione del connettore alla volta. Per informazioni sull'aggiornamento di una versione del connettore, consulta [the section called “Aggiornamento delle versioni dei connettori”](#).

Consulta anche

- [Integrazione con servizi e protocolli tramite i connettori](#)
- [the section called “Nozioni di base sui connettori \(console\)”](#)
- [the section called “Nozioni di base sui connettori \(CLI\)”](#)

SiteWise Connettore IoT

Il SiteWise connettore IoT invia i dati locali di dispositivi e apparecchiature alle proprietà degli asset in AWS IoT SiteWise. È possibile utilizzare questo connettore per raccogliere dati da più server OPC-UA e pubblicarli su IoT. SiteWise Il connettore invia i dati alle proprietà degli asset nell'area corrente Account AWS e nella regione.

 Note

SiteWise L'IoT è un servizio completamente gestito che raccoglie, elabora e visualizza i dati provenienti da dispositivi e apparecchiature industriali. Puoi configurare le proprietà degli asset che elaborano i dati grezzi inviati da questo connettore alle proprietà di misurazione degli asset. Ad esempio, puoi definire una proprietà di trasformazione che converte i punti dati di temperatura Celsius di un dispositivo in Fahrenheit oppure puoi definire una proprietà del parametro che calcola la temperatura oraria media. Per ulteriori informazioni, consulta [Che cos'è AWS IoT SiteWise?](#) nella Guida per l'utente di AWS IoT SiteWise .

Il connettore invia dati all'IoT SiteWise con i percorsi del flusso di dati OPC-UA inviati dai server OPC-UA. Ad esempio, il percorso del flusso di dati `/company/windfarm/3/turbine/7/temperature` potrebbe rappresentare il sensore di temperatura della turbina #7 nel parco eolico #3. Se il AWS IoT Greengrass core perde la connessione a Internet, il connettore memorizza i dati nella cache finché non riesce a connettersi correttamente a Cloud AWS. Puoi configurare la dimensione massima del buffer del disco utilizzato per il caching dei dati. Se la dimensione della cache supera la dimensione massima del buffer del disco, il connettore elimina i dati meno recenti dalla coda.

[Dopo aver configurato e distribuito il SiteWise connettore IoT, puoi aggiungere un gateway e sorgenti OPC-UA nella console IoT. SiteWise](#) Quando configuri una sorgente nella console, puoi filtrare o aggiungere un prefisso ai percorsi del flusso di dati OPC-UA inviati dal connettore IoT. SiteWise Per istruzioni su come completare la configurazione del gateway e delle origini, consulta [Aggiunta del gateway](#) nel Manuale dell'utente di AWS IoT SiteWise .

L'IoT SiteWise riceve dati solo dai flussi di dati mappati alle proprietà di misurazione degli asset SiteWise IoT. Per mappare i flussi di dati alle proprietà degli asset, puoi impostare l'alias di una proprietà in modo che sia equivalente a un percorso del flusso di dati OPC-UA. Per informazioni sulla definizione dei modelli di asset e la creazione di asset, consulta la sezione relativa alla [modellazione degli asset industriali](#) nella Guida per l'utente di AWS IoT SiteWise .

 Note

Puoi utilizzare stream manager per caricare dati su IoT SiteWise da fonti diverse dai server OPC-UA. Stream manager fornisce anche un supporto personalizzabile per la persistenza e la gestione della larghezza di banda. Per ulteriori informazioni, consulta [Gestione dei flussi di dati](#).

Questo connettore funziona in modalità [No container](#) isolation, quindi puoi distribuirlo a un gruppo Greengrass in esecuzione in un contenitore Docker.

Questo connettore ha le seguenti versioni.

Versione	ARN
12 (consigliato)	<code>arn:aws:greengrass: <i>region</i>::/connectors/IoTSiteWise/versions/ 12</code>
11	<code>arn:aws:greengrass: <i>region</i>::/connectors/IoTSiteWise/versions/ 11</code>
10	<code>arn:aws:greengrass: <i>region</i>::/connectors/IoTSiteWise/versions/ 10</code>
9	<code>arn:aws:greengrass: <i>region</i>::/connectors/IoTSiteWise/versions/ 9</code>
8	<code>arn:aws:greengrass: <i>region</i>::/connectors/IoTSiteWise/versions/ 8</code>
7	<code>arn:aws:greengrass: <i>region</i>::/connectors/IoTSiteWise/versions/ 7</code>
6	<code>arn:aws:greengrass: <i>region</i>::/connectors/IoTSiteWise/versions/ 6</code>

Versione	ARN
5	arn:aws:greengrass: <i>region</i> ::/connectors/IoTSiteWise/versions/ 5
4	arn:aws:greengrass: <i>region</i> ::/connectors/IoTSiteWise/versions/ 4
3	arn:aws:greengrass: <i>region</i> ::/connectors/IoTSiteWise/versions/ 3
2	arn:aws:greengrass: <i>region</i> ::/connectors/IoTSiteWise/versions/ 2
1	arn:aws:greengrass: <i>region</i> ::/connectors/IoTSiteWise/versions/ 1

Per informazioni sulle modifiche di ogni versione, consulta [Changelog](#).

Requisiti

Questo connettore presenta i seguenti requisiti:

Version 9, 10, 11, and 12

Important

[Questa versione introduce nuovi requisiti: software di AWS IoT Greengrass base v1.10.2 e stream manager.](#)

- AWS IoT Greengrass Software di base v1.10.2.
- [Gestore di flusso](#) attivato nel gruppo Greengrass.

- Java 8 installato sul dispositivo core e aggiunto alla variabile di ambiente PATH.
- Questo connettore può essere utilizzato solo nelle regioni di Amazon Web Services in cui SiteWise sono supportati [AWS IoT Greengrass](#) sia l'[IoT](#).
- Una policy IAM aggiunta al ruolo del gruppo Greengrass. Questo ruolo consente al AWS IoT Greengrass gruppo di accedere all'`iotsitewise:BatchPutAssetPropertyValue` azione sull'asset root di destinazione e sui relativi figli, come illustrato nell'esempio seguente. Puoi rimuoverlo `Condition` dalla policy per consentire al connettore di accedere a tutte le tue SiteWise risorse IoT.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iotsitewise:BatchPutAssetPropertyValue",
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "iotsitewise:assetHierarchyPath": [
            "/root node asset ID",
            "/root node asset ID/*"
          ]
        }
      }
    }
  ]
}
```

Per ulteriori informazioni, consulta [Aggiungere e rimuovere le policy IAM](#) nella IAM User Guide.

Versions 6, 7, and 8

Important

[Questa versione introduce nuovi requisiti: software di AWS IoT Greengrass base v1.10.0 e stream manager.](#)

- AWS IoT Greengrass Software di base v1.10.0.

- [Gestore di flusso](#) attivato nel gruppo Greengrass.
- Java 8 installato sul dispositivo core e aggiunto alla variabile di ambiente PATH.
- Questo connettore può essere utilizzato solo nelle regioni di Amazon Web Services in cui SiteWise sono supportati [AWS IoT Greengrass](#) sia l'[IoT](#).
- Una policy IAM aggiunta al ruolo del gruppo Greengrass. Questo ruolo consente al AWS IoT Greengrass gruppo di accedere all'`iotsitewise:BatchPutAssetPropertyValue` azione sull'asset root di destinazione e sui relativi figli, come illustrato nell'esempio seguente. Puoi rimuoverlo `Condition` dalla policy per consentire al connettore di accedere a tutte le tue SiteWise risorse IoT.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iotsitewise:BatchPutAssetPropertyValue",
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "iotsitewise:assetHierarchyPath": [
            "/root node asset ID",
            "/root node asset ID/*"
          ]
        }
      }
    }
  ]
}
```

Per ulteriori informazioni, consulta [Aggiungere e rimuovere le policy IAM](#) nella IAM User Guide.

Version 5

- AWS IoT Greengrass Software di base v1.9.4.
- Java 8 installato sul dispositivo core e aggiunto alla variabile di ambiente PATH.
- Questo connettore può essere utilizzato solo nelle regioni di Amazon Web Services in cui SiteWise sono supportati [AWS IoT Greengrass](#) sia l'[IoT](#).

- Una policy IAM aggiunta al ruolo del gruppo Greengrass. Questo ruolo consente al AWS IoT Greengrass gruppo di accedere all'`iotsitewise:BatchPutAssetPropertyValue` azione sull'asset root di destinazione e sui relativi figli, come illustrato nell'esempio seguente. Puoi rimuoverlo `Condition` dalla policy per consentire al connettore di accedere a tutte le tue SiteWise risorse IoT.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iotsitewise:BatchPutAssetPropertyValue",
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "iotsitewise:assetHierarchyPath": [
            "/root node asset ID",
            "/root node asset ID/*"
          ]
        }
      }
    }
  ]
}
```

Per ulteriori informazioni, consulta [Aggiungere e rimuovere le policy IAM](#) nella IAM User Guide.

Version 4

- AWS IoT Greengrass Software di base v1.10.0.
- Java 8 installato sul dispositivo core e aggiunto alla variabile di ambiente `PATH`.
- Questo connettore può essere utilizzato solo nelle regioni di Amazon Web Services in cui SiteWise sono supportati [AWS IoT Greengrass](#) sia l'[IoT](#).
- Una policy IAM aggiunta al ruolo del gruppo Greengrass. Questo ruolo consente al AWS IoT Greengrass gruppo di accedere all'`iotsitewise:BatchPutAssetPropertyValue` azione sull'asset root di destinazione e sui relativi figli, come illustrato nell'esempio seguente. Puoi rimuoverlo `Condition` dalla policy per consentire al connettore di accedere a tutte le tue SiteWise risorse IoT.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iotsitewise:BatchPutAssetPropertyValue",
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "iotsitewise:assetHierarchyPath": [
            "/root node asset ID",
            "/root node asset ID/*"
          ]
        }
      }
    }
  ]
}
```

Per ulteriori informazioni, consulta [Aggiungere e rimuovere le policy IAM](#) nella IAM User Guide.

Version 3

- AWS IoT Greengrass Software di base v1.9.4.
- Java 8 installato sul dispositivo core e aggiunto alla variabile di ambiente PATH.
- Questo connettore può essere utilizzato solo nelle regioni di Amazon Web Services in cui SiteWise sono supportati [AWS IoT Greengrass](#) e [l'IoT](#).
- Una policy IAM aggiunta al ruolo del gruppo Greengrass. Questo ruolo consente al AWS IoT Greengrass gruppo di accedere all'`iotsitewise:BatchPutAssetPropertyValue` azione sull'asset root di destinazione e sui relativi figli, come illustrato nell'esempio seguente. Puoi rimuoverlo Condition dalla policy per consentire al connettore di accedere a tutte le tue SiteWise risorse IoT.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iotsitewise:BatchPutAssetPropertyValue",
```

```

        "Resource": "*",
        "Condition": {
            "StringLike": {
                "iotsitewise:assetHierarchyPath": [
                    "/root node asset ID",
                    "/root node asset ID/*"
                ]
            }
        }
    }
]
}

```

Per ulteriori informazioni, consulta [Aggiungere e rimuovere le policy IAM](#) nella IAM User Guide.

Versions 1 and 2

- AWS IoT Greengrass Software di base v1.9.4.
- Java 8 installato sul dispositivo core e aggiunto alla variabile di ambiente PATH.
- Questo connettore può essere utilizzato solo nelle regioni di Amazon Web Services in cui SiteWise sono supportati [AWS IoT Greengrass](#) sia l'[IoT](#).
- Una policy IAM aggiunta al ruolo del gruppo Greengrass che consente l'accesso AWS IoT Core e l'`iotsitewise:BatchPutAssetPropertyValue` azione sull'asset root di destinazione e sui relativi figli, come illustrato nell'esempio seguente. Puoi rimuoverlo `Condition` dalla policy per consentire al connettore di accedere a tutte le tue SiteWise risorse IoT.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iotsitewise:BatchPutAssetPropertyValue",
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "iotsitewise:assetHierarchyPath": [
            "/root node asset ID",
            "/root node asset ID/*"
          ]
        }
      }
    }
  ]
}

```

```
    }
  },
  {
    "Effect": "Allow",
    "Action": [
      "iot:Connect",
      "iot:DescribeEndpoint",
      "iot:Publish",
      "iot:Receive",
      "iot:Subscribe"
    ],
    "Resource": "*"
  }
]
```

Per ulteriori informazioni, consulta [Aggiunta e rimozione di autorizzazioni per identità IAM](#) nella Guida per l'utente di IAM .

Parametri

Versions 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, and 12

SiteWiseLocalStoragePath

La directory sull' AWS IoT Greengrass host su cui il SiteWise connettore IoT può scrivere dati persistenti. L'impostazione predefinita è `/var/sitewise`.

Nome visualizzato nella AWS IoT console: Percorso di archiviazione locale

Obbligatorio: false

Tipo: string

Schema valido: `^\s*$|\/`.

AWSecretsArnList

Un elenco di segreti, ciascuno dei quali contiene una coppia chiave-valore di nome utente e password OPC-UA. AWS Secrets Manager Ciascun segreto deve essere di tipo coppia chiave-valore.

Nome visualizzato nella AWS IoT console: elenco dei segreti relativi a nome utente/password OPC-UA ARNs

Obbligatorio: false

Tipo: `JSONArrayOfStrings`

Schema valido: `\[( ? , ? ? \"(arn:(aws(-[a-z]+)*):secretsmanager:[a-z0-9\\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\\\\\\\]+\\|)*[a-zA-Z0-9\\|_+=, .@\\-]+-[a-zA-Z0-9]+)*\" )*\]`

`MaximumBufferSize`

La dimensione massima in GB per l'utilizzo SiteWise del disco IoT. Il valore predefinito è 10 GB.

Nome visualizzato nella AWS IoT console: dimensione massima del buffer del disco

Obbligatorio: false

Tipo: `string`

Schema valido: `^\\s*$|[0-9]+`

Version 1

`SiteWiseLocalStoragePath`

La directory sull' AWS IoT Greengrass host su cui il SiteWise connettore IoT può scrivere dati persistenti. L'impostazione predefinita è `/var/sitewise`.

Nome visualizzato nella AWS IoT console: Percorso di archiviazione locale

Obbligatorio: false

Tipo: `string`

Schema valido: `^\\s*$|\\|\\.`

`SiteWiseOpcuaUserIdentityTokenSecretArn`

Il segreto contiene AWS Secrets Manager la coppia chiave-valore del nome utente e della password OPC-UA. Questo segreto deve essere di tipo coppia chiave-valore.

Nome visualizzato nella AWS IoT console: ARN del nome utente/password segreto OPC-UA

Obbligatorio: false

Tipo: string

Schema valido: `^$|arn:(aws(-[a-z]+)*):secretsmanager:[a-z0-9\\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\-\\+\\/]*[a-zA-Z0-9/_+=, .@\\-]+-[a-zA-Z0-9]+`

`SiteWiseOpcuaUserIdentityTokenSecretArn-ResourceId`

La risorsa segreta del AWS IoT Greengrass gruppo che fa riferimento a un nome utente e una password OPC-UA segreti.

Nome visualizzato nella AWS IoT console: risorsa segreta per nome utente/password OPC-UA

Obbligatorio: false

Tipo: string

Schema valido: `^$|.+`

`MaximumBufferSize`

La dimensione massima in GB per l'utilizzo SiteWise del disco IoT. Il valore predefinito è 10 GB.

Nome visualizzato nella AWS IoT console: dimensione massima del buffer del disco

Obbligatorio: false

Tipo: string

Schema valido: `^\\s*$|[0-9]+`

Esempio di creazione di un connettore (AWS CLI)

Il AWS CLI comando seguente crea un file `ConnectorDefinition` con una versione iniziale che contiene il SiteWise connettore IoT.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
  "Connectors": [
    {
```

```
"Id": "MyIoTSiteWiseConnector",
"ConnectorArn": "arn:aws:greengrass:region::/connectors/IoTSiteWise/
versions/11"
    }
  ]
}'
```

Note

Le funzioni Lambda di questo connettore hanno un ciclo di vita di [lunga durata](#).

Nella AWS IoT Greengrass console, puoi aggiungere un connettore dalla pagina Connettori del gruppo. Per ulteriori informazioni, consulta [the section called “Nozioni di base sui connettori \(console\)”](#).

Dati di input

Questo connettore non accetta messaggi MQTT come dati di input.

Dati di output

Questo connettore non pubblica messaggi MQTT come dati di output.

Limiti

Questo connettore è soggetto ai seguenti limiti imposti dall'IoT SiteWise, inclusi i seguenti. Per ulteriori informazioni, consulta [AWS IoT SiteWise endpoint e quote](#) in. Riferimenti generali di AWS

- Numero massimo di gateway per. Account AWS
- Numero massimo di origini OPC-UA per gateway.
- Velocità massima di punti dati timestamp-quality-value (TQV) archiviati per. Account AWS
- Tasso massimo di punti dati TQV archiviati per proprietà asset.

Licenze

Version 9, 10, 11, and 12

Il SiteWise connettore IoT include i seguenti software/licenze di terze parti:

- [MapDB](#)
- [Elsa](#)
- [Eclissi Milo](#)

Questo connettore è rilasciato ai sensi del contratto di [licenza del software Greengrass Core](#).

Versions 6, 7, and 8

Il SiteWise connettore IoT include i seguenti software/licenze di terze parti:

- [Milo](#) / EDL 1.0

Questo connettore è rilasciato ai sensi del contratto di [licenza del software Greengrass Core](#).

Versions 1, 2, 3, 4, and 5

Il SiteWise connettore IoT include i seguenti software/licenze di terze parti:

- [Milo](#) / EDL 1.0
- [Chronicle-Queue](#) /Licenza Apache 2.0

Questo connettore è rilasciato ai sensi del contratto di [licenza del software Greengrass Core](#).

Changelog

La tabella seguente descrive le modifiche apportate a ciascuna versione del connettore.

Versione	Modifiche	Data
12	• Questa versione contiene correzioni di bug.	22 dicembre 2021
11	• Support per stringhe che contengono caratteri nascosti o non stampabili. I caratteri nascosti e non stampabili vengono rimossi automaticamente prima che	24 marzo 2021

Versione	Modifiche	Data
	le stringhe vengano inviate a. Cloud AWS • È stato risolto un problema che faceva sì che il SiteWise gateway IoT tentasse all'infinito di richieste non valide. • È stato risolto un problema che causava un checkpoint danneggiato quando il SiteWise gateway IoT era connesso a una fonte di dati ad alta frequenza. • Messaggi di errore migliorati per aiutare a risolvere i problemi di configurazione del gateway.	
10	Configurato StreamManager per migliorare la gestione in caso di interruzione e ripristino della connessione di origine. Questa versione accetta anche valori OPC-UA con un ServerTimestamp quando non è SourceTimestamp disponibile.	22 gennaio 2021

Versione	Modifiche	Data
9	Supporto lanciato per destinazioni di StreamManager streaming Greengrass personalizzate, deadbanding OPC-UA, modalità di scansione personalizzata e velocità di scansione personalizzata. Include anche prestazioni migliorate durante gli aggiornamenti della configurazione effettuati dal SiteWise gateway IoT.	15 dicembre 2020
8	Maggiore stabilità quando il connettore presenta una connettività di rete intermittente.	19 novembre 2020
7	È stato risolto un problema con le metriche del gateway.	14 agosto 2020
6	È stato aggiunto il supporto per le CloudWatch metriche e l'individuazione automatica di nuovi tag OPC-UA. Questa versione richiede lo stream manager e il software AWS IoT Greengrass Core v1.10.0 o superiore.	29 aprile 2020
5	È stato risolto un problema di compatibilità con il software AWS IoT Greengrass Core v1.9.4.	12 febbraio 2020

Versione	Modifiche	Data
4	Risolto un problema con la riconnessione del server OPC-UA.	7 febbraio 2020
3	Rimozione del requisito delle autorizzazioni <code>iot:*</code> .	17 dicembre 2019
2	Aggiunto il supporto per più risorse segrete OPC-UA.	10 dicembre 2019
1	Versione iniziale.	2 dicembre 2019

Un gruppo Greengrass può contenere una sola versione del connettore alla volta. Per informazioni sull'aggiornamento di una versione del connettore, consulta [the section called “Aggiornamento delle versioni dei connettori”](#).

Consulta anche

- [Integrazione con servizi e protocolli tramite i connettori](#)
- [the section called “Nozioni di base sui connettori \(console\)”](#)
- [the section called “Nozioni di base sui connettori \(CLI\)”](#)
- Consulta i seguenti argomenti nella Guida per l'utente di AWS IoT SiteWise :
 - [Che cos'è? AWS IoT SiteWise](#)
 - [Utilizzo di un gateway](#)
 - [Metriche del CloudWatch gateway](#)
 - [Risoluzione dei problemi di un SiteWise gateway IoT](#)

Kinesis Firehose

Il [connettore](#) Kinesis Firehose pubblica i dati tramite un flusso di distribuzione Amazon Data Firehose verso destinazioni come Amazon S3, Amazon Redshift o Amazon Service. OpenSearch

Questo connettore è un produttore di dati per un flusso di distribuzione Kinesis. Riceve i dati di input in un argomento MQTT e invia i dati al flusso di distribuzione specificato. Il flusso di distribuzione invia il record dei dati alla destinazione configurata (per esempio, un bucket S3).

Questo connettore ha le seguenti versioni.

Versione	ARN
5	arn:aws:greengrass: <i>region</i> ::/connectors/KinesisFirehose/versions/5
4	arn:aws:greengrass: <i>region</i> ::/connectors/KinesisFirehose/versions/4
3	arn:aws:greengrass: <i>region</i> ::/connectors/KinesisFirehose/versions/3
2	arn:aws:greengrass: <i>region</i> ::/connectors/KinesisFirehose/versions/2
1	arn:aws:greengrass: <i>region</i> ::/connectors/KinesisFirehose/versions/1

Per informazioni sulle modifiche di ogni versione, consulta [Changelog](#).

Requisiti

Questo connettore presenta i seguenti requisiti:

Version 4 - 5

- AWS IoT Greengrass Software principale v1.9.3 o successivo.
- [Python](#) versione 3.7 o 3.8 installata sul dispositivo principale e aggiunta alla variabile di ambiente PATH.

 Note

Per usare Python 3.8, esegui il seguente comando per creare un collegamento simbolico dalla cartella di installazione predefinita di Python 3.7 ai binari Python 3.8 installati.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Questo configura il dispositivo in modo che soddisfi il requisito Python per AWS IoT Greengrass.

- Un flusso di distribuzione Kinesis configurato. Per ulteriori informazioni, consulta [Creazione di un flusso di distribuzione di Amazon Data Firehose](#) nella Amazon Kinesis Firehose Developer Guide.
- Il [ruolo del gruppo Greengrass](#) è configurato per consentire le `firehose:PutRecordBatch` azioni `firehose:PutRecord` e sul flusso di consegna di destinazione, come mostrato nel seguente esempio di politica IAM.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1528133056761",
      "Action": [
        "firehose:PutRecord",
        "firehose:PutRecordBatch"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:firehose:region:account-id:deliverystream/stream-name"
      ]
    }
  ]
}
```

Questo connettore consente di sostituire dinamicamente il flusso di distribuzione predefinito nel payload del messaggio di input. Se l'implementazione utilizza questa funzionalità, la policy IAM dovrebbe includere tutti i flussi di destinazione come risorse. Puoi concedere alle risorse un

accesso granulare o condizionale (ad esempio, utilizzando uno schema di denominazione con il carattere jolly *).

Per il requisito del ruolo di gruppo, è necessario configurare il ruolo in modo da concedere le autorizzazioni necessarie e assicurarsi che il ruolo sia stato aggiunto al gruppo. Per ulteriori informazioni, consulta [the section called “Gestione del ruolo del gruppo \(console\)”](#) o [the section called “Gestire il ruolo del gruppo \(CLI\)”](#).

Versions 2 - 3

- AWS IoT Greengrass Software di base v1.7 o successivo.
- [Python](#) versione 2.7 installato sul dispositivo principale e aggiunto alla variabile di ambiente PATH.
- Un flusso di distribuzione Kinesis configurato. Per ulteriori informazioni, consulta [Creazione di un flusso di distribuzione di Amazon Data Firehose](#) nella Amazon Kinesis Firehose Developer Guide.
- Il [ruolo del gruppo Greengrass](#) è configurato per consentire le `firehose:PutRecordBatch` azioni `firehose:PutRecord` e sul flusso di consegna di destinazione, come mostrato nel seguente esempio di politica IAM.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1528133056761",
      "Action": [
        "firehose:PutRecord",
        "firehose:PutRecordBatch"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:firehose:region:account-id:deliverystream/stream-name"
      ]
    }
  ]
}
```

Questo connettore consente di sostituire dinamicamente il flusso di distribuzione predefinito nel payload del messaggio di input. Se l'implementazione utilizza questa funzionalità, la policy

IAM dovrebbe includere tutti i flussi di destinazione come risorse. Puoi concedere alle risorse un accesso granulare o condizionale (ad esempio, utilizzando uno schema di denominazione con il carattere jolly *).

Per il requisito del ruolo di gruppo, è necessario configurare il ruolo in modo da concedere le autorizzazioni necessarie e assicurarsi che il ruolo sia stato aggiunto al gruppo. Per ulteriori informazioni, consulta [the section called “Gestione del ruolo del gruppo \(console\)”](#) o [the section called “Gestire il ruolo del gruppo \(CLI\)”](#).

Version 1

- AWS IoT Greengrass Software di base v1.7 o successivo.
- [Python](#) versione 2.7 installato sul dispositivo principale e aggiunto alla variabile di ambiente PATH.
- Un flusso di distribuzione Kinesis configurato. Per ulteriori informazioni, consulta [Creazione di un flusso di distribuzione di Amazon Data Firehose](#) nella Amazon Kinesis Firehose Developer Guide.
- Il [ruolo del gruppo Greengrass](#) è configurato per consentire l'firehose:PutRecordazione sul flusso di consegna di destinazione, come mostrato nel seguente esempio di politica IAM.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1528133056761",
      "Action": [
        "firehose:PutRecord"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:firehose:region:account-id:deliverystream/stream-name"
      ]
    }
  ]
}
```

Questo connettore consente di sostituire dinamicamente il flusso di distribuzione predefinito nel payload del messaggio di input. Se l'implementazione utilizza questa funzionalità, la policy IAM dovrebbe includere tutti i flussi di destinazione come risorse. Puoi concedere alle risorse un

accesso granulare o condizionale (ad esempio, utilizzando uno schema di denominazione con il carattere jolly *).

Per il requisito del ruolo di gruppo, è necessario configurare il ruolo in modo da concedere le autorizzazioni necessarie e assicurarsi che il ruolo sia stato aggiunto al gruppo. Per ulteriori informazioni, consulta [the section called “Gestione del ruolo del gruppo \(console\)”](#) o [the section called “Gestire il ruolo del gruppo \(CLI\)”](#).

Parametri del connettore

Questo connettore fornisce i seguenti parametri:

Versions 5

DefaultDeliveryStreamArn

L'ARN del flusso di distribuzione Firehose predefinito a cui inviare i dati. Il flusso di destinazione può essere ignorato con la proprietà `delivery_stream_arn` del payload del messaggio di input.

Note

Il ruolo del gruppo deve consentire le operazioni appropriate su tutti i flussi di distribuzione di destinazione. Per ulteriori informazioni, consulta [the section called “Requisiti”](#).

Nome visualizzato nella AWS IoT console: flusso di distribuzione predefinito ARN

Obbligatorio: true

Tipo: string

Schema valido: `arn:aws:firehose:([a-z]{2}-[a-z]+\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$`

DeliveryStreamQueueSize

Il numero massimo di record da conservare in memoria prima che vengano rifiutati nuovi record per lo stesso flusso di distribuzione. Il valore minimo è 2000.

Nome visualizzato nella AWS IoT console: numero massimo di record da memorizzare nel buffer (per stream)

Obbligatorio: true

Tipo: string

Schema valido: `^([2-9]\\d{3}|[1-9]\\d{4,})$`

MemorySize

La quantità di memoria (in KB) da allocare al connettore.

Nome visualizzato nella AWS IoT console: dimensione della memoria

Obbligatorio: true

Tipo: string

Schema valido: `^[0-9]+$`

PublishInterval

L'intervallo (in secondi) per la pubblicazione dei record su Firehose. Per disabilitare il batch, impostare questo valore su 0.

Nome visualizzato nella AWS IoT console: Intervallo di pubblicazione

Obbligatorio: true

Tipo: string

Valori validi: 0 - 900

Schema valido: `[0-9]|[1-9]\\d|[1-9]\\d\\d|900`

IsolationMode

Modalità di [containerizzazione](#) per questo connettore. L'impostazione predefinita è `GreengrassContainer`, il che significa che il connettore viene eseguito in un ambiente di runtime isolato all'interno del AWS IoT Greengrass contenitore.

Note

L'impostazione predefinita della containerizzazione per il gruppo non si applica ai connettori.

Nome visualizzato nella AWS IoT console: modalità di isolamento del contenitore

Obbligatorio: false

Tipo: string

Valori validi: GreengrassContainer o NoContainer

Schema valido: ^NoContainer\$|^GreengrassContainer\$

Versions 2 - 4

DefaultDeliveryStreamArn

L'ARN del flusso di distribuzione Firehose predefinito a cui inviare i dati. Il flusso di destinazione può essere ignorato con la proprietà `delivery_stream_arn` del payload del messaggio di input.

Note

Il ruolo del gruppo deve consentire le operazioni appropriate su tutti i flussi di distribuzione di destinazione. Per ulteriori informazioni, consulta [the section called “Requisiti”](#).

Nome visualizzato nella AWS IoT console: flusso di distribuzione predefinito ARN

Obbligatorio: true

Tipo: string

Schema valido: `arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$`

DeliveryStreamQueueSize

Il numero massimo di record da conservare in memoria prima che vengano rifiutati nuovi record per lo stesso flusso di distribuzione. Il valore minimo è 2000.

Nome visualizzato nella AWS IoT console: numero massimo di record da memorizzare nel buffer (per stream)

Obbligatorio: true

Tipo: `string`

Schema valido: `^([2-9]\\d{3}|[1-9]\\d{4,})$`

MemorySize

La quantità di memoria (in KB) da allocare al connettore.

Nome visualizzato nella AWS IoT console: dimensione della memoria

Obbligatorio: `true`

Tipo: `string`

Schema valido: `^[0-9]+$`

PublishInterval

L'intervallo (in secondi) per la pubblicazione dei record su Firehose. Per disabilitare il batch, impostare questo valore su 0.

Nome visualizzato nella AWS IoT console: Intervallo di pubblicazione

Obbligatorio: `true`

Tipo: `string`

Valori validi: 0 - 900

Schema valido: `[0-9]|[1-9]\\d|[1-9]\\d\\d|900`

Version 1

DefaultDeliveryStreamArn

L'ARN del flusso di distribuzione Firehose predefinito a cui inviare i dati. Il flusso di destinazione può essere ignorato con la proprietà `delivery_stream_arn` del payload del messaggio di input.

Note

Il ruolo del gruppo deve consentire le operazioni appropriate su tutti i flussi di distribuzione di destinazione. Per ulteriori informazioni, consulta [the section called “Requisiti”](#).

Nome visualizzato nella AWS IoT console: flusso di distribuzione predefinito ARN

Obbligatorio: true

Tipo: string

Schema valido: `arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$`

Example

Esempio di creazione di connettore (AWS CLI)

Il seguente comando CLI crea un `ConnectorDefinition` con una versione iniziale che contiene il connettore.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
  "Connectors": [
    {
      "Id": "MyKinesisFirehoseConnector",
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/KinesisFirehose/
versions/5",
      "Parameters": {
        "DefaultDeliveryStreamArn": "arn:aws:firehose:region:account-
id:deliverystream/stream-name",
        "DeliveryStreamQueueSize": "5000",
        "MemorySize": "65535",
        "PublishInterval": "10",
        "IsolationMode" : "GreengrassContainer"
      }
    }
  ]
}'
```

Nella AWS IoT Greengrass console, è possibile aggiungere un connettore dalla pagina Connettori del gruppo. Per ulteriori informazioni, consulta [the section called “Nozioni di base sui connettori \(console\)”](#).

Dati di input

Questo connettore accetta i contenuti in streaming negli argomenti MQTT e quindi invia i contenuti nel flusso di distribuzione di destinazione. Accetta due tipi di dati di input:

- Dati JSON nell'argomento `kinesisfirehose/message`.
- Dati binari nell'argomento `kinesisfirehose/message/binary/#`.

Versions 2 - 5

Filtro di argomenti: `kinesisfirehose/message`

Utilizzare questo argomento per inviare un messaggio contenente dati JSON.

Proprietà dei messaggi

`request`

I dati da inviare al flusso di distribuzione e al flusso di distribuzione di destinazione, se diverso da quello predefinito.

Obbligatorio: `true`

Tipo: `object` che include le seguenti proprietà:

`data`

I dati da inviare al flusso di distribuzione.

Obbligatorio: `true`

Tipo: `string`

`delivery_stream_arn`

L'ARN del flusso di distribuzione Kinesis di destinazione. Includi questa proprietà per sovrascrivere il flusso di distribuzione predefinito.

Richiesto: `false`

Tipo: `string`

Schema valido: `arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$`

id

Un ID arbitrario della richiesta. Questa proprietà viene utilizzata per associare una richiesta di input a una risposta di output. Quando specificato, la proprietà `id` nell'oggetto della risposta è impostata su questo valore. Se non utilizzi questa funzione, puoi omettere la proprietà oppure specificare una stringa vuota.

Richiesto: `false`

Tipo: `string`

Schema valido: `.*`

Input di esempio

```
{
  "request": {
    "delivery_stream_arn": "arn:aws:firehose:region:account-
id:deliverystream/stream2-name",
    "data": "Data to send to the delivery stream."
  },
  "id": "request123"
}
```

Filtro di argomenti: `kinesisfirehose/message/binary/#`

Utilizzare questo argomento per inviare un messaggio contenente dati binari. Il connettore non analizza i dati binari. I dati sono trasferiti in streaming così come sono.

Per associare la richiesta di input a una risposta di output, sostituisci il carattere jolly `#` nell'argomento del messaggio con un ID richiesta arbitrario. Ad esempio, se pubblichi un messaggio in `kinesisfirehose/message/binary/request123`, la proprietà `id` nell'oggetto di risposta viene impostata su `request123`.

Se non desideri associare una richiesta a una risposta, puoi pubblicare i messaggi in `kinesisfirehose/message/binary/`. Assicurarsi di includere la barra finale.

Version 1

Filtro di argomenti: `kinesisfirehose/message`

Utilizzare questo argomento per inviare un messaggio contenente dati JSON.

Proprietà dei messaggi

`request`

I dati da inviare al flusso di distribuzione e al flusso di distribuzione di destinazione, se diverso da quello predefinito.

Richiesto: `true`

Tipo: `object` che include le seguenti proprietà:

`data`

I dati da inviare al flusso di distribuzione.

Obbligatorio: `true`

Tipo: `string`

`delivery_stream_arn`

L'ARN del flusso di distribuzione Kinesis di destinazione. Includi questa proprietà per sovrascrivere il flusso di distribuzione predefinito.

Richiesto: `false`

Tipo: `string`

Schema valido: `arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$`

`id`

Un ID arbitrario della richiesta. Questa proprietà viene utilizzata per associare una richiesta di input a una risposta di output. Quando specificato, la proprietà `id` nell'oggetto della risposta è impostata su questo valore. Se non utilizzi questa funzione, puoi omettere la proprietà oppure specificare una stringa vuota.

Richiesto: `false`

Tipo: string

Schema valido: .*

Input di esempio

```
{
  "request": {
    "delivery_stream_arn": "arn:aws:firehose:region:account-  
id:deliverystream/stream2-name",
    "data": "Data to send to the delivery stream."
  },
  "id": "request123"
}
```

Filtro di argomenti: kinesisfirehose/message/binary/#

Utilizzare questo argomento per inviare un messaggio contenente dati binari. Il connettore non analizza i dati binari. I dati sono trasferiti in streaming così come sono.

Per associare la richiesta di input a una risposta di output, sostituisci il carattere jolly # nell'argomento del messaggio con un ID richiesta arbitrario. Ad esempio, se pubblichi un messaggio in kinesisfirehose/message/binary/request123, la proprietà id nell'oggetto di risposta viene impostata su request123.

Se non desideri associare una richiesta a una risposta, puoi pubblicare i messaggi in kinesisfirehose/message/binary/. Assicurarsi di includere la barra finale.

Dati di output

Questo connettore pubblica le informazioni di stato come dati di output su un argomento MQTT.

Versions 2 - 5

Filtro argomento in sottoscrizione

kinesisfirehose/message/status

Output di esempio

La risposta contiene lo stato di ogni record di dati inviati nel batch.

```
{
  "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

Se il connettore rileva un errore riutilizzabile (ad esempio errori di connessione), riprova la pubblicazione nel batch successivo. Il backoff esponenziale viene gestito dall'SDK. AWS Le richieste respinte con errori riproducibili vengono aggiunti alla fine della coda per ulteriore pubblicazione.

Version 1

Filtro argomento in sottoscrizione

kinesisfirehose/message/status

Output di esempio: Operazione riuscita

```
{
  "response": {
    "firehose_record_id": "1lxfuuuFomkpJYzt/34ZU/r8JYPf8Wyf7AXq1Xm",
    "status": "success"
  },
}
```

```
{
  "id": "request123"
}
```

Esempio di output: Errore

```
{
  "response" : {
    "error": "ResourceNotFoundException",
    "error_message": "An error occurred (ResourceNotFoundException) when calling the PutRecord operation: Firehose test1 not found under account 123456789012.",
    "status": "fail"
  },
  "id": "request123"
}
```

Esempio di utilizzo

Usa i seguenti passaggi di alto livello per configurare una funzione Lambda di esempio di Python 3.7 che puoi usare per provare il connettore.

Note

- Se usi altri runtime Python, puoi creare un collegamento simbolico da Python3.x a Python 3.7.
- Gli argomenti [Nozioni di base sui connettori \(console\)](#) e [Nozioni di base sui connettori \(CLI\)](#) contengono passaggi dettagliati che illustrano come configurare e distribuire un connettore Twilio Notifications di esempio.

1. Assicurarsi di soddisfare i [requisiti](#) per il connettore.

Per il requisito del ruolo di gruppo, è necessario configurare il ruolo in modo da concedere le autorizzazioni necessarie e assicurarsi che il ruolo sia stato aggiunto al gruppo. Per ulteriori informazioni, consulta [the section called “Gestione del ruolo del gruppo \(console\)”](#) o [the section called “Gestire il ruolo del gruppo \(CLI\)”](#).

2. Crea e pubblica una funzione Lambda che invia dati di input al connettore.

Salvare il [codice di esempio](#) come file PY. Scarica e decomprimi il [AWS IoT Greengrass Core SDK per Python](#). Quindi, crea un pacchetto zip che contiene il file PY e la cartella greengrasssdk a livello root. Questo pacchetto zip è il pacchetto di distribuzione in cui carichi AWS Lambda

Dopo aver creato la funzione Python 3.7 Lambda, pubblica una versione della funzione e crea un alias.

3. Configurare il gruppo Greengrass.
 - a. Aggiungi la funzione Lambda tramite il relativo alias (consigliato). Configura il ciclo di vita Lambda come longevo (o nella "Pinned": true CLI).
 - b. Aggiungere il connettore e configurarne i relativi [parametri](#).
 - c. Aggiungere sottoscrizioni che consentono al connettore di ricevere [i dati di input JSON](#) e inviare [i dati di output](#) nei filtri degli argomenti supportati.
 - Imposta la funzione Lambda come origine, il connettore come destinazione e utilizza un filtro per argomenti di input supportato.
 - Imposta il connettore come origine, AWS IoT Core come destinazione e utilizza un filtro per l'argomento di output supportato. Utilizzi questo abbonamento per visualizzare i messaggi di stato nella AWS IoT console.
4. Distribuisci il gruppo.
5. Nella AWS IoT console, nella pagina Test, sottoscrivi l'argomento relativo ai dati di output per visualizzare i messaggi di stato dal connettore. La funzione Lambda di esempio è di lunga durata e inizia a inviare messaggi subito dopo l'implementazione del gruppo.

Al termine del test, puoi impostare il ciclo di vita Lambda su richiesta (o nella CLI) e "Pinned": false distribuire il gruppo. Ciò impedisce alla funzione di inviare messaggi.

Esempio

L'esempio seguente della funzione Lambda invia un messaggio di input al connettore. Questo messaggio contiene dati JSON.

```
import greengrasssdk
import time
import json
```

```
iot_client = greengrasssdk.client('iot-data')
send_topic = 'kinesisfirehose/message'

def create_request_with_all_fields():
    return {
        "request": {
            "data": "Message from Firehose Connector Test"
        },
        "id" : "req_123"
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
        payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

Licenze

Il connettore Kinesis Firehose include i seguenti software/licenze di terze parti:

- [AWS SDK per Python \(Boto3\)](#)/Apache License 2.0
- [botocore](#)/Apache License 2.0
- [dateutil](#)/PSF License
- [docutils](#)/BSD License, GNU General Public License (GPL), Python Software Foundation License, Public Domain
- [jmespath](#)/MIT License
- [s3transfer](#)/Apache License 2.0
- [urllib3](#)/MIT License

Questo connettore è rilasciato ai sensi del contratto di [licenza del software Greengrass Core](#).

Changelog

La tabella seguente descrive le modifiche apportate a ciascuna versione del connettore.

Versione	Modifiche
5	Aggiunto il parametro <code>IsolationMode</code> per configurare la modalità di containerizzazione per il connettore.
4	È stato aggiornato il runtime Lambda a Python 3.7, che modifica i requisiti di runtime.
3	Correzione per ridurre la registrazione eccessiva e altre correzioni di bug minori.
2	È stato aggiunto il supporto per l'invio di record di dati in batch a Firehose a intervalli specificati. • Nel ruolo del gruppo è necessaria anche l'operazione <code>firehose:PutRecordBatch</code>. • Nuovi parametri <code>MemorySize</code>, <code>DeliveryStreamQueueSize</code> e <code>PublishInterval</code>. • Il messaggio di output contiene una serie di risposte di stato per i record di dati pubblicati.
1	Versione iniziale.

Un gruppo Greengrass può contenere una sola versione del connettore alla volta. Per informazioni sull'aggiornamento di una versione del connettore, consulta [the section called “Aggiornamento delle versioni dei connettori”](#).

Consulta anche

- [Integrazione con servizi e protocolli tramite i connettori](#)
- [the section called “Nozioni di base sui connettori \(console\)”](#)
- [the section called “Nozioni di base sui connettori \(CLI\)”](#)
- [Cos'è Amazon Kinesis Data Firehose?](#) nella Amazon Kinesis Developer Guide

Connettore ML Feedback

Warning

Questo connettore è entrato nella fase di vita prolungata e AWS IoT Greengrass non rilascerà aggiornamenti che forniscano funzionalità, miglioramenti alle funzionalità esistenti, patch di sicurezza o correzioni di bug. Per ulteriori informazioni, consulta [AWS IoT Greengrass Version 1 politica di manutenzione](#).

Il connettore ML Feedback semplifica l'accesso ai dati del modello di machine learning (ML) per la riqualificazione e l'analisi dei modelli. Il connettore:

- Carica i dati di input (esempi) utilizzati dal tuo modello ML su Amazon S3. L'input del modello può essere in qualsiasi formato, ad esempio immagini, JSON o audio. Dopo che gli esempi sono stati caricati nel cloud, puoi utilizzarli per eseguire un nuovo training del modello per migliorare l'accuratezza e la precisione delle sue previsioni. Ad esempio, puoi utilizzare [SageMaker AI Ground Truth](#) per etichettare i tuoi campioni e [SageMaker intelligenza artificiale](#) per riqualificare il modello.
- Pubblica i risultati delle previsioni ricevuti dal modello come messaggi MQTT. In questo modo puoi monitorare e analizzare la qualità dell'inferenza del modello in tempo reale. Puoi anche archiviare i risultati delle previsioni e utilizzarli per analizzare le tendenze nel corso del tempo.
- Pubblica metriche sui caricamenti e sui dati di esempio su Amazon. CloudWatch

Per configurare questo connettore, descrivi le configurazioni di feedback supportate in formato JSON. [Una configurazione di feedback definisce proprietà come il bucket Amazon S3 di destinazione, il tipo di contenuto e la strategia di campionamento.](#) (Una strategia di campionamento viene utilizzata per determinare quali esempi caricare.)

Puoi utilizzare il connettore ML Feedback nei seguenti scenari:

- Con funzioni Lambda definite dall'utente. Le funzioni Lambda di inferenza locale utilizzano il Machine AWS IoT Greengrass Learning SDK per richiamare questo connettore e trasmettere la configurazione del feedback di destinazione, l'input e l'output del modello (risultati di previsione). Per vedere un esempio, consulta [the section called “Esempio di utilizzo”](#).

- Con il connettore [ML Image Classification](#) (v2). Per utilizzare questo connettore con il connettore ML Image Classification, configura il `MLFeedbackConnectorConfigId` parametro per il connettore ML Image Classification.
- Con il [connettore ML Object Detection](#). Per utilizzare questo connettore con il connettore ML Object Detection, configura il `MLFeedbackConnectorConfigId` parametro per il connettore ML Object Detection.

ARN: `arn:aws:greengrass:region::/connectors/MLFeedback/versions/1`

Requisiti

Questo connettore presenta i seguenti requisiti:

- AWS IoT Greengrass Core Software v1.9.3 o versione successiva.
- [Python](#) versione 3.7 o 3.8 installata sul dispositivo principale e aggiunta alla variabile di ambiente `PATH`.

Note

Per usare Python 3.8, esegui il seguente comando per creare un collegamento simbolico dalla cartella di installazione predefinita di Python 3.7 ai binari Python 3.8 installati.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Questo configura il dispositivo in modo che soddisfi il requisito Python per AWS IoT Greengrass.

- Uno o più bucket Amazon S3. Il numero di bucket utilizzati dipende dalla strategia di campionamento.
- Il [ruolo del gruppo Greengrass](#) è configurato per consentire l'`s3:PutObject` azione sugli oggetti nel bucket Amazon S3 di destinazione, come illustrato nel seguente esempio di policy IAM.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:PutObject",
```

```
    "Resource": [
      "arn:aws:s3:::bucket-name/*"
    ]
  }
]
```

La policy deve includere tutti i bucket di destinazione come risorse. Puoi concedere alle risorse un accesso granulare o condizionale (ad esempio, utilizzando uno schema di denominazione con il carattere jolly *).

Per il requisito del ruolo di gruppo, è necessario configurare il ruolo in modo da concedere le autorizzazioni necessarie e assicurarsi che il ruolo sia stato aggiunto al gruppo. Per ulteriori informazioni, consulta [the section called “Gestione del ruolo del gruppo \(console\)”](#) o [the section called “Gestire il ruolo del gruppo \(CLI\)”](#).

- Il [connettore CloudWatch Metrics](#) è stato aggiunto al gruppo Greengrass e configurato. Questo è obbligatorio solo se desideri utilizzare la funzionalità di creazione di report dei parametri.
- AWS IoT Greengrass Per interagire con questo connettore è necessario [Machine Learning SDK v1.1.0](#).

Parametri

FeedbackConfigurationMap

Un set di una o più configurazioni di feedback che il connettore può utilizzare per caricare campioni su Amazon S3. Una configurazione di feedback definisce proprietà quali il bucket di destinazione, il tipo di contenuti e la [strategia di campionamento](#). Quando questo connettore viene richiamato, la funzione o il connettore Lambda chiamante specifica una configurazione di feedback di destinazione.

Nome visualizzato nella AWS IoT console: mappa di configurazione del feedback

Obbligatorio: true

Tipo: una stringa JSON ben formata che definisce l'insieme di configurazioni di feedback supportate. Per vedere un esempio, consulta [the section called “FeedbackConfigurationMap esempio”](#).

L'ID di un oggetto di configurazione di feedback dispone dei seguenti requisiti.

L'ID:

- Deve essere univoco tra gli oggetti di configurazione.
- Deve iniziare con una lettera o un numero. Può contenere lettere minuscole e maiuscole, numeri e trattini.
- Deve avere una lunghezza compresa tra 2 e 63 caratteri.

Obbligatorio: `true`

Tipo: `string`

Schema valido: `^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$`

Esempi: `MyConfig0`, `config-a`, `12id`

Il corpo di un oggetto di configurazione di feedback contiene le seguenti proprietà.

`s3-bucket-name`

Il nome del bucket Amazon S3 di destinazione.

 Note

Il ruolo del gruppo deve consentire l'operazione `s3:PutObject` su tutti i bucket di destinazione. Per ulteriori informazioni, consulta [the section called “Requisiti”](#).

Obbligatorio: `true`

Tipo: `string`

Schema valido: `^[a-z0-9\.\-]{3,63}$`

`content-type`

Il tipo di contenuto dei campioni da caricare. Tutti i contenuti di una singola configurazione di feedback devono essere dello stesso tipo.

Richiesto: `true`

Tipo: `string`

Esempi: `image/jpeg`, `application/json`, `audio/ogg`

s3-prefix

Il prefisso della chiave da utilizzare per i campioni caricati. Un prefisso è simile al nome di una directory. Consente di archiviare dati simili nella stessa directory in un bucket. Per ulteriori informazioni, consulta la [chiave dell'oggetto e i metadati](#) nella Guida per l'utente di Amazon Simple Storage Service.

Obbligatorio: false

Tipo: string

file-ext

L'estensione da utilizzare per i campioni caricati. Deve essere un'estensione file valida per il tipo di contenuti.

Richiesto: false

Tipo: string

Esempi: jpg, json, ogg

sampling-strategy

La [strategia di campionamento](#) da utilizzare per filtrare i campioni da caricare. Se omessa, il connettore tenta di caricare tutti i campioni che riceve.

Richiesto: false

Tipo: una stringa JSON ben formata che contiene le seguenti proprietà.

strategy-name

Il nome della strategia di campionamento.

Obbligatorio: true

Tipo: string

Valori validi: RANDOM_SAMPLING, LEAST_CONFIDENCE, MARGIN o ENTROPY

rate

La velocità della strategia di campionamento [causale](#).

Obbligatorio: true se lo strategy-name è RANDOM_SAMPLING.

Tipo: number

Valori validi: 0.0 - 1.0

threshold

La soglia per la strategia di campionamento [Fiducia minima](#), [Margine](#) o [Entropia](#).

Obbligatorio: true se strategy-name è LEAST_CONFIDENCEMARGIN, oENTROPY.

Tipo: number

Valori validi:

- 0.0 - 1.0 per la strategia LEAST_CONFIDENCE o MARGIN.
- 0.0 - no limit per la strategia ENTROPY.

RequestLimit

Il numero massimo di richieste che il connettore può elaborare alla volta.

Puoi utilizzare questo parametro per limitare il consumo di memoria limitando il numero di richieste che il connettore elabora nello stesso momento. Le richieste che superano questo limite vengono ignorate.

Nome visualizzato nella AWS IoT console: Request limit

Obbligatorio: false

Tipo: string

Valori validi: 0 - 999

Schema valido: ^\$|^[0-9]{1,3}\$

Esempio di creazione di un connettore (AWS CLI)

Il seguente comando CLI crea un file ConnectorDefinition con una versione iniziale che contiene il connettore ML Feedback.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
```

```

"Connectors": [
  {
    "Id": "MyMLFeedbackConnector",
    "ConnectorArn": "arn:aws:greengrass:region::/connectors/MLFeedback/
versions/1",
    "Parameters": {
      "FeedbackConfigurationMap": "{ \"RandomSamplingConfiguration\":
{ \"s3-bucket-name\": \"my-aws-bucket-random-sampling\", \"content-type\":
\"image/png\", \"file-ext\": \"png\", \"sampling-strategy\": { \"strategy-name
\": \"RANDOM_SAMPLING\", \"rate\": 0.5 } }, \"LeastConfidenceConfiguration\": {
  \"s3-bucket-name\": \"my-aws-bucket-least-confidence-sampling\", \"content-type\":
  \"image/png\", \"file-ext\": \"png\", \"sampling-strategy\": { \"strategy-name\":
  \"LEAST_CONFIDENCE\", \"threshold\": 0.4 } } }",
      "RequestLimit": "10"
    }
  }
]
}'

```

FeedbackConfigurationMap esempio

Di seguito è riportato un valore di esempio espanso per il parametro FeedbackConfigurationMap. Questo esempio include diverse configurazioni di feedback che utilizzano varie strategie di campionamento.

```

{
  "ConfigID1": {
    "s3-bucket-name": "my-aws-bucket-random-sampling",
    "content-type": "image/png",
    "file-ext": "png",
    "sampling-strategy": {
      "strategy-name": "RANDOM_SAMPLING",
      "rate": 0.5
    }
  },
  "ConfigID2": {
    "s3-bucket-name": "my-aws-bucket-margin-sampling",
    "content-type": "image/png",
    "file-ext": "png",
    "sampling-strategy": {
      "strategy-name": "MARGIN",
      "threshold": 0.4
    }
  }
}

```



```
    },
    "ConfigID3": {
      "s3-bucket-name": "my-aws-bucket-least-confidence-sampling",
      "content-type": "image/png",
      "file-ext": "png",
      "sampling-strategy": {
        "strategy-name": "LEAST_CONFIDENCE",
        "threshold": 0.4
      }
    },
    "ConfigID4": {
      "s3-bucket-name": "my-aws-bucket-entropy-sampling",
      "content-type": "image/png",
      "file-ext": "png",
      "sampling-strategy": {
        "strategy-name": "ENTROPY",
        "threshold": 2
      }
    },
    "ConfigID5": {
      "s3-bucket-name": "my-aws-bucket-no-sampling",
      "s3-prefix": "DeviceA",
      "content-type": "application/json"
    }
  }
}
```

Strategie di campionamento

Il connettore supporta quattro strategie di campionamento che determinano se caricare i campioni che vengono passati al connettore. Gli esempi sono istanze discrete di dati utilizzati da un modello per una previsione. Puoi utilizzare strategie di campionamento per filtrare i campioni che è più probabile migliorino l'accuratezza del modello.

RANDOM_SAMPLING

Carica in modo casuale campioni in base alla velocità fornita. Carica un campione se un valore generato casualmente è inferiore alla velocità. Più alta è la velocità, maggiore è il numero di campioni caricati.

Note

Questa strategia ignora qualsiasi previsione del modello fornita.

LEAST_CONFIDENCE

Carica esempi la cui probabilità di affidabilità massima scende al di sotto della soglia fornita.

Scenario di esempio:

Soglia: .6

Previsione del modello: [.2, .2, .4, .2]

Probabilità di affidabilità massima: .4

Risultato:

Utilizza il campione perché la probabilità di affidabilità massima (.4) $\leq$ soglia (.6).

MARGIN

Carica esempi se il margine tra le prime due probabilità di affidabilità principali rientra nella soglia fornita. Il margine è la differenza tra le prime due probabilità principali.

Scenario di esempio:

Soglia: .02

Previsione del modello: [.3, .35, .34, .01]

Le due probabilità di affidabilità principali: [.35, .34]

Margine: .01 (.35 - .34)

Risultato:

Utilizza il campione perché margine (.01) $\leq$ soglia (.02).

ENTROPY

Carica campioni la cui entropia è maggiore della soglia specificata. Utilizza l'entropia normalizzata della previsione del modello.

Scenario di esempio:

Soglia: 0.75

Previsione del modello: [.5, .25, .25]

Entropia per la previsione: 1.03972

Risultato:

Utilizza campione perché entropia (1.03972) > soglia (0.75).

Dati di input

Le funzioni Lambda definite dall'utente utilizzano `publish` la funzione del client feedback nel Machine AWS IoT Greengrass Learning SDK per richiamare il connettore. Per vedere un esempio, consulta [the section called “Esempio di utilizzo”](#).

Note

Questo connettore non accetta messaggi MQTT come dati di input.

La funzione `publish` accetta i seguenti argomenti:

ConfigId

L'ID della configurazione di feedback di destinazione. Questo deve corrispondere all'ID di una configurazione di feedback definita nel [FeedbackConfigurationMap](#) parametro per il connettore ML Feedback.

Obbligatorio: true

Tipo: stringa

ModelInput

I dati di input che sono stati passati a un modello per l'inferenza. Questi dati di input vengono caricati utilizzando la configurazione di destinazione, a meno che non vengano filtrati in base alla strategia di campionamento.

Obbligatorio: true

Tipo: byte

ModelPrediction

I risultati delle previsioni dal modello. Il tipo di risultato può essere un dizionario o un elenco. Ad esempio, i risultati della previsione del connettore ML Image Classification sono un elenco di probabilità (ad esempio). [0.25, 0.60, 0.15] Questi dati vengono pubblicati nell'argomento / `feedback/message/prediction`.

Obbligatorio: true

Tipo: dizionario o elenco di float valori

Metadati

Metadati specifici dell'applicazione definiti dal cliente collegati al campione caricato e pubblicati nell'argomento `/feedback/message/prediction`. Il connettore inserisce inoltre una chiave `publish-ts` con un valore di timestamp nei metadati.

Obbligatorio: false

Tipo: dizionario

Esempio: `{"some-key": "some value"}`

Dati di output

Questo connettore pubblica i dati in tre argomenti MQTT:

- Le informazioni sullo stato del connettore nell'argomento `feedback/message/status`.
- Risultati delle previsioni nell'argomento `feedback/message/prediction`.
- Metriche destinate all' CloudWatch `cloudwatch/metric/putargomento`.

È necessario configurare le sottoscrizioni per consentire al connettore di comunicare su argomenti MQTT. Per ulteriori informazioni, consulta [the section called “Input e output”](#).

Filtro di argomenti: `feedback/message/status`

Utilizza questo argomento per monitorare lo stato dei caricamenti di campioni e dei campioni rimossi. Il connettore pubblica in questo argomento ogni volta che riceve una richiesta.

Output di esempio: caricamento campione riuscito

```
{
  "response": {
    "status": "success",
    "s3_response": {
      "ResponseMetadata": {
        "HostId": "I0WQ4fDEXAMPLEQM+ey7N9WgVhSnQ6JEXAMPLEZb7hSQDASK
+Jd1vEXAMPLEa3Km",
        "RetryAttempts": 1,
```

```

    "HTTPStatusCode": 200,
    "RequestId": "79104EXAMPLEB723",
    "HTTPHeaders": {
      "content-length": "0",
      "x-amz-id-2":
"1bbqaDVF0hMlyU3gRvAX1ZIdg8P0WkGkCSSFsYFvSwLZk3j7QZhG5EXAMPLEdd4/pEXAMPLEUqU=",
      "server": "AmazonS3",
      "x-amz-expiration": "expiry-date=\"Wed, 17 Jul 2019 00:00:00 GMT\",
rule-id=\"OGZjYWY3OTgtYWl2Zi00ZDl1LWE4YmQtNzMyYzEXAMPLEoUw\"",
      "x-amz-request-id": "79104EXAMPLEB723",
      "etag": "\"b9c4f172e64458a5fd674EXAMPLE5628\"",
      "date": "Thu, 11 Jul 2019 00:12:50 GMT",
      "x-amz-server-side-encryption": "AES256"
    }
  },
  "bucket": "greengrass-feedback-connector-data-us-west-2",
  "ETag": "\"b9c4f172e64458a5fd674EXAMPLE5628\"",
  "Expiration": "expiry-date=\"Wed, 17 Jul 2019 00:00:00 GMT\", rule-id=
\"OGZjYWY3OTgtYWl2Zi00ZDl1LWE4YmQtNzMyYzEXAMPLEoUw\"",
  "key": "s3-key-prefix/UUID.file_ext",
  "ServerSideEncryption": "AES256"
}
},
"id": "5aaa913f-97a3-48ac-5907-18cd96b89eeb"
}

```

Il connettore aggiunge i key campi bucket e alla risposta di Amazon S3. Per ulteriori informazioni sulla risposta di Amazon S3, consulta l'[oggetto PUT](#) nel riferimento all'API di Amazon Simple Storage Service.

Output di esempio: campione rimosso a causa della strategia di campionamento

```

{
  "response": {
    "status": "sample_dropped_by_strategy"
  },
  "id": "4bf5aeb0-d1e4-4362-5bb4-87c05de78ba3"
}

```

Output di esempio: caricamento campione non riuscito

Uno stato di errore include il messaggio di errore come il valore `error_message` e la classe di eccezione come il valore `error`.

```
{
  "response": {
    "status": "fail",
    "error_message": "[RequestId: 4bf5aeb0-d1e4-4362-5bb4-87c05de78ba3] Failed
to upload model input data due to exception. Model prediction will not be
published. Exception type: NoSuchBucket, error: An error occurred (NoSuchBucket)
when calling the PutObject operation: The specified bucket does not exist",
    "error": "NoSuchBucket"
  },
  "id": "4bf5aeb0-d1e4-4362-5bb4-87c05de78ba3"
}
```

Output di esempio: richiesta con throttling a causa del limite di richieste

```
{
  "response": {
    "status": "fail",
    "error_message": "Request limit has been reached (max request: 10 ). Dropping
request.",
    "error": "Queue.Full"
  },
  "id": "4bf5aeb0-d1e4-4362-5bb4-87c05de78ba3"
}
```

Filtro di argomenti: feedback/message/prediction

Utilizza questo argomento per ascoltare le previsioni in base ai dati campione caricati. Questo consente di analizzare le prestazioni del modello in tempo reale. Le previsioni dei modelli vengono pubblicate su questo argomento solo se i dati vengono caricati correttamente su Amazon S3. I messaggi pubblicati in questo argomento sono in formato JSON. Contengono il collegamento all'oggetto dati caricato, la previsione del modello e i metadati inclusi nella richiesta.

Puoi anche archiviare i risultati delle previsioni e utilizzarli per segnalare e analizzare le tendenze nel tempo. Le tendenze possono fornire informazioni preziose. Ad esempio, una tendenza precisione decrescente nel tempo consente di decidere se occorre eseguire un nuovo training del modello.

Output di esempio

```
{
  "source-ref": "s3://greengrass-feedback-connector-data-us-west-2/s3-key-prefix/
UUID.file_ext",
```

```
"model-prediction": [
  0.5,
  0.2,
  0.2,
  0.1
],
"config-id": "ConfigID2",
"metadata": {
  "publish-ts": "2019-07-11 00:12:48.816752"
}
}
```

 Tip

Puoi configurare il [connettore IoT Analytics](#) per iscriverti a questo argomento e inviare le informazioni a AWS IoT Analytics per ulteriori analisi o analisi cronologiche.

Filtro di argomenti: cloudwatch/metric/put

Questo è l'argomento di output utilizzato per pubblicare i parametri in CloudWatch. Questa funzionalità richiede l'installazione e la configurazione del [connettore CloudWatch Metrics](#).

I parametri includono:

- Il numero di campioni caricati.
- Le dimensioni dei campioni caricati.
- Il numero di errori causati dai caricamenti su Amazon S3.
- Il numero di campioni rimossi in base alla strategia di campionamento.
- Il numero di richieste sottoposte a throttling.

Output di esempio: dimensioni del campione di dati (pubblicato prima del caricamento effettivo)

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 47592,
      "unit": "Bytes",
      "metricName": "SampleSize"
    }
  }
}
```

```
}  
}  
}
```

Output di esempio: caricamento campione riuscito

```
{  
  "request": {  
    "namespace": "GreengrassFeedbackConnector",  
    "metricData": {  
      "value": 1,  
      "unit": "Count",  
      "metricName": "SampleUploadSuccess"  
    }  
  }  
}
```

Output di esempio: caricamento campione riuscito e risultato delle previsioni pubblicato

```
{  
  "request": {  
    "namespace": "GreengrassFeedbackConnector",  
    "metricData": {  
      "value": 1,  
      "unit": "Count",  
      "metricName": "SampleAndPredictionPublished"  
    }  
  }  
}
```

Output di esempio: caricamento campione non riuscito

```
{  
  "request": {  
    "namespace": "GreengrassFeedbackConnector",  
    "metricData": {  
      "value": 1,  
      "unit": "Count",  
      "metricName": "SampleUploadFailure"  
    }  
  }  
}
```


Output di esempio: campione rimosso a causa della strategia di campionamento

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "SampleNotUsed"
    }
  }
}
```

Output di esempio: richiesta con throttling a causa del limite di richieste

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "ErrorRequestThrottled"
    }
  }
}
```

Esempio di utilizzo

L'esempio seguente è una funzione Lambda definita dall'utente che utilizza [Machine AWS IoT Greengrass Learning](#) SDK per inviare dati al connettore ML Feedback.

Note

Puoi scaricare il AWS IoT Greengrass Machine Learning SDK dalla [pagina dei AWS IoT Greengrass download](#).

```
import json
import logging
import os
```

```
import sys
import greengrass_machine_learning_sdk as ml

client = ml.client('feedback')

try:
    feedback_config_id = os.environ["FEEDBACK_CONFIG_ID"]
    model_input_data_dir = os.environ["MODEL_INPUT_DIR"]
    model_prediction_str = os.environ["MODEL_PREDICTIONS"]
    model_prediction = json.loads(model_prediction_str)
except Exception as e:
    logging.info("Failed to open environment variables. Failed with exception:
{}".format(e))
    sys.exit(1)

try:
    with open(os.path.join(model_input_data_dir, os.listdir(model_input_data_dir)[0]),
'rb') as f:
        content = f.read()
except Exception as e:
    logging.info("Failed to open model input directory. Failed with exception:
{}".format(e))
    sys.exit(1)

def invoke_feedback_connector():
    logging.info("Invoking feedback connector.")
    try:
        client.publish(
            ConfigId=feedback_config_id,
            ModelInput=content,
            ModelPrediction=model_prediction
        )
    except Exception as e:
        logging.info("Exception raised when invoking feedback connector:{}".format(e))
        sys.exit(1)

invoke_feedback_connector()

def function_handler(event, context):
    return
```

Licenze

Il connettore ML Feedback include i seguenti software/licenze di terze parti:

- [AWS SDK per Python \(Boto3\)](#)/Apache License 2.0
- [botocore](#)/Apache License 2.0
- [dateutil](#)/PSF License
- [docutils](#)/BSD License, GNU General Public License (GPL), Python Software Foundation License, Public Domain
- [jmespath](#)/MIT License
- [s3transfer](#)/Apache License 2.0
- [urllib3](#)/MIT License

- [six](#)/MIT

Questo connettore è rilasciato ai sensi del contratto di [licenza del software Greengrass Core](#).

Consulta anche

- [Integrazione con servizi e protocolli tramite i connettori](#)
- [the section called “Nozioni di base sui connettori \(console\)”](#)
- [the section called “Nozioni di base sui connettori \(CLI\)”](#)

Connettore ML Image Classification

Warning

Questo connettore è entrato nella fase di vita prolungata e AWS IoT Greengrass non rilascerà aggiornamenti che forniscano funzionalità, miglioramenti alle funzionalità esistenti, patch di sicurezza o correzioni di bug. Per ulteriori informazioni, consulta [AWS IoT Greengrass Version 1 politica di manutenzione](#).

I [connettori](#) ML Image Classification forniscono un servizio di inferenza di machine learning (ML) che viene eseguito sul core. AWS IoT Greengrass Questo servizio di inferenza locale esegue la classificazione delle immagini utilizzando un modello addestrato dall'algoritmo di classificazione delle immagini SageMaker AI.

Le funzioni Lambda definite dall'utente utilizzano il Machine AWS IoT Greengrass Learning SDK per inviare richieste di inferenza al servizio di inferenza locale. Il servizio esegue l'inferenza in locale e restituisce le probabilità che l'immagine di input appartenga a categorie specifiche.

AWS IoT Greengrass fornisce le seguenti versioni di questo connettore, disponibile per più piattaforme.

Version 2

Connector	Descrizione e ARN
Classificazione delle immagini ML Aarch64 JTX2	Servizio di inferenza per la classificazione delle immagini per NVIDIA Jetson. TX2 Supporta l'accelerazione GPU. ARN: <code>arn:aws:greengrass: <i>region</i>:/connectors/ImageClassificationAarch64JTX2/versions/2</code>
Classificazione delle immagini ML x86_64	Servizio di inferenza di classificazione delle immagini per le piattaforme x86_64. ARN: <code>arn:aws:greengrass: <i>region</i>:/connectors/ImageClassificationx86-64/versions/2</code>
Classificazione delle immagini ML ARMv7	Servizio di inferenza per la classificazione delle immagini per ARMv7 piattaforme. ARN: <code>arn:aws:greengrass: <i>region</i>:/connectors/ImageClassificationARMv7/versions/2</code>

Version 1

Connector	Descrizione e ARN
Classificazione delle immagini ML Aarch64 JTX2	Servizio di inferenza per la classificazione delle immagini per NVIDIA Jetson. TX2 Supporta l'accelerazione GPU. ARN: <code>arn:aws:greengrass: : <i>region</i>::/connectors/ImageClassificationAarch64JTX2/ versions/1</code>
Classificazione delle immagini ML x86_64	Servizio di inferenza di classificazione delle immagini per le piattaforme x86_64. ARN: <code>arn:aws:greengrass: : <i>region</i>::/connectors/ImageClassificationx86-64/versi ons/1</code>
Classificazione delle immagini ML Armv7	Servizio di inferenza di classificazione delle immagini per le piattaforme Armv7. ARN: <code>arn:aws:greengrass: : <i>region</i>::/connectors/ImageClassificationARMv7/versio ns/1</code>

Per informazioni sulle modifiche di ogni versione, consulta [Changelog](#).

Requisiti

Questi connettori presentano i seguenti requisiti:

Version 2

- AWS IoT Greengrass Core Software v1.9.3 o versione successiva.

- [Python](#) versione 3.7 o 3.8 installata sul dispositivo principale e aggiunta alla variabile di ambiente PATH.

 Note

Per usare Python 3.8, esegui il seguente comando per creare un collegamento simbolico dalla cartella di installazione predefinita di Python 3.7 ai binari Python 3.8 installati.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Questo configura il dispositivo in modo che soddisfi il requisito Python per AWS IoT Greengrass.

- Dipendenze per il framework Apache installato sul dispositivo principale. MXNet Per ulteriori informazioni, consulta [the section called “Installazione MXNet delle dipendenze”](#).
- Una [risorsa ML](#) del gruppo Greengrass che fa riferimento a una fonte di modelli di SageMaker intelligenza artificiale. Questo modello deve essere addestrato dall'algoritmo di classificazione delle immagini SageMaker AI. Per ulteriori informazioni, consulta [Algoritmo di classificazione delle immagini](#) nella Amazon SageMaker AI Developer Guide.
- Il [connettore ML Feedback](#) aggiunto al gruppo Greengrass e configurato. Questo è obbligatorio solo se desideri utilizzare il connettore per caricare i dati di input del modello e pubblicare le previsioni in un argomento MQTT.
- Il [ruolo del gruppo Greengrass](#) è configurato per consentire l'`sagemaker:DescribeTrainingJob` azione sul lavoro di formazione target, come mostrato nel seguente esempio di politica IAM.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sagemaker:DescribeTrainingJob"
      ],
      "Resource": "arn:aws:sagemaker:region:account-id:training-job:training-job-name"
    }
  ]
}
```

```
]
}
```

Per il requisito del ruolo di gruppo, è necessario configurare il ruolo in modo da concedere le autorizzazioni necessarie e assicurarsi che il ruolo sia stato aggiunto al gruppo. Per ulteriori informazioni, consulta [the section called “Gestione del ruolo del gruppo \(console\)”](#) o [the section called “Gestire il ruolo del gruppo \(CLI\)”](#).

Puoi concedere alle risorse un accesso granulare o condizionale (ad esempio, utilizzando uno schema di denominazione con il carattere jolly *). Se in futuro cambierai il tipo di formazione desiderato, assicurati di aggiornare il ruolo del gruppo.

- AWS IoT Greengrass Per interagire con questo connettore è necessario [Machine Learning SDK v1.1.0](#).

Version 1

- AWS IoT Greengrass Core Software v1.7 o versione successiva.
- [Python](#) versione 2.7 installato sul dispositivo principale e aggiunto alla variabile di ambiente PATH.
- Dipendenze per il MXNet framework Apache installato sul dispositivo principale. Per ulteriori informazioni, consulta [the section called “Installazione MXNet delle dipendenze”](#).
- Una [risorsa ML](#) del gruppo Greengrass che fa riferimento a una fonte di modelli di SageMaker intelligenza artificiale. Questo modello deve essere addestrato dall'algoritmo di classificazione delle immagini SageMaker AI. Per ulteriori informazioni, consulta [Algoritmo di classificazione delle immagini](#) nella Amazon SageMaker AI Developer Guide.
- Il [ruolo del gruppo Greengrass](#) è configurato per consentire l'`sagemaker:DescribeTrainingJob` azione sul lavoro di formazione target, come mostrato nel seguente esempio di politica IAM.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sagemaker:DescribeTrainingJob"
      ],
    },
  ],
}
```

```
        "Resource": "arn:aws:sagemaker:region:account-id:training-  
job:training-job-name"  
    }  
]  
}
```

Per il requisito del ruolo di gruppo, è necessario configurare il ruolo in modo da concedere le autorizzazioni necessarie e assicurarsi che il ruolo sia stato aggiunto al gruppo. Per ulteriori informazioni, consulta [the section called “Gestione del ruolo del gruppo \(console\)”](#) o [the section called “Gestire il ruolo del gruppo \(CLI\)”](#).

Puoi concedere alle risorse un accesso granulare o condizionale (ad esempio, utilizzando uno schema di denominazione con il carattere jolly *). Se in futuro cambierai il tipo di formazione desiderato, assicurati di aggiornare il ruolo del gruppo.

- AWS IoT Greengrass Per interagire con questo connettore è necessario [Machine Learning SDK v1.0.0](#) o versione successiva.

Parametri del connettore

Questi connettori forniscono i seguenti parametri.

Version 2

MLModelDestinationPath

Il percorso locale assoluto della risorsa ML all'interno dell'ambiente Lambda. Si tratta del percorso di destinazione specificato per la risorsa ML.

Note

Se hai creato la risorsa ML nella console, si tratta del percorso locale.

Nome visualizzato nella AWS IoT console: percorso di destinazione del modello

Obbligatorio: true

Tipo: string

Modello valido: . +

MLModelResourceId

L'ID della risorsa ML che fa riferimento al modello di origine.

Nome visualizzato nella AWS IoT console: risorsa SageMaker ARN del lavoro

Obbligatorio: true

Tipo: string

Modello valido: [a-zA-Z0-9:_-]+

MLModelSageMakerJobArn

L'ARN del job di formazione sull' SageMaker intelligenza artificiale che rappresenta la fonte del modello SageMaker AI. Il modello deve essere addestrato dall'algoritmo di classificazione delle immagini SageMaker AI.

Nome visualizzato nella AWS IoT console: SageMaker job ARN

Richiesto: true

Tipo: string

Modello valido: ^arn:aws:sagemaker:[a-zA-Z0-9-]+:[0-9]+:training-job/[a-zA-Z0-9-]+[a-zA-Z0-9-]+\$

LocalInferenceServiceName

Il nome del servizio di inferenza locale. Le funzioni Lambda definite dall'utente richiamano il servizio passando il nome alla funzione `invoke_inference_service` del Machine AWS IoT Greengrass Learning SDK. Per vedere un esempio, consulta [the section called “Esempio di utilizzo”](#).

Nome visualizzato nella AWS IoT console: nome del servizio di inferenza locale

Obbligatorio: true

Tipo: string

Modello valido: [a-zA-Z0-9][a-zA-Z0-9-]{1,62}

LocalInferenceServiceTimeoutSeconds

L'intervallo di tempo (in secondi) prima che la richiesta di inferenza venga terminata. Il valore minimo è 1.

Nome visualizzato nella AWS IoT console: Timeout (secondo)

Richiesto: true

Tipo: string

Modello valido: [1-9][0-9]*

LocalInferenceServiceMemoryLimitKB

La quantità di memoria (in KB) a cui ha accesso il servizio. Il valore minimo è 1.

Nome visualizzato nella AWS IoT console: limite di memoria (KB)

Richiesto: true

Tipo: string

Modello valido: [1-9][0-9]*

GPUAcceleration

Il contesto di calcolo della CPU o GPU (accelerata). Questa proprietà si applica solo al JTX2 connettore ML Image Classification Aarch64.

Nome visualizzato nella AWS IoT console: accelerazione GPU

Richiesto: true

Tipo: string

Valori validi: CPU o GPU

MLFeedbackConnectorConfigId

L'ID della configurazione di feedback da utilizzare per caricare i dati di input del modello. Deve corrispondere all'ID di una configurazione di feedback definita per il [connettore ML Feedback](#).

Questo parametro è obbligatorio solo se desideri utilizzare il connettore ML Feedback per caricare i dati di input del modello e pubblicare le previsioni in un argomento MQTT.

Nome visualizzato nella AWS IoT console: ID di configurazione del connettore ML Feedback

Obbligatorio: false

Tipo: string

Modello valido: `^$|^([a-zA-Z0-9][a-zA-Z0-9-]){1,62}$`

Version 1

MLModelDestinationPath

Il percorso locale assoluto della risorsa ML all'interno dell'ambiente Lambda. Si tratta del percorso di destinazione specificato per la risorsa ML.

Note

Se hai creato la risorsa ML nella console, si tratta del percorso locale.

Nome visualizzato nella AWS IoT console: percorso di destinazione del modello

Obbligatorio: `true`

Tipo: `string`

Modello valido: `.*`

MLModelResourceId

L'ID della risorsa ML che fa riferimento al modello di origine.

Nome visualizzato nella AWS IoT console: risorsa SageMaker ARN del lavoro

Obbligatorio: `true`

Tipo: `string`

Modello valido: `[a-zA-Z0-9: _-]+`

MLModelSageMakerJobArn

L'ARN del job di formazione sull' SageMaker intelligenza artificiale che rappresenta la fonte del modello SageMaker AI. Il modello deve essere addestrato dall'algoritmo di classificazione delle immagini SageMaker AI.

Nome visualizzato nella AWS IoT console: SageMaker job ARN

Richiesto: `true`

Tipo: `string`

Modello valido: `^arn:aws:sagemaker:[a-zA-Z0-9-]+:[0-9]+:training-job/[a-zA-Z0-9-]+[a-zA-Z0-9-]+$`

`LocalInferenceServiceName`

Il nome del servizio di inferenza locale. Le funzioni Lambda definite dall'utente richiamano il servizio passando il nome alla funzione `invoke_inference_service` del Machine AWS IoT Greengrass Learning SDK. Per vedere un esempio, consulta [the section called “Esempio di utilizzo”](#).

Nome visualizzato nella AWS IoT console: nome del servizio di inferenza locale

Obbligatorio: `true`

Tipo: `string`

Modello valido: `[a-zA-Z0-9-][a-zA-Z0-9-]{1,62}`

`LocalInferenceServiceTimeoutSeconds`

L'intervallo di tempo (in secondi) prima che la richiesta di inferenza venga terminata. Il valore minimo è 1.

Nome visualizzato nella AWS IoT console: Timeout (secondo)

Richiesto: `true`

Tipo: `string`

Modello valido: `[1-9][0-9]*`

`LocalInferenceServiceMemoryLimitKB`

La quantità di memoria (in KB) a cui ha accesso il servizio. Il valore minimo è 1.

Nome visualizzato nella AWS IoT console: limite di memoria (KB)

Richiesto: `true`

Tipo: `string`

Modello valido: `[1-9][0-9]*`

GPUAcceleration

Il contesto di calcolo della CPU o GPU (accelerata). Questa proprietà si applica solo al JTX2 connettore ML Image Classification Aarch64.

Nome visualizzato nella AWS IoT console: accelerazione GPU

Richiesto: true

Tipo: string

Valori validi: CPU o GPU

Esempio di creazione di un connettore (AWS CLI)

I seguenti comandi CLI creano una `ConnectorDefinition` versione iniziale che contiene un connettore ML Image Classification.

Esempio: istanza CPU

Questo esempio crea un'istanza del connettore ML Image Classification ARMv7l.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-  
version '{  
  "Connectors": [  
    {  
      "Id": "MyImageClassificationConnector",  
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/  
ImageClassificationARMv7/versions/2",  
      "Parameters": {  
        "MLModelDestinationPath": "/path-to-model",  
        "MLModelResourceId": "my-ml-resource",  
        "MLModelSageMakerJobArn": "arn:aws:sagemaker:us-  
west-2:123456789012:training-job:MyImageClassifier",  
        "LocalInferenceServiceName": "imageClassification",  
        "LocalInferenceServiceTimeoutSeconds": "10",  
        "LocalInferenceServiceMemoryLimitKB": "500000",  
        "MLFeedbackConnectorConfigId": "MyConfig0"  
      }  
    }  
  ]  
}'
```

Esempio: istanza GPU

Questo esempio crea un'istanza del JTX2 connettore ML Image Classification Aarch64, che supporta l'accelerazione GPU su una scheda NVIDIA Jetson. TX2

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-  
version '{  
  "Connectors": [  
    {  
      "Id": "MyImageClassificationConnector",  
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/  
ImageClassificationAarch64JTX2/versions/2",  
      "Parameters": {  
        "MLModelDestinationPath": "/path-to-model",  
        "MLModelResourceId": "my-ml-resource",  
        "MLModelSageMakerJobArn": "arn:aws:sagemaker:us-  
west-2:123456789012:training-job:MyImageClassifier",  
        "LocalInferenceServiceName": "imageClassification",  
        "LocalInferenceServiceTimeoutSeconds": "10",  
        "LocalInferenceServiceMemoryLimitKB": "500000",  
        "GPUAcceleration": "GPU",  
        "MLFeedbackConnectorConfigId": "MyConfig0"  
      }  
    }  
  ]  
}'
```

Note

La funzione Lambda di questi connettori ha un ciclo di vita di [lunga durata](#).

Nella AWS IoT Greengrass console, puoi aggiungere un connettore dalla pagina Connettori del gruppo. Per ulteriori informazioni, consulta [the section called “Nozioni di base sui connettori \(console\)”](#).

Dati di input

Questi connettori accettano un file di immagine come input. I file di immagine di input devono essere in formato png o jpeg. Per ulteriori informazioni, consulta [the section called “Esempio di utilizzo”](#).

Questi connettori non accettano messaggi MQTT come dati di input.

Dati di output

Questi connettori restituiscono una previsione formattata per l'oggetto identificato nell'immagine di input:

```
[0.3,0.1,0.04,...]
```

La previsione contiene un elenco di valori che corrispondono alle categorie utilizzate nel set di dati di training durante il training del modello. Ogni valore rappresenta la probabilità che l'immagine rientri nella categoria corrispondente. La categoria con la probabilità più alta è la previsione dominante.

Questi connettori non pubblicano messaggi MQTT come dati di output.

Esempio di utilizzo

L'esempio seguente della funzione Lambda utilizza il [AWS IoT Greengrass Machine Learning SDK](#) per interagire con un connettore ML Image Classification.

Note

Puoi scaricare l'SDK dalla pagina dei download di [AWS IoT Greengrass Machine Learning SDK](#).

In questo esempio viene inizializzato un client SDK e viene chiamata in modo sincrono la funzione `invoke_inference_service` di SDK per richiamare il servizio di inferenza locale. Trasferisce il tipo di algoritmo, il nome del servizio, il tipo di immagine e il contenuto dell'immagine. Quindi, l'esempio analizza la risposta del servizio per ottenere i risultati di probabilità (previsioni).

Python 3.7

```
import logging
from threading import Timer

import numpy as np

import greengrass_machine_learning_sdk as ml

# We assume the inference input image is provided as a local file
```

```

# to this inference client Lambda function.
with open('/test_img/test.jpg', 'rb') as f:
    content = bytearray(f.read())

client = ml.client('inference')

def infer():
    logging.info('invoking Greengrass ML Inference service')

    try:
        resp = client.invoke_inference_service(
            AlgoType='image-classification',
            ServiceName='imageClassification',
            ContentType='image/jpeg',
            Body=content
        )
    except ml.GreengrassInferenceException as e:
        logging.info('inference exception {}'.format(e.__class__.__name__, e))
        return
    except ml.GreengrassDependencyException as e:
        logging.info('dependency exception {}'.format(e.__class__.__name__,
e))
        return

    logging.info('resp: {}'.format(resp))
    predictions = resp['Body'].read().decode("utf-8")
    logging.info('predictions: {}'.format(predictions))

    # The connector output is in the format: [0.3,0.1,0.04,...]
    # Remove the '[' and ']' at the beginning and end.
    predictions = predictions[1:-1]
    count = len(predictions.split(','))
    predictions_arr = np.fromstring(predictions, count=count, sep=',')

    # Perform business logic that relies on the predictions_arr, which is an array
    # of probabilities.

    # Schedule the infer() function to run again in one second.
    Timer(1, infer).start()
    return

infer()

def function_handler(event, context):

```



```
return
```

Python 2.7

```
import logging
from threading import Timer

import numpy

import greengrass_machine_learning_sdk as gg_ml

# The inference input image.
with open("/test_img/test.jpg", "rb") as f:
    content = f.read()

client = gg_ml.client("inference")

def infer():
    logging.info("Invoking Greengrass ML Inference service")

    try:
        resp = client.invoke_inference_service(
            AlgoType="image-classification",
            ServiceName="imageClassification",
            ContentType="image/jpeg",
            Body=content,
        )
    except gg_ml.GreengrassInferenceException as e:
        logging.info('Inference exception %s("%s")', e.__class__.__name__, e)
        return
    except gg_ml.GreengrassDependencyException as e:
        logging.info('Dependency exception %s("%s")', e.__class__.__name__, e)
        return

    logging.info("Response: %s", resp)
    predictions = resp["Body"].read()
    logging.info("Predictions: %s", predictions)

    # The connector output is in the format: [0.3,0.1,0.04,...]
    # Remove the '[' and ']' at the beginning and end.
    predictions = predictions[1:-1]
    predictions_arr = numpy.fromstring(predictions, sep=",")
```

```
logging.info("Split into %s predictions.", len(predictions_arr))

# Perform business logic that relies on predictions_arr, which is an array
# of probabilities.

# Schedule the infer() function to run again in one second.
Timer(1, infer).start()

infer()

# In this example, the required AWS Lambda handler is never called.
def function_handler(event, context):
    return
```

La `invoke_inference_service` funzione nel AWS IoT Greengrass Machine Learning SDK accetta i seguenti argomenti.

Argomento	Descrizione
AlgoType	Il nome del tipo di algoritmo da utilizzare per l'inferenza. Attualmente è supportato solo <code>image-classification</code> . Obbligatorio: true Tipo: string Valori validi: <code>image-classification</code>
ServiceName	Il nome del servizio di inferenza locale. Utilizza il nome specificato per il parametro <code>LocalInferenceServiceName</code> al momento della configurazione del connettore. Richiesto: true

Argomento	Descrizione
	Tipo: <code>string</code>
ContentType	Il tipo mime dell'immagine di input. Richiesto: <code>true</code> Tipo: <code>string</code> Valori validi: <code>image/jpeg</code> , <code>image/png</code>
Body	Il contenuto del file immagine di input. Richiesto: <code>true</code> Tipo: <code>binary</code>

Installazione MXNet delle dipendenze sul core AWS IoT Greengrass

Per utilizzare un connettore ML Image Classification, è necessario installare le dipendenze per il MXNet framework Apache sul dispositivo principale. I connettori utilizzano il framework per servire il modello ML.

Note

Questi connettori sono forniti in bundle con una MXNet libreria precompilata, quindi non è necessario installare il MXNet framework sul dispositivo principale.

AWS IoT Greengrass fornisce script per installare le dipendenze per le seguenti piattaforme e dispositivi comuni (o da utilizzare come riferimento per l'installazione). Se utilizzi una piattaforma o un dispositivo diverso, consulta la [MXNet documentazione](#) per la configurazione.

Prima di installare le MXNet dipendenze, assicurati che le [librerie di sistema](#) richieste (con le versioni minime specificate) siano presenti sul dispositivo.

NVIDIA Jetson TX2

1. Installa CUDA Toolkit 9.0 e cuDNN 7.0. Puoi seguire le istruzioni di [the section called “Configurazione di altri dispositivi”](#) nel tutorial Nozioni di base.
2. Abilita i repository universali in modo che il connettore sia in grado di installare l'open software gestito dalla community. Per ulteriori informazioni, consulta [Repository/Ubuntu](#) nella documentazione Ubuntu.
 - a. Apri il file `/etc/apt/sources.list`.
 - b. Assicurati che le seguenti righe non presentino commenti.

```
deb http://ports.ubuntu.com/ubuntu-ports/ xenial universe
deb-src http://ports.ubuntu.com/ubuntu-ports/ xenial universe
deb http://ports.ubuntu.com/ubuntu-ports/ xenial-updates universe
deb-src http://ports.ubuntu.com/ubuntu-ports/ xenial-updates universe
```

3. Salva una copia del seguente script di installazione nel file `nvidiajtx2.sh` del dispositivo core.

Python 3.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."
echo 'Assuming that universe repos are enabled and checking dependencies...'
apt-get -y update
apt-get -y dist-upgrade
apt-get install -y liblapack3 libopenblas-dev liblapack-dev libatlas-base-dev
apt-get install -y python3.7 python3.7-dev

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install
OpenCV with pip on this platform. Try building the latest OpenCV from source
(https://github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

 Note

Se [OpenCV](#) non viene installato correttamente utilizzando questo script, puoi provare a compilare dall'origine. Per ulteriori informazioni, consulta [Installazione in Linux](#) nella documentazione di OpenCV o fai riferimento ad altre risorse online per la tua piattaforma.

Python 2.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."
echo 'Assuming that universe repos are enabled and checking dependencies...'
apt-get -y update
apt-get -y dist-upgrade
apt-get install -y liblapack3 libopenblas-dev liblapack-dev libatlas-base-dev
python-dev

echo 'Install latest pip...'
wget https://bootstrap.pypa.io/get-pip.py
python get-pip.py
rm get-pip.py

pip install numpy==1.15.0 scipy

echo 'Dependency installation/upgrade complete.'
```

4. Dalla directory in cui hai salvato il file, esegui il seguente comando:

```
sudo nvidiajtx2.sh
```

x86_64 (Ubuntu or Amazon Linux)

1. Salva una copia del seguente script di installazione nel file `x86_64.sh` del dispositivo core.

Python 3.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

release=$(awk -F= '/^NAME/{print $2}' /etc/os-release)

if [ "$release" == '"Ubuntu"' ]; then
    # Ubuntu. Supports EC2 and DeepLens. DeepLens has all the dependencies
    # installed, so
    # this is mostly to prepare dependencies on Ubuntu EC2 instance.
    apt-get -y update
    apt-get -y dist-upgrade

    apt-get install -y libgfortran3 libsm6 libxext6 libxrender1
    apt-get install -y python3.7 python3.7-dev
elif [ "$release" == '"Amazon Linux"' ]; then
    # Amazon Linux. Expect python to be installed already
    yum -y update
    yum -y upgrade

    yum install -y compat-gcc-48-libgfortran libSM libXrender libXext
else
    echo "OS Release not supported: $release"
    exit 1
fi

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install
OpenCV with pip on this platform. Try building the latest OpenCV from source
(https://github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

Note

Se [OpenCV](#) non viene installato correttamente utilizzando questo script, puoi provare a compilare dall'origine. Per ulteriori informazioni, consulta [Installazione](#)

[in Linux](#) nella documentazione di OpenCV o fai riferimento ad altre risorse online per la tua piattaforma.

Python 2.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

release=$(awk -F= '/^NAME/{print $2}' /etc/os-release)

if [ "$release" == '"Ubuntu"' ]; then
    # Ubuntu. Supports EC2 and DeepLens. DeepLens has all the dependencies
    # installed, so
    # this is mostly to prepare dependencies on Ubuntu EC2 instance.
    apt-get -y update
    apt-get -y dist-upgrade

    apt-get install -y libgfortran3 libsm6 libxext6 libxrender1 python-dev
    python-pip
elif [ "$release" == '"Amazon Linux"' ]; then
    # Amazon Linux. Expect python to be installed already
    yum -y update
    yum -y upgrade

    yum install -y compat-gcc-48-libgfortran libSM libXrender libXext python-
    pip
else
    echo "OS Release not supported: $release"
    exit 1
fi

pip install numpy==1.15.0 scipy opencv-python

echo 'Dependency installation/upgrade complete.'
```

2. Dalla directory in cui hai salvato il file, esegui il comando seguente:

```
sudo x86_64.sh
```

Armv7 (Raspberry Pi)

1. Salva una copia del seguente script di installazione nel file `armv7l.sh` del dispositivo core.

Python 3.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

apt-get update
apt-get -y upgrade

apt-get install -y liblapack3 libopenblas-dev liblapack-dev
apt-get install -y python3.7 python3.7-dev

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install
OpenCV with pip on this platform. Try building the latest OpenCV from source
(https://github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

Note

Se [OpenCV](#) non viene installato correttamente utilizzando questo script, puoi provare a compilare dall'origine. Per ulteriori informazioni, consulta [Installazione in Linux](#) nella documentazione di OpenCV o fai riferimento ad altre risorse online per la tua piattaforma.

Python 2.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

apt-get update
```



```
apt-get -y upgrade

apt-get install -y liblapack3 libopenblas-dev liblapack-dev python-dev

# python-opencv depends on python-numpy. The latest version in the APT
# repository is python-numpy-1.8.2
# This script installs python-numpy first so that python-opencv can be
# installed, and then install the latest
# numpy-1.15.x with pip
apt-get install -y python-numpy python-opencv
dpkg --remove --force-depends python-numpy

echo 'Install latest pip...'
wget https://bootstrap.pypa.io/get-pip.py
python get-pip.py
rm get-pip.py

pip install --upgrade numpy==1.15.0 picamera scipy

echo 'Dependency installation/upgrade complete.'
```

2. Dalla directory in cui hai salvato il file, esegui il comando seguente:

```
sudo bash armv7l.sh
```

Note

Su un Raspberry Pi, l'utilizzo di pip per installare dipendenze di machine learning è un'operazione con elevati requisiti di memoria che può esaurire la memoria del dispositivo e causarne il blocco. Per risolvere il problema, è possibile aumentare temporaneamente la dimensione di swap:

In `/etc/dphys-swapfile`, aumenta il valore della variabile `CONF_SWAPSIZE` e quindi esegui il comando seguente per riavviare `dphys-swapfile`.

```
/etc/init.d/dphys-swapfile restart
```

Registrazione e risoluzione dei problemi

A seconda delle impostazioni del gruppo, i registri degli eventi e degli errori vengono scritti CloudWatch nei registri, nel file system locale o in entrambi. I log di questo connettore utilizzano il prefisso `LocalInferenceServiceName`. Se il connettore si comporta inaspettatamente, controlla i log del connettore. Questi di solito contengono utili informazioni di debug, ad esempio una dipendenza della libreria ML mancante o la causa di un errore di avvio del connettore.

Se il AWS IoT Greengrass gruppo è configurato per scrivere registri locali, il connettore scrive i file di registro su. `greengrass-root/ggc/var/log/user/region/aws/` Per ulteriori informazioni sulla registrazione di Greengrass, vedere. [the section called “Monitoraggio con i log AWS IoT Greengrass”](#)

Utilizzate le seguenti informazioni per risolvere i problemi relativi ai connettori ML Image Classification.

Librerie di sistema richieste

Le schede seguenti elencano le librerie di sistema richieste per ogni connettore ML Image Classification.

ML Image Classification Aarch64 JTX2

Libreria	Versione minima
ld-linux-aarch64.so.1	GLIBC_2.17
libc.so.6	GLIBC_2.17
libcublas.so.9.0	non applicabile
libcudart.so.9.0	non applicabile
libcudnn.so.7	non applicabile
libcufft.so.9.0	non applicabile
libcurand.so.9.0	non applicabile
libcusolver.so.9.0	non applicabile

Libreria	Versione minima
libgcc_s.so.1	GCC_4.2.0
libgomp.so.1	GOMP_4.0, OMP_1.0
libm.so.6	GLIBC_2.23
libpthread.so.0	GLIBC_2.17
librt.so.1	GLIBC_2.17
libstdc++.so.6	GLIBCXX_3.4.21, CXXABI_1.3.8

ML Image Classification x86_64

Libreria	Versione minima
ld-linux-x86-64.so.2	GCC_4.0.0
libc.so.6	GLIBC_2.4
libgfortran.so.3	GFORTTRAN_1.0
libm.so.6	GLIBC_2.23
libpthread.so.0	GLIBC_2.2.5
librt.so.1	GLIBC_2.2.5
libstdc++.so.6	CXXABI_1.3.8, GLIBCXX_3.4.21

ML Image Classification Armv7

Libreria	Versione minima
ld-linux-armhf.so.3	GLIBC_2.4
libc.so.6	GLIBC_2.7

Libreria	Versione minima
libgcc_s.so.1	GCC_4.0.0
libgfortran.so.3	GFORTTRAN_1.0
libm.so.6	GLIBC_2.4
libpthread.so.0	GLIBC_2.4
librt.so.1	GLIBC_2.4
libstdc++.so.6	CXXABI_1.3.8, CXXABI_ARM_1.3.3, GLIBCXX_3.4.20

Problemi

Caratteristiche	Soluzione
In un Raspberry Pi, il seguente messaggio di errore viene registrato e non si sta utilizzando la fotocamera: <code>Failed to initialize libdc1394</code>	Per disabilitare il driver, esegui il seguente comando: <pre>sudo ln /dev/null /dev/raw1394</pre> Questa operazione è effimera e il collegamento simbolico scompare dopo il riavvio. Consulta il manuale di distribuzione del sistema operativo per ulteriori informazioni su come creare automaticamente il link al riavvio.

Licenze

I connettori ML Image Classification includono i seguenti software/licenze di terze parti:

- [AWS SDK per Python \(Boto3\)](#)/Apache License 2.0
- [botocore](#)/Apache License 2.0
- [dateutil](#)/PSF License

- [docutils](#)/BSD License, GNU General Public License (GPL), Python Software Foundation License, Public Domain
- [jmespath](#)/MIT License
- [s3transfer](#)/Apache License 2.0
- [urllib3](#)/MIT License
- [Deep Neural Network Library \(DNNL\)](#)/Apache License 2.0
- [OpenMP* Runtime Library](#)/Consulta [Libreria di licenze di Intel OpenMP Runtime](#).
- [mxnet](#)/Apache License 2.0
- [six](#)/MIT

Libreria di licenze di Intel OpenMP Runtime. Il runtime Intel® OpenMP* è dotato di doppia licenza, con una licenza commerciale (COM) come parte dei prodotti Intel® Parallel Studio XE Suite e una licenza open source BSD (OSS).

Questo connettore è rilasciato ai sensi del contratto di [licenza del software Greengrass Core](#).

Changelog

La tabella seguente descrive le modifiche apportate a ciascuna versione del connettore.

Versione	Modifiche
2	È stato aggiunto il <code>MLFeedbackConnectorConfigId</code> parametro per supportare l'uso del connettore ML Feedback per caricare i dati di input del modello, pubblicare previsioni su un argomento MQTT e pubblicare metriche su Amazon. CloudWatch
1	Versione iniziale.

Un gruppo Greengrass può contenere una sola versione del connettore alla volta. Per informazioni sull'aggiornamento di una versione del connettore, consulta [the section called “Aggiornamento delle versioni dei connettori”](#).

Consulta anche

- [Integrazione con servizi e protocolli tramite i connettori](#)
- [the section called “Nozioni di base sui connettori \(console\)”](#)
- [the section called “Nozioni di base sui connettori \(CLI\)”](#)
- [Esecuzione dell'inferenza di Machine Learning](#)
- [Algoritmo di classificazione delle immagini](#) nella Amazon SageMaker AI Developer Guide

Connettore ML Object Detection

Warning

Questo connettore è entrato nella fase di vita prolungata e AWS IoT Greengrass non rilascerà aggiornamenti che forniscano funzionalità, miglioramenti alle funzionalità esistenti, patch di sicurezza o correzioni di bug. Per ulteriori informazioni, consulta [AWS IoT Greengrass Version 1 politica di manutenzione](#).

I [connettori](#) ML Object Detection forniscono un servizio di inferenza di machine learning (ML) che viene eseguito sul core. AWS IoT Greengrass Questo servizio di inferenza locale esegue il rilevamento degli oggetti utilizzando un modello di rilevamento degli oggetti compilato dal compilatore di deep learning SageMaker AI Neo. Sono supportati due tipi di modelli di rilevamento dell'oggetto: Single Shot Multibox Detector (SSD) e You Only Look Once (YOLO) v3. Per ulteriori informazioni, consulta la sezione relativa ai [requisiti del modello di rilevamento dell'oggetto](#).

Le funzioni Lambda definite dall'utente utilizzano il Machine AWS IoT Greengrass Learning SDK per inviare richieste di inferenza al servizio di inferenza locale. Il servizio esegue l'inferenza locale su un'immagine di input e restituisce un elenco di previsioni per ogni oggetto rilevato nell'immagine. Ogni previsione contiene una categoria dell'oggetto, un punteggio di affidabilità della previsione e le coordinate dei pixel che specificano un riquadro di delimitazione intorno all'oggetto previsto.

AWS IoT Greengrass fornisce connettori ML Object Detection per più piattaforme:

Connector	Descrizione e ARN
ML Object Detection Aarch64 JTX2	Servizio di inferenza per il rilevamento di oggetti per NVIDIA Jetson. TX2 Supporta l'accelerazione GPU. ARN: <code>arn:aws:greengrass: <i>region</i>::/connectors/ObjectDetectionAarch64JTX2/versions/1</code>
Rilevamento di oggetti ML x86_64	Servizio di inferenza di rilevamento oggetti per piattaforme x86_64. ARN: <code>arn:aws:greengrass: <i>region</i>::/connectors/ObjectDetectiononx86-64/versions/1</code>
Rilevamento di oggetti ML ARMv7	Servizio di inferenza per il rilevamento di oggetti per ARMv7 piattaforme. ARN: <code>arn:aws:greengrass: <i>region</i>::/connectors/ObjectDetectionARMv7/versions/1</code>

Requisiti

Questi connettori presentano i seguenti requisiti:

- AWS IoT Greengrass Core Software v1.9.3 o versione successiva.
- [Python](#) versione 3.7 o 3.8 installata sul dispositivo principale e aggiunta alla variabile di ambiente PATH.

Note

Per usare Python 3.8, esegui il seguente comando per creare un collegamento simbolico dalla cartella di installazione predefinita di Python 3.7 ai binari Python 3.8 installati.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Questo configura il dispositivo in modo che soddisfi il requisito Python per AWS IoT Greengrass.

- Dipendenze per il runtime di deep learning SageMaker AI Neo installato sul dispositivo principale. Per ulteriori informazioni, consulta [the section called “Installazione delle dipendenze Neo Deep Learning Runtime”](#).
- Una [risorsa ML](#) nel gruppo Greengrass. La risorsa ML deve fare riferimento a un bucket Amazon S3 che contiene un modello di rilevamento degli oggetti. Per ulteriori informazioni, consulta i sorgenti dei [modelli Amazon S3](#).

 Note

Il modello deve essere un tipo di modello di rilevamento dell'oggetto Single Shot Multibox Detector o You Only Look Once v3. Deve essere compilato utilizzando il compilatore di deep learning SageMaker AI Neo. Per ulteriori informazioni, consulta la sezione relativa ai [requisiti del modello di rilevamento dell'oggetto](#).

- Il [connettore ML Feedback](#) aggiunto al gruppo Greengrass e configurato. Questo è obbligatorio solo se desideri utilizzare il connettore per caricare i dati di input del modello e pubblicare le previsioni in un argomento MQTT.
- AWS IoT Greengrass Per interagire con questo connettore è necessario [Machine Learning SDK v1.1.0](#).

Requisiti del modello di rilevamento dell'oggetto

I connettori ML Object Detection supportano i modelli di rilevamento di oggetti Single Shot multibox Detector (SSD) e You Only Look Once (YOLO) v3. Puoi utilizzare i componenti di rilevamento dell'oggetto forniti da [GluonCV](#) per eseguire il training del modello con il tuo set di dati. In alternativa, puoi utilizzare modelli pre-formati da GluonCV Model Zoo:

- [Modello SSD pre-formato](#)
- [Modello YOLO v3 pre-formato](#)

Il training del modello di rilevamento dell'oggetto deve essere eseguito con immagini di input 512 x 512. I modelli pre-formati da GluonCV Model Zoo soddisfano già questo requisito.

I modelli di rilevamento di oggetti addestrati devono essere compilati con il compilatore di deep learning AI Neo. SageMaker Durante la compilazione, assicurati che l'hardware di destinazione corrisponda all'hardware del dispositivo core Greengrass. Per ulteriori informazioni, consulta [SageMaker AI Neo](#) nella Amazon SageMaker AI Developer Guide.

Il modello compilato deve essere aggiunto come risorsa ML ([origine del modello Amazon S3](#)) allo stesso gruppo Greengrass del connettore.

Parametri del connettore

Questi connettori forniscono i seguenti parametri.

MLModelDestinationPath

Il percorso assoluto verso il bucket Amazon S3 che contiene il modello ML compatibile con NEO. Si tratta del percorso di destinazione specificato per la risorsa del modello ML.

Nome visualizzato nella AWS IoT console: percorso di destinazione del modello

Obbligatorio: true

Tipo: string

Schema valido: .+

MLModelResourceId

L'ID della risorsa ML che fa riferimento al modello di origine.

Nome visualizzato nella AWS IoT console: risorsa ML del gruppo Greengrass

Obbligatorio: true

Tipo: S3MachineLearningModelResource

Schema valido: ^[a-zA-Z0-9: _-]+\$

LocalInferenceServiceName

Il nome del servizio di inferenza locale. Le funzioni Lambda definite dall'utente richiamano il servizio passando il nome alla funzione `invoke_inference_service` del Machine AWS IoT Greengrass Learning SDK. Per vedere un esempio, consulta [the section called “Esempio di utilizzo”](#).

Nome visualizzato nella AWS IoT console: nome del servizio di inferenza locale

Obbligatorio: true

Tipo: string

Schema valido: `^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$`

`LocalInferenceServiceTimeoutSeconds`

Il tempo (in secondi) prima che la richiesta di inferenza venga terminata. Il valore minimo è 1. Il valore predefinito è 10.

Nome visualizzato nella AWS IoT console: Timeout (secondo)

Obbligatorio: true

Tipo: string

Schema valido: `^[1-9][0-9]*$`

`LocalInferenceServiceMemoryLimitKB`

La quantità di memoria (in KB) a cui ha accesso il servizio. Il valore minimo è 1.

Nome visualizzato nella AWS IoT console: limite di memoria

Obbligatorio: true

Tipo: string

Schema valido: `^[1-9][0-9]*$`

`GPUAcceleration`

Il contesto di calcolo della CPU o GPU (accelerata). Questa proprietà si applica solo al JTX2 connettore ML Image Classification Aarch64.

Nome visualizzato nella AWS IoT console: accelerazione GPU

Obbligatorio: true

Tipo: string

Valori validi: CPU o GPU

MLFeedbackConnectorConfigId

L'ID della configurazione di feedback da utilizzare per caricare i dati di input del modello. Deve corrispondere all'ID di una configurazione di feedback definita per il [connettore ML Feedback](#).

Questo parametro è obbligatorio solo se desideri utilizzare il connettore ML Feedback per caricare i dati di input del modello e pubblicare le previsioni in un argomento MQTT.

Nome visualizzato nella AWS IoT console: ID di configurazione del connettore ML Feedback

Obbligatorio: false

Tipo: string

Schema valido: `^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$`

Esempio di creazione di un connettore (AWS CLI)

Il seguente comando CLI crea una `ConnectorDefinition` versione iniziale che contiene un connettore ML Object Detection. Questo esempio crea un'istanza del ARMv7l connettore ML Object Detection.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-  
version '{  
  "Connectors": [  
    {  
      "Id": "MyObjectDetectionConnector",  
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/  
ObjectDetectionARMv7/versions/1",  
      "Parameters": {  
        "MLModelDestinationPath": "/path-to-model",  
        "MLModelResourceId": "my-ml-resource",  
        "LocalInferenceServiceName": "objectDetection",  
        "LocalInferenceServiceTimeoutSeconds": "10",  
        "LocalInferenceServiceMemoryLimitKB": "500000",  
        "MLFeedbackConnectorConfigId" : "object-detector-random-sampling"  
      }  
    }  
  ]  
}'
```

Note

La funzione Lambda di questi connettori ha un ciclo di vita di [lunga durata](#).

Nella AWS IoT Greengrass console, puoi aggiungere un connettore dalla pagina Connettori del gruppo. Per ulteriori informazioni, consulta [the section called “Nozioni di base sui connettori \(console\)”](#).

Dati di input

Questi connettori accettano un file di immagine come input. I file di immagine di input devono essere in formato png o jpeg. Per ulteriori informazioni, consulta [the section called “Esempio di utilizzo”](#).

Questi connettori non accettano messaggi MQTT come dati di input.

Dati di output

Questi connettori restituiscono un elenco formattato di risultati delle previsioni per gli oggetti identificati nell'immagine di input:

```
{
  "prediction": [
    [
      14,
      0.9384938478469849,
      0.37763649225234985,
      0.5110225081443787,
      0.6697432398796082,
      0.8544386029243469
    ],
    [
      14,
      0.8859519958496094,
      0,
      0.43536216020584106,
      0.3314110040664673,
      0.9538808465003967
    ],
    [
      12,
      0.04128098487854004,
```

```
        0.5976729989051819,  
        0.5747185945510864,  
        0.704264223575592,  
        0.857937216758728  
    ],  
    ...  
]  
}
```

Ogni previsione nell'elenco è contenuta tra parentesi quadre e contiene sei valori:

- Il primo valore rappresenta la categoria di oggetto prevista per l'oggetto identificato. Le categorie di oggetti e i valori corrispondenti vengono determinati durante l'addestramento del modello di apprendimento automatico per il rilevamento di oggetti nel compilatore di deep learning Neo.
- Il secondo valore è il punteggio di affidabilità per la previsione della categoria dell'oggetto. Questo rappresenta la probabilità che la previsione fosse corretta.
- Gli ultimi quattro valori corrispondono alle dimensioni in pixel che rappresentano un riquadro di delimitazione intorno all'oggetto previsto nell'immagine.

Questi connettori non pubblicano messaggi MQTT come dati di output.

Esempio di utilizzo

L'esempio seguente della funzione Lambda utilizza il [AWS IoT Greengrass Machine Learning SDK](#) per interagire con un connettore ML Object Detection.

Note

Puoi scaricare l'SDK dalla pagina dei download di [AWS IoT Greengrass Machine Learning SDK](#).

In questo esempio viene inizializzato un client SDK e viene chiamata in modo sincrono la funzione `invoke_inference_service` di SDK per richiamare il servizio di inferenza locale. Trasferisce il tipo di algoritmo, il nome del servizio, il tipo di immagine e il contenuto dell'immagine. Quindi, l'esempio analizza la risposta del servizio per ottenere i risultati di probabilità (previsioni).

```
import logging  
from threading import Timer
```

```
import numpy as np

import greengrass_machine_learning_sdk as ml

# We assume the inference input image is provided as a local file
# to this inference client Lambda function.
with open('/test_img/test.jpg', 'rb') as f:
    content = bytearray(f.read())

client = ml.client('inference')

def infer():
    logging.info('invoking Greengrass ML Inference service')

    try:
        resp = client.invoke_inference_service(
            AlgoType='object-detection',
            ServiceName='objectDetection',
            ContentType='image/jpeg',
            Body=content
        )
    except ml.GreengrassInferenceException as e:
        logging.info('inference exception {}'.format(e.__class__.__name__, e))
        return
    except ml.GreengrassDependencyException as e:
        logging.info('dependency exception {}'.format(e.__class__.__name__, e))
        return

    logging.info('resp: {}'.format(resp))
    predictions = resp['Body'].read().decode("utf-8")
    logging.info('predictions: {}'.format(predictions))
    predictions = eval(predictions)

    # Perform business logic that relies on the predictions.

    # Schedule the infer() function to run again in ten second.
    Timer(10, infer).start()
    return

infer()

def function_handler(event, context):
```

```
return
```

La `invoke_inference_service` funzione nel AWS IoT Greengrass Machine Learning SDK accetta i seguenti argomenti.

Argomento	Descrizione
<code>AlgoType</code>	Il nome del tipo di algoritmo da utilizzare per l'inferenza. Attualmente è supportato solo <code>object-detection</code> . Obbligatorio: <code>true</code> Tipo: <code>string</code> Valori validi: <code>object-detection</code>
<code>ServiceName</code>	Il nome del servizio di inferenza locale. Utilizza il nome specificato per il parametro <code>LocalInferenceServiceName</code> al momento della configurazione del connettore. Richiesto: <code>true</code> Tipo: <code>string</code>
<code>ContentType</code>	Il tipo mime dell'immagine di input. Richiesto: <code>true</code> Tipo: <code>string</code> Valori validi: <code>image/jpeg</code>, <code>image/png</code>
<code>Body</code>	Il contenuto del file immagine di input. Richiesto: <code>true</code> Tipo: <code>binary</code>

Installazione delle dipendenze del runtime di Neo Deep Learning sul core AWS IoT Greengrass

I connettori ML Object Detection sono forniti in bundle con il deep learning runtime (DLR) SageMaker AI Neo. I connettori utilizzano il runtime per servire il modello ML. Per utilizzare questi connettori, devi installare le dipendenze per il DLR sul dispositivo core.

Prima di installare le dipendenze DLR, assicurati che le [librerie di sistema](#) richieste (nelle versioni minime specificate) siano presenti sul dispositivo.

NVIDIA Jetson TX2

1. Installa CUDA Toolkit 9.0 e cuDNN 7.0. Puoi seguire le istruzioni di [the section called “Configurazione di altri dispositivi”](#) nel tutorial Nozioni di base.
2. Abilita i repository universali in modo che il connettore sia in grado di installare l'open software gestito dalla community. Per ulteriori informazioni, consulta [Repository/Ubuntu](#) nella documentazione Ubuntu.
 - a. Apri il file `/etc/apt/sources.list`.
 - b. Assicurati che le seguenti righe non presentino commenti.

```
deb http://ports.ubuntu.com/ubuntu-ports/ xenial universe
deb-src http://ports.ubuntu.com/ubuntu-ports/ xenial universe
deb http://ports.ubuntu.com/ubuntu-ports/ xenial-updates universe
deb-src http://ports.ubuntu.com/ubuntu-ports/ xenial-updates universe
```

3. Salva una copia del seguente script di installazione nel file `nvidiajtx2.sh` del dispositivo core.

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."
echo 'Assuming that universe repos are enabled and checking dependencies...'
apt-get -y update
apt-get -y dist-upgrade
apt-get install -y liblapack3 libopenblas-dev liblapack-dev libatlas-base-dev
apt-get install -y python3.7 python3.7-dev

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
```



```
python3.7 -m pip install opencv-python || echo 'Error: Unable to install OpenCV
with pip on this platform. Try building the latest OpenCV from source (https://
github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

 Note

Se [OpenCV](#) non viene installato correttamente utilizzando questo script, puoi provare a compilare dall'origine. Per ulteriori informazioni, consulta [Installazione in Linux](#) nella documentazione di OpenCV o fai riferimento ad altre risorse online per la tua piattaforma.

4. Dalla directory in cui hai salvato il file, esegui il seguente comando:

```
sudo nvidiajtx2.sh
```

x86_64 (Ubuntu or Amazon Linux)

1. Salva una copia del seguente script di installazione nel file `x86_64.sh` del dispositivo core.

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

release=$(awk -F= '/^NAME/{print $2}' /etc/os-release)

if [ "$release" == '"Ubuntu"' ]; then
    # Ubuntu. Supports EC2 and DeepLens. DeepLens has all the dependencies
    installed, so
    # this is mostly to prepare dependencies on Ubuntu EC2 instance.
    apt-get -y update
    apt-get -y dist-upgrade

    apt-get install -y libgfortran3 libsm6 libxext6 libxrender1
    apt-get install -y python3.7 python3.7-dev
elif [ "$release" == '"Amazon Linux"' ]; then
    # Amazon Linux. Expect python to be installed already
    yum -y update
```

```
yum -y upgrade

yum install -y compat-gcc-48-libgfortran libSM libXrender libXext
else
    echo "OS Release not supported: $release"
    exit 1
fi

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install OpenCV
with pip on this platform. Try building the latest OpenCV from source (https://
github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

Note

Se [OpenCV](#) non viene installato correttamente utilizzando questo script, puoi provare a compilare dall'origine. Per ulteriori informazioni, consulta [Installazione in Linux](#) nella documentazione di OpenCV o fai riferimento ad altre risorse online per la tua piattaforma.

2. Dalla directory in cui hai salvato il file, esegui il comando seguente:

```
sudo x86_64.sh
```

ARMv7 (Raspberry Pi)

1. Salva una copia del seguente script di installazione nel file `armv7l.sh` del dispositivo core.

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

apt-get update
apt-get -y upgrade

apt-get install -y liblapack3 libopenblas-dev liblapack-dev
```

```
apt-get install -y python3.7 python3.7-dev

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install OpenCV
with pip on this platform. Try building the latest OpenCV from source (https://
github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

Note

Se [OpenCV](#) non viene installato correttamente utilizzando questo script, puoi provare a compilare dall'origine. Per ulteriori informazioni, consulta [Installazione in Linux](#) nella documentazione di OpenCV o fai riferimento ad altre risorse online per la tua piattaforma.

2. Dalla directory in cui hai salvato il file, esegui il comando seguente:

```
sudo bash armv7l.sh
```

Note

Su un Raspberry Pi, l'utilizzo di pip per installare dipendenze di machine learning è un'operazione con elevati requisiti di memoria che può esaurire la memoria del dispositivo e causarne il blocco. Per risolvere il problema, è possibile aumentare temporaneamente la dimensione di swap: In `/etc/dphys-swapfile`, aumenta il valore della variabile `CONF_SWAPSIZE` e quindi esegui il comando seguente per riavviare `dphys-swapfile`.

```
/etc/init.d/dphys-swapfile restart
```

Registrazione e risoluzione dei problemi

A seconda delle impostazioni del gruppo, i registri degli eventi e degli errori vengono scritti CloudWatch nei registri, nel file system locale o in entrambi. I log di questo connettore utilizzano il prefisso `LocalInferenceServiceName`. Se il connettore si comporta inaspettatamente,

controlla i log del connettore. Questi di solito contengono utili informazioni di debug, ad esempio una dipendenza della libreria ML mancante o la causa di un errore di avvio del connettore.

Se il AWS IoT Greengrass gruppo è configurato per scrivere registri locali, il connettore scrive i file di registro su. `greengrass-root/ggc/var/log/user/region/aws/` Per ulteriori informazioni sulla registrazione di Greengrass, vedere. [the section called “Monitoraggio con i log AWS IoT Greengrass”](#)

Utilizzate le seguenti informazioni per risolvere i problemi relativi ai connettori ML Object Detection.

Librerie di sistema richieste

Le seguenti schede elencano le librerie di sistema necessarie per ogni connettore ML Object Detection.

ML Object Detection Aarch64 JTX2

Libreria	Versione minima
ld-linux-aarch64.so.1	GLIBC_2.17
libc.so.6	GLIBC_2.17
libcublas.so.9.0	non applicabile
libcudart.so.9.0	non applicabile
libcudnn.so.7	non applicabile
libcufft.so.9.0	non applicabile
libcurand.so.9.0	non applicabile
libcusolver.so.9.0	non applicabile
libgcc_s.so.1	GCC_4.2.0
libgomp.so.1	GOMP_4.0, OMP_1.0
libm.so.6	GLIBC_2.23
libnvinfer.so.4	non applicabile

Libreria	Versione minima
libnvm_gpu.so	non applicabile
libnvm.so	non applicabile
libnvidia-fatbinaryloader.so.28.2.1	non applicabile
libnvos.so	non applicabile
libpthread.so.0	GLIBC_2.17
librt.so.1	GLIBC_2.17
libstdc++.so.6	GLIBCXX_3.4.21, CXXABI_1.3.8

ML Object Detection x86_64

Libreria	Versione minima
ld-linux-x86-64.so.2	GCC_4.0.0
libc.so.6	GLIBC_2.4
libgfortran.so.3	GFORTTRAN_1.0
libm.so.6	GLIBC_2.23
libpthread.so.0	GLIBC_2.2.5
librt.so.1	GLIBC_2.2.5
libstdc++.so.6	CXXABI_1.3.8, GLIBCXX_3.4.21

ML Object Detection ARMv7

Libreria	Versione minima
ld-linux-armhf.so.3	GLIBC_2.4

Libreria	Versione minima
libc.so.6	GLIBC_2.7
libgcc_s.so.1	GCC_4.0.0
libgfortran.so.3	GFORTRAN_1.0
libm.so.6	GLIBC_2.4
libpthread.so.0	GLIBC_2.4
librt.so.1	GLIBC_2.4
libstdc++.so.6	CXXABI_1.3.8, CXXABI_ARM_1.3.3, GLIBCXX_3.4.20

Problemi

Caratteristiche	Soluzione
In un Raspberry Pi, il seguente messaggio di errore viene registrato e non si sta utilizzando la fotocamera: <code>Failed to initialize libdc1394</code>	Per disabilitare il driver, esegui il seguente comando: <pre>sudo ln /dev/null /dev/raw1394</pre> Questa operazione è temporanea. Il collegamento simbolico scompare dopo il riavvio. Consulta il manuale di distribuzione del sistema operativo per ulteriori informazioni su come creare automaticamente il link al riavvio.

Licenze

I connettori ML Object Detection includono i seguenti software/licenze di terze parti:

- [AWS SDK per Python \(Boto3\)](#)/Apache License 2.0
- [botocore](#)/Apache License 2.0

- [dateutil](#)/PSF License
- [docutils](#)/BSD License, GNU General Public License (GPL), Python Software Foundation License, Public Domain
- [jmespath](#)/MIT License
- [s3transfer](#)/Apache License 2.0
- [urllib3](#)/MIT License

- [Deep Learning Runtime](#)/Apache License 2.0
- [six](#)/MIT

Questo connettore è rilasciato ai sensi del contratto di [licenza del software Greengrass Core](#).

Consulta anche

- [Integrazione con servizi e protocolli tramite i connettori](#)
- [the section called “Nozioni di base sui connettori \(console\)”](#)
- [the section called “Nozioni di base sui connettori \(CLI\)”](#)
- [Esecuzione dell'inferenza di Machine Learning](#)
- [Algoritmo di rilevamento degli oggetti](#) nell'Amazon SageMaker AI Developer Guide

Connettore adattatore di protocollo Modbus-RTU

Il [connettore](#) Modbus-RTU Protocol Adapter raccoglie le informazioni dai dispositivi Modbus RTU che fanno parte del gruppo. AWS IoT Greengrass

Questo connettore riceve i parametri per una richiesta Modbus RTU da una funzione Lambda definita dall'utente. Invia la richiesta corrispondente e quindi pubblica la risposta dal dispositivo di destinazione come messaggio MQTT.

Questo connettore ha le seguenti versioni.

Versione	ARN
3	arn:aws:greengrass: <i>region</i> ::/ connectors/ModbusRTUProtocolAdapter/versions/3
2	arn:aws:greengrass: <i>region</i> ::/ connectors/ModbusRTUProtocolAdapter/versions/2
1	arn:aws:greengrass: <i>region</i> ::/ connectors/ModbusRTUProtocolAdapter/versions/1

Per informazioni sulle modifiche di ogni versione, consulta [Changelog](#).

Requisiti

Questo connettore presenta i seguenti requisiti:

Version 3

- AWS IoT Greengrass Software principale v1.9.3 o versione successiva.
- [Python](#) versione 3.7 o 3.8 installata sul dispositivo principale e aggiunta alla variabile di ambiente PATH.

Note

Per usare Python 3.8, esegui il seguente comando per creare un collegamento simbolico dalla cartella di installazione predefinita di Python 3.7 ai binari Python 3.8 installati.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Questo configura il dispositivo in modo che soddisfi il requisito Python per AWS IoT Greengrass.

- Una connessione fisica tra il core e i dispositivi Modbus. AWS IoT Greengrass Il core deve essere fisicamente connesso alla rete RTU Modbus attraverso una porta seriale (ad esempio, una porta USB).
- Una [risorsa di dispositivo locale](#) del gruppo Greengrass che punta alla porta seriale Modbus fisica.
- Una funzione Lambda definita dall'utente che invia i parametri di richiesta Modbus RTU a questo connettore. I parametri di richiesta devono essere conformi ai modelli previsti e includere gli indirizzi IDs e gli indirizzi dei dispositivi di destinazione sulla rete Modbus RTU. Per ulteriori informazioni, consulta [the section called "Dati di input"](#).

Versions 1 - 2

- AWS IoT Greengrass Software principale v1.7 o successivo.
- [Python](#) versione 2.7 installato sul dispositivo principale e aggiunto alla variabile di ambiente PATH.
- Una connessione fisica tra il AWS IoT Greengrass core e i dispositivi Modbus. Il core deve essere fisicamente connesso alla rete RTU Modbus attraverso una porta seriale (ad esempio, una porta USB).
- Una [risorsa di dispositivo locale](#) del gruppo Greengrass che punta alla porta seriale Modbus fisica.
- Una funzione Lambda definita dall'utente che invia i parametri di richiesta Modbus RTU a questo connettore. I parametri di richiesta devono essere conformi ai modelli previsti e includere gli indirizzi IDs e gli indirizzi dei dispositivi di destinazione sulla rete Modbus RTU. Per ulteriori informazioni, consulta [the section called "Dati di input"](#).

Parametri del connettore

Questo connettore supporta i seguenti parametri:

ModbusSerialPort-ResourceId

L'ID della risorsa del dispositivo locale fisico che rappresenta la porta seriale Modbus fisica.

Note

Al connettore è concesso l'accesso in lettura e scrittura alla risorsa.

Nome visualizzato nella AWS IoT console: risorsa della porta seriale Modbus

Richiesto: true

Tipo: string

Schema valido: .+

ModbusSerialPort

Il percorso assoluto della porta seriale Modbus fisica sul dispositivo. Si tratta del percorso di origine specificato per la risorsa del dispositivo locale Modbus.

Nome visualizzato nella AWS IoT console: percorso di origine della risorsa della porta seriale Modbus

Obbligatorio: true

Tipo: string

Schema valido: .+

Esempio di creazione di un connettore (AWS CLI)

Il seguente comando CLI crea un `ConnectorDefinition` con una versione iniziale che contiene il connettore Modbus-RTU Protocol Adapter.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-  
version '{  
  "Connectors": [  
    {  
      "Id": "MyModbusRTUProtocolAdapterConnector",  
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/  
ModbusRTUProtocolAdapter/versions/3",  
      "Parameters": {  
        "ModbusSerialPort-ResourceId": "MyLocalModbusSerialPort",  
        "ModbusSerialPort": "/path-to-port"  
      }  
    }  
  ]  
}'
```

 Note

La funzione Lambda in questo connettore ha un ciclo di vita di [lunga durata](#).

Nella AWS IoT Greengrass console, puoi aggiungere un connettore dalla pagina Connettori del gruppo. Per ulteriori informazioni, consulta [the section called “Nozioni di base sui connettori \(console\)”](#).

 Note

Dopo aver distribuito il connettore Modbus-RTU Protocol Adapter, puoi utilizzarlo AWS IoT Things Graph per orchestrare le interazioni tra i dispositivi del gruppo. Per ulteriori informazioni, consulta [Modbus](#) nella AWS IoT Things Graph Guida per l'utente.

Dati di input

Questo connettore accetta i parametri di richiesta Modbus RTU da una funzione Lambda definita dall'utente su un argomento MQTT. I messaggi di input devono essere in formato JSON.

Filtro argomento in sottoscrizione

`modbus/adapter/request`

Proprietà dei messaggi

Il messaggio di richiesta varia in base al tipo di richiesta RTU Modbus che rappresenta. Le seguenti proprietà sono necessarie per tutte le richieste:

- Nell'oggetto `request`:
 - `operation`. Il nome dell'operazione da eseguire. Ad esempio, specificare `"operation": "ReadCoilsRequest"` per leggere i nastri. Questo valore deve essere una stringa Unicode. Per le operazioni supportate, consulta [the section called “Richieste e risposte RTU Modbus”](#).
 - `device`. Il dispositivo di destinazione della richiesta. Questo valore deve essere compreso tra 0 - 247.
- La proprietà `id`. L'ID della richiesta. Tale valore viene utilizzato per la deduplicazione dei dati e viene restituito come è nella proprietà `id` di tutte le risposte, incluse quelle di errore. Questo valore deve essere una stringa Unicode.

Note

Se la richiesta include un campo indirizzo, è necessario specificare il valore come numero intero. Ad esempio "address": 1.

Gli altri parametri di includere nella richiesta variano a seconda dell'operazione. Tutti i parametri della richiesta sono necessari tranne il CRC, che viene gestito separatamente. Per alcuni esempi, consulta [the section called “Richieste e risposte di esempio”](#).

Esempio di input: richiesta lettura nastri

```
{
  "request": {
    "operation": "ReadCoilsRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "TestRequest"
}
```

Dati di output

Questo connettore pubblica le risposte nelle richieste RTU Modbus in entrata.

Filtro argomento in sottoscrizione

modbus/adapter/response

Proprietà dei messaggi

Il formato del messaggio di risposta varia in base alla richiesta corrispondente e allo stato della risposta. Per alcuni esempi, consulta [the section called “Richieste e risposte di esempio”](#).

Note

Una risposta a un'operazione di scrittura è semplicemente un eco della richiesta. Benché non vengano restituite informazioni rilevanti per le risposte in scrittura, è buona prassi controllare lo stato della risposta.

Ogni risposta include le seguenti proprietà:

- Nell'oggetto `response`:
 - `status`. Lo stato della richiesta. Lo stato può avere uno dei seguenti valori:
 - `Success`. La richiesta era valida, è stata inviata alla rete Modbus RTU ed è stata restituita una risposta.
 - `Exception`. La richiesta era valida, è stata inviata alla rete Modbus RTU ed è stata restituita una risposta di eccezione. Per ulteriori informazioni, consulta [the section called "Stato risposta: eccezione"](#).
 - `No Response`. La richiesta non era valida e il connettore ha rilevato l'errore prima che la richiesta fosse inviata sulla rete Modbus RTU. Per ulteriori informazioni, consulta [the section called "Stato risposta: nessuna risposta"](#).
 - `device`. Il dispositivo a cui è stata inviata la richiesta.
 - `operation`. Il tipo di richiesta che è stata inviata.
 - `payload`. Il contenuto della risposta che è stato restituito. Se lo `status` è `No Response`, questo oggetto contiene solo una proprietà `error` con la descrizione dell'errore (ad esempio, `"error": "[Input/Output] No Response received from the remote unit"`).
- La proprietà `id`. L'ID della richiesta, utilizzato per la deduplicazione dei dati.

Output di esempio: Operazione riuscita

```
{
  "response" : {
    "status" : "success",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "function_code": 1,
      "bits": [1]
    }
  },
  "id" : "TestRequest"
}
```

Esempio di output: Errore

```
{
  "response" : {
    "status" : "fail",
```

```
    "error_message": "Internal Error",
    "error": "Exception",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
        "function_code": 129,
        "exception_code": 2
    }
  },
  "id" : "TestRequest"
}
```

Per ulteriori esempi, consulta [the section called “Richieste e risposte di esempio”](#).

Richieste e risposte RTU Modbus

Questo connettore accetta i parametri della richiesta RTU Modbus come [dati di input](#) e pubblica le risposte come [dati di output](#).

Sono supportate le seguenti operazioni comuni.

Nome dell'operazione nella richiesta	Codice della funzione in risposta
ReadCoilsRequest	01
ReadDiscreteInputsRequest	02
ReadHoldingRegistersRequest	03
ReadInputRegistersRequest	04
WriteSingleCoilRequest	05
WriteSingleRegisterRequest	06
WriteMultipleCoilsRequest	15
WriteMultipleRegistersRequest	16
MaskWriteRegisterRequest	22

Nome dell'operazione nella richiesta	Codice della funzione in risposta
ReadWriteMultipleRegistersRequest	23

Richieste e risposte di esempio

Di seguito sono riportate alcune richieste e risposte di esempio per le operazioni supportate.

Lettura nastri

Esempio di richiesta:

```
{
  "request": {
    "operation": "ReadCoilsRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "TestRequest"
}
```

Esempio di risposta:

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "function_code": 1,
      "bits": [1]
    }
  },
  "id" : "TestRequest"
}
```

Lettura di input discreti

Esempio di richiesta:

```
{
```

```
{
  "request": {
    "operation": "ReadDiscreteInputsRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "TestRequest"
}
```

Esempio di risposta:

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "ReadDiscreteInputsRequest",
    "payload": {
      "function_code": 2,
      "bits": [1]
    }
  },
  "id" : "TestRequest"
}
```

Lettura registri di sospensione

Esempio di richiesta:

```
{
  "request": {
    "operation": "ReadHoldingRegistersRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "TestRequest"
}
```

Esempio di risposta:

```
{
  "response": {
```



```
    "status": "success",
    "device": 1,
    "operation": "ReadHoldingRegistersRequest",
    "payload": {
      "function_code": 3,
      "registers": [20,30]
    },
    "id" : "TestRequest"
  }
```

Lettura registri di input

Esempio di richiesta:

```
{
  "request": {
    "operation": "ReadInputRegistersRequest",
    "device": 1,
    "address": 1,
    "value": 1
  },
  "id": "TestRequest"
}
```

Scrittura nastro singolo

Esempio di richiesta:

```
{
  "request": {
    "operation": "WriteSingleCoilRequest",
    "device": 1,
    "address": 1,
    "value": 1
  },
  "id": "TestRequest"
}
```

Esempio di risposta:

```
{
  "response": {
```

```
    "status": "success",
    "device": 1,
    "operation": "WriteSingleCoilRequest",
    "payload": {
      "function_code": 5,
      "address": 1,
      "value": true
    }
  },
  "id" : "TestRequest"
```

Scrittura registro singolo

Esempio di richiesta:

```
{
  "request": {
    "operation": "WriteSingleRegisterRequest",
    "device": 1,
    "address": 1,
    "value": 1
  },
  "id": "TestRequest"
}
```

Scrittura di più nastri

Esempio di richiesta:

```
{
  "request": {
    "operation": "WriteMultipleCoilsRequest",
    "device": 1,
    "address": 1,
    "values": [1,0,0,1]
  },
  "id": "TestRequest"
}
```

Esempio di risposta:

```
{
```

```
"response": {
  "status": "success",
  "device": 1,
  "operation": "WriteMultipleCoilsRequest",
  "payload": {
    "function_code": 15,
    "address": 1,
    "count": 4
  }
},
"id" : "TestRequest"
}
```

Scrittura di più registri

Esempio di richiesta:

```
{
  "request": {
    "operation": "WriteMultipleRegistersRequest",
    "device": 1,
    "address": 1,
    "values": [20,30,10]
  },
  "id": "TestRequest"
}
```

Esempio di risposta:

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "WriteMultipleRegistersRequest",
    "payload": {
      "function_code": 23,
      "address": 1,
      "count": 3
    }
  },
  "id" : "TestRequest"
}
```

Mascheramento dei registri di scrittura

Esempio di richiesta:

```
{
  "request": {
    "operation": "MaskWriteRegisterRequest",
    "device": 1,
    "address": 1,
    "and_mask": 175,
    "or_mask": 1
  },
  "id": "TestRequest"
}
```

Esempio di risposta:

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "MaskWriteRegisterRequest",
    "payload": {
      "function_code": 22,
      "and_mask": 0,
      "or_mask": 8
    }
  },
  "id" : "TestRequest"
}
```

Scrittura/lettura di più registri

Esempio di richiesta:

```
{
  "request": {
    "operation": "ReadWriteMultipleRegistersRequest",
    "device": 1,
    "read_address": 1,
    "read_count": 2,
    "write_address": 3,
    "write_registers": [20,30,40]
  }
}
```

```
  },  
  "id": "TestRequest"  
}
```

Esempio di risposta:

```
{  
  "response": {  
    "status": "success",  
    "device": 1,  
    "operation": "ReadWriteMultipleRegistersRequest",  
    "payload": {  
      "function_code": 23,  
      "registers": [10,20,10,20]  
    }  
  },  
  "id" : "TestRequest"  
}
```

Note

I registri restituiti in questa risposta sono quelli che vengono letti.

Stato risposta: eccezione

Le eccezioni possono verificarsi se il formato della richiesta è valido, ma la richiesta non è stata completata. In questo caso, la risposta contiene le seguenti informazioni:

- Il `status` è impostato su `Exception`.
- `function_code` è pari al codice della funzione della richiesta + 128.
- `exception_code` contiene il codice dell'eccezione. Per ulteriori informazioni sull'eccezione , consulta [Codici delle eccezioni Modbus](#).

Esempio:

```
{  
  "response" : {  
    "status" : "fail",  
    "error_message": "Internal Error",
```

```

    "error": "Exception",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
        "function_code": 129,
        "exception_code": 2
    }
  },
  "id" : "TestRequest"
}

```

Stato risposta: nessuna risposta

Questo connettore esegue controlli di convalida sulla richiesta Modbus. Ad esempio, verifica l'eventuale presenza di formati non validi e campi non compilati. Se la convalida ha esito negativo, il connettore non invia la richiesta. Al contrario, restituirà una risposta contenente le seguenti informazioni:

- Il `status` è impostato su `No Response`.
- `error` Contiene il motivo dell'errore.
- `error_message` contiene il messaggio dell'errore.

Esempi:

```

{
  "response" : {
    "status" : "fail",
    "error_message": "Invalid address field. Expected <type 'int'>, got <type 'str'>",
    "error": "No Response",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "error": "Invalid address field. Expected <type 'int'>, got <type 'str'>"
    }
  },
  "id" : "TestRequest"
}

```

Se la richiesta è destinata a un dispositivo inesistente o se la rete RTU Modbus non funziona, potrebbe venire restituito `ModbusIOException`, che utilizza il formato `Nessuna risposta`.

```
{
  "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"
}
```

Esempio di utilizzo

Usa i seguenti passaggi di alto livello per configurare una funzione Lambda di esempio di Python 3.7 che puoi usare per provare il connettore.

Note

- Se usi altri runtime Python, puoi creare un collegamento simbolico da Python3.x a Python 3.7.
- Gli argomenti [Nozioni di base sui connettori \(console\)](#) e [Nozioni di base sui connettori \(CLI\)](#) contengono passaggi dettagliati che illustrano come configurare e distribuire un connettore Twilio Notifications di esempio.

1. Assicurarsi di soddisfare i [requisiti](#) per il connettore.
2. Crea e pubblica una funzione Lambda che invia dati di input al connettore.

Salvare il [codice di esempio](#) come file PY. Scarica e decomprimi il [AWS IoT Greengrass Core SDK per Python](#). Quindi, crea un pacchetto zip che contiene il file PY e la cartella greengrasssdk a livello root. Questo pacchetto zip è il pacchetto di distribuzione in cui carichi AWS Lambda

Dopo aver creato la funzione Python 3.7 Lambda, pubblica una versione della funzione e crea un alias.

3. Configurare il gruppo Greengrass.

- a. Aggiungi la funzione Lambda tramite il relativo alias (consigliato). Configura il ciclo di vita Lambda come longevo (o nella "Pinned": `true` CLI).
 - b. Aggiungi la risorsa del dispositivo locale richiesta e concedi l'accesso in lettura/scrittura alla funzione Lambda.
 - c. Aggiungere il connettore e configurarne i relativi [parametri](#).
 - d. Aggiungere sottoscrizioni che consentono al connettore di ricevere [i dati di input](#) e inviare [i dati di output](#) nei filtri degli argomenti supportati.
 - Imposta la funzione Lambda come origine, il connettore come destinazione e utilizza un filtro per argomenti di input supportato.
 - Imposta il connettore come origine, AWS IoT Core come destinazione e utilizza un filtro per l'argomento di output supportato. Utilizzi questo abbonamento per visualizzare i messaggi di stato nella AWS IoT console.
4. Distribuisci il gruppo.
 5. Nella AWS IoT console, nella pagina Test, sottoscrivi l'argomento relativo ai dati di output per visualizzare i messaggi di stato dal connettore. La funzione Lambda di esempio è di lunga durata e inizia a inviare messaggi subito dopo l'implementazione del gruppo.

Al termine del test, puoi impostare il ciclo di vita Lambda su richiesta (o nella CLI) e "Pinned": `false` distribuire il gruppo. Ciò impedisce alla funzione di inviare messaggi.

Esempio

L'esempio seguente della funzione Lambda invia un messaggio di input al connettore.

```
import greengrasssdk
import json

TOPIC_REQUEST = 'modbus/adapter/request'

# Creating a greengrass core sdk client
iot_client = greengrasssdk.client('iot-data')

def create_read_coils_request():
    request = {
        "request": {
            "operation": "ReadCoilsRequest",
            "device": 1,
```



```
"address": 1,
"count": 1
},
"id": "TestRequest"
}
return request

def publish_basic_request():
    iot_client.publish(payload=json.dumps(create_read_coils_request()),
    topic=TOPIC_REQUEST)

publish_basic_request()

def lambda_handler(event, context):
    return
```

Licenze

Il connettore Modbus-RTU Protocol Adapter include i seguenti software/licenze di terze parti:

- [pymodbus](#)/BSD
- [pyserial](#)/BSD

Questo connettore è rilasciato ai sensi del contratto di [licenza del software Greengrass Core](#).

Changelog

La tabella seguente descrive le modifiche apportate a ciascuna versione del connettore.

Versione	Modifiche
3	È stato aggiornato il runtime Lambda a Python 3.7, che modifica i requisiti di runtime.
2	Arn del connettore aggiornato per Regione AWS il supporto. Registrazione degli errori migliorata.
1	Versione iniziale.

Un gruppo Greengrass può contenere una sola versione del connettore alla volta. Per informazioni sull'aggiornamento di una versione del connettore, consulta [the section called “Aggiornamento delle versioni dei connettori”](#).

Consulta anche

- [Integrazione con servizi e protocolli tramite i connettori](#)
- [the section called “Nozioni di base sui connettori \(console\)”](#)
- [the section called “Nozioni di base sui connettori \(CLI\)”](#)

Connettore adattatore di protocollo Modbus-TCP

Il [connettore](#) Modbus-TCP Protocol Adapter raccoglie i dati dai dispositivi locali tramite il protocollo Modbus-TCP e li pubblica negli stream selezionati. `StreamManager`

Puoi utilizzare questo connettore anche con il SiteWise connettore IoT e il tuo SiteWise gateway IoT. Il gateway deve fornire la configurazione per il connettore. Per ulteriori informazioni, consulta [Configurare una sorgente Modbus TCP](#) nella guida SiteWise utente IoT.

Note

Questo connettore funziona in modalità [Nessun contenitore](#) isolato, quindi puoi distribuirlo a un AWS IoT Greengrass gruppo in esecuzione in un contenitore Docker.

Questo connettore ha le seguenti versioni.

Versione	ARN
3	<code>arn:aws:greengrass: <i>region</i>::/connectors/ModbusTCPConnector/versions/3</code>
2	<code>arn:aws:greengrass: <i>region</i>::/connectors/ModbusTCPConnector/versions/2</code>

Versione	ARN
1	<code>arn:aws:greengrass: <i>region</i>::/connectors/ModbusTCPConnector/versions/1</code>

Per informazioni sulle modifiche di ogni versione, consulta [Changelog](#).

Requisiti

Questo connettore presenta i seguenti requisiti:

Version 1 - 3

- AWS IoT Greengrass Software principale v1.10.2 o successivo.
- Stream manager abilitato sul gruppo. AWS IoT Greengrass
- Java 8 installato sul dispositivo principale e aggiunto alla variabile di PATH ambiente.

Note

Questo connettore è disponibile solo nelle seguenti regioni:

- ap-southeast-1
- ap-southeast-2
- eu-central-1
- eu-west-1
- us-east-1
- us-west-2
- cn-north-1

Parametri del connettore

Questo connettore supporta i seguenti parametri:

LocalStoragePath

La directory sull' AWS IoT Greengrass host su cui il SiteWise connettore IoT può scrivere dati persistenti. La directory predefinita è `/var/sitewise`.

Nome visualizzato nella AWS IoT console: Percorso di archiviazione locale

Obbligatorio: `false`

Tipo: `string`

Schema valido: `^\s*$|\/.`

MaximumBufferSize

La dimensione massima in GB per l'utilizzo SiteWise del disco IoT. La dimensione predefinita è 10 GB.

Nome visualizzato nella AWS IoT console: dimensione massima del buffer del disco

Obbligatorio: `false`

Tipo: `string`

Schema valido: `^\s*$|[0-9]+`

CapabilityConfiguration

L'insieme di configurazioni del collettore Modbus TCP da cui il connettore raccoglie i dati e a cui si connette.

Nome visualizzato nella console: AWS IoT CapabilityConfiguration

Obbligatorio: `false`

Tipo: una stringa JSON ben formata che definisce l'insieme di configurazioni di feedback supportate.

Di seguito è riportato un esempio di: `CapabilityConfiguration`

```
{
  "sources": [
    {
      "type": "ModBusTCPSource",
      "name": "SourceName1",
```

```

    "measurementDataStreamPrefix": "SourceName1_Prefix",
    "destination": {
      "type": "StreamManager",
      "streamName": "SiteWise_Stream_1",
      "streamBufferSize": 8
    },
    "endpoint": {
      "ipAddress": "127.0.0.1",
      "port": 8081,
      "unitId": 1
    },
    "propertyGroups": [
      {
        "name": "GroupName",
        "tagPathDefinitions": [
          {
            "type": "ModBusTCPAddress",
            "tag": "TT-001",
            "address": "30001",
            "size": 2,
            "srcDataType": "float",
            "transformation": "byteWordSwap",
            "dstDataType": "double"
          }
        ],
        "scanMode": {
          "type": "POLL",
          "rate": 100
        }
      }
    ]
  }
}

```

Esempio di creazione di un connettore (AWS CLI)

Il seguente comando CLI crea una versione iniziale che contiene il connettore Modbus-TCP ConnectorDefinition Protocol Adapter.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-version '1.0.0'
```

```
{
  "Connectors": [
    {
      "Id": "MyModbusTCPConnector",
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/ModbusTCP/versions/3",
      "Parameters": {
        "capability_configuration": "{\"version\":1,\"namespace\": \"iotsitewise:modbuscollector:1\", \"configuration\": {\"sources\": [{\"type\": \"ModBusTCPSource\", \"name\": \"SourceName1\", \"measurementDataStreamPrefix\": \"\", \"endpoint\": {\"ipAddress\": \"127.0.0.1\", \"port\": 8081, \"unitId\": 1}, \"propertyGroups\": [{\"name\": \"PropertyGroupName\", \"tagPathDefinitions\": [{\"type\": \"ModBusTCPAddress\", \"tag\": \"TT-001\", \"address\": \"30001\", \"size\": 2, \"srcDataType\": \"hexdump\", \"transformation\": \"noSwap\", \"dstDataType\": \"string\"}], \"scanMode\": {\"rate\": 200, \"type\": \"POLL\"}}], \"destination\": {\"type\": \"StreamManager\", \"streamName\": \"SiteWise_Stream\", \"streamBufferSize\": 10, \"minimumInterRequestDuration\": 200}}]}\""}
      }
    ]
  }
}
```

Note

La funzione Lambda in questo connettore ha un ciclo di vita di [lunga durata](#).

Dati di input

Questo connettore non accetta messaggi MQTT come dati di input.

Dati di output

Questo connettore pubblica dati su `StreamManager`. È necessario configurare il flusso di messaggi di destinazione. I messaggi di output hanno la seguente struttura:

```
{
  "alias": "string",
  "messages": [
    {
      "name": "string",
      "value": boolean|double|integer|string,
```

```
        "timestamp": number,  
        "quality": "string"  
    }  
]  
}
```

Licenze

Il connettore Modbus-TCP Protocol Adapter include i seguenti software/licenze di terze parti:

- [Petri Modbus](#) digitale

Questo connettore è rilasciato ai sensi del contratto di [licenza del software Greengrass Core](#).

Changelog

La tabella seguente descrive le modifiche apportate a ciascuna versione del connettore.

Versione	Modifiche	Data
3 (consigliato)	Questa versione contiene correzioni di bug.	22 dicembre 2021
2	È stato aggiunto il supporto per ASCII e UTF8 stringhe sorgente ISO8859 codificate.	24 maggio 2021
1	Versione iniziale.	15 dicembre 2020

Un gruppo Greengrass può contenere una sola versione del connettore alla volta. Per informazioni sull'aggiornamento di una versione del connettore, consulta [the section called “Aggiornamento delle versioni dei connettori”](#).

Consulta anche

- [Integrazione con servizi e protocolli tramite i connettori](#)
- [the section called “Nozioni di base sui connettori \(console\)”](#)
- [the section called “Nozioni di base sui connettori \(CLI\)”](#)

Connettore GPIO Raspberry Pi

Warning

Questo connettore è entrato nella fase di vita prolungata e AWS IoT Greengrass non rilascerà aggiornamenti che forniscano funzionalità, miglioramenti alle funzionalità esistenti, patch di sicurezza o correzioni di bug. Per ulteriori informazioni, consulta [AWS IoT Greengrass Version 1 politica di manutenzione](#).

Il [connettore](#) GPIO Raspberry Pi controlla i pin di input/output (GPIO) generici su un dispositivo principale Raspberry Pi.

Questo connettore esegue il polling dei pin di input a un intervallo specificato e pubblica le modifiche allo stato negli argomenti MQTT. Accetta anche richieste di lettura e scrittura come messaggi MQTT da funzioni Lambda definite dall'utente. Le richieste di scrittura vengono utilizzate per impostare il pin sull'alta o bassa tensione.

Il connettore fornisce parametri utilizzabili per indicare i pin di input e output. Questo comportamento viene configurato prima della distribuzione del gruppo. Non può essere modificato in fase di runtime.

- I pin di input possono essere utilizzati per ricevere i dati da dispositivi periferici.
- I pin di output possono essere utilizzati per controllare le periferiche o inviare dati alle periferiche.

Puoi utilizzare questo connettore per molti scenari, ad esempio:

- Controllo delle luci LED verde, gialle e rosse di un semaforo.
- Controllo di una ventola (collegata a un relè elettrico) in base ai dati forniti da un sensore di umidità.
- Informare i dipendenti di un negozio sul fatto che i clienti hanno premuto un pulsante.
- Utilizzo di un interruttore smart per controllare altri dispositivi IoT.

Note

Questo connettore non è idoneo per applicazioni con requisiti in tempo reale. Gli eventi di breve durata potrebbero non venire rilevati.

Questo connettore ha le seguenti versioni.

Versione	ARN
3	arn:aws:greengrass: <i>region</i> ::/ connectors/RaspberryPiGPIO/ versions/3
2	arn:aws:greengrass: <i>region</i> ::/ connectors/RaspberryPiGPIO/ versions/2
1	arn:aws:greengrass: <i>region</i> ::/ connectors/RaspberryPiGPIO/ versions/1

Per informazioni sulle modifiche di ogni versione, consulta [Changelog](#).

Requisiti

Questo connettore presenta i seguenti requisiti:

Version 3

- AWS IoT Greengrass Software principale v1.9.3 o successivo.
- [Python](#) versione 3.7 installata sul dispositivo principale e aggiunta alla variabile di ambiente PATH.
- Raspberry Pi 4 modello B o Raspberry Pi 3 modello B/B+. È necessario conoscere la sequenza pin del Raspberry Pi. Per ulteriori informazioni, consulta [the section called “Sequenza pin GPIO”](#).
- Una [risorsa del dispositivo locale](#) nel gruppo Greengrass che rimanda al /dev/gpiomem Raspberry Pi. Se si crea la risorsa nella console, è necessario selezionare l'opzione Aggiungi automaticamente le autorizzazioni del gruppo di sistema operativo del gruppo Linux proprietario della risorsa. Nell'API, imposta la `GroupOwnerSetting.AutoAddGroupOwner` proprietà su `true`.
- Il [RPimodulo.GPIO](#) installato sul Raspberry Pi. In Raspbian, questo modulo è installato per impostazione predefinita. Puoi utilizzare il seguente comando per reinstallarlo:

```
sudo pip install RPi.GPIO
```

Versions 1 - 2

- AWS IoT Greengrass Software di base v1.7 o successivo.
- [Python](#) versione 2.7 installato sul dispositivo principale e aggiunto alla variabile di ambiente PATH.
- Raspberry Pi 4 modello B o Raspberry Pi 3 modello B/B+. È necessario conoscere la sequenza pin del Raspberry Pi. Per ulteriori informazioni, consulta [the section called “Sequenza pin GPIO”](#).
- Una [risorsa del dispositivo locale](#) nel gruppo Greengrass che rimanda al /dev/gpiomem Raspberry Pi. Se si crea la risorsa nella console, è necessario selezionare l'opzione Aggiungi automaticamente le autorizzazioni del gruppo di sistema operativo del gruppo Linux proprietario della risorsa. Nell'API, imposta la GroupOwnerSetting.AutoAddGroupOwner proprietà su true.
- Il [RPImodulo.GPIO](#) installato sul Raspberry Pi. In Raspbian, questo modulo è installato per impostazione predefinita. Puoi utilizzare il seguente comando per reinstallarlo:

```
sudo pip install RPi.GPIO
```

Sequenza pin GPIO

Il connettore GPIO Raspberry Pi fa riferimento ai pin GPIO in base allo schema di numerazione del System on Chip (SoC) sottostante, non in base al layout fisico dei pin GPIO. L'ordine fisico dei pin può variare nelle versioni di Raspberry Pi. Per ulteriori informazioni, consulta [GPIO](#) nella documentazione di Raspberry Pi.

Il connettore non è in grado di verificare se i pin di input e output in fase di configurazione sono associati correttamente all'hardware sottostante del Raspberry Pi. Se la configurazione dei pin non è valida, il connettore restituisce un errore di runtime al momento dell'avvio nel dispositivo. Per risolvere il problema, riconfigurare il connettore, quindi ridistribuirlo.

 Note

Assicurati che le periferiche dei pin GPIO siano correttamente cablate per evitare danni ai componenti.

Parametri del connettore

Questo connettore fornisce i seguenti parametri:

InputGpios

Un elenco separato da virgole di numeri pin GPIO da configurare come input. Se lo desideri, puoi aggiungere U per impostare una resistenza pull-up del pin oppure D per impostare la resistenza pull-down. Esempio: "5,6U,7D".

Nome visualizzato nella AWS IoT console: pin GPIO di input

Obbligatorio: false. Devi specificare i pin di input, quelli di output o entrambi.

Tipo: string

Modello valido: `^$|^[0-9]+[UD]?([0-9]+[UD]?)*$`

InputPollPeriod

L'intervallo (in millisecondi) fra ciascuna operazione di polling, che controlla le modifiche dello stato dei pin GPIO di input. Il valore minimo è 1.

Questo valore dipende dallo scenario e dal tipo di dispositivi in fase di polling. Ad esempio, un valore pari a 50 dovrebbe essere sufficientemente rapido per rilevare la pressione di un pulsante.

Nome visualizzato nella AWS IoT console: periodo di polling GPIO di input

Obbligatorio: false

Tipo: string

Schema valido: `^$|^[1-9][0-9]*$`

OutputGpios

Un elenco separato da virgole di numeri pin GPIO da configurare come output. Puoi aggiungere H per impostare uno stato elevato (1) oppure L per impostare uno stato basso (0). Esempio: "8H,9,27L".

Nome visualizzato nella AWS IoT console: pin GPIO di output

Obbligatorio: false. Devi specificare i pin di input, quelli di output o entrambi.

Tipo: string

Modello valido: `^[0-9]+[HL]?([0-9]+[HL]?)*$`

GpioMem-ResourceId

L'ID della risorsa del dispositivo locale che rappresenta `/dev/gpiomem`.

Note

Al connettore è concesso l'accesso in lettura e scrittura alla risorsa.

Nome visualizzato nella AWS IoT console: Resource for `/dev/gpiomem` device

Obbligatorio: true

Tipo: string

Schema valido: `.+`

Esempio di creazione di un connettore (AWS CLI)

Il seguente comando CLI crea un `ConnectorDefinition` con una versione iniziale che contiene il connettore GPIO Raspberry Pi.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-  
version '{  
  "Connectors": [  
    {
```

```
"Id": "MyRaspberryPiGPIOConnector",
"ConnectorArn": "arn:aws:greengrass:region::/connectors/RaspberryPiGPIO/
versions/3",
"Parameters": {
  "GpioMem-ResourceId": "my-gpio-resource",
  "InputGpios": "5,6U,7D",
  "InputPollPeriod": 50,
  "OutputGpios": "8H,9,27L"
}
}
```

Note

La funzione Lambda in questo connettore ha un ciclo di vita di [lunga durata](#).

Nella AWS IoT Greengrass console, puoi aggiungere un connettore dalla pagina Connettori del gruppo. Per ulteriori informazioni, consulta [the section called “Nozioni di base sui connettori \(console\)”](#).

Dati di input

Questo connettore accetta richieste di lettura o scrittura per i pin GPIO in due argomenti MQTT.

- Richieste di lettura nell'argomento `gpio/+/
read`.
- Richieste di scrittura nell'argomento `gpio/+/
write`.

Per pubblicare in questi argomenti, sostituisci i caratteri jolly + rispettivamente con il nome dell'oggetto core e il numero di pin di destinazione. Per esempio:

```
gpio/core-thing-name/gpio-number/read
```

Note

Attualmente, quando si crea un abbonamento che utilizza il connettore GPIO Raspberry Pi, è necessario specificare un valore per almeno una delle + wildcard presenti nell'argomento.

Filtro di argomenti: gpio/+ /+ /read

Utilizza questo argomento per indicare al connettore di leggere lo stato del pin GPIO specificato nell'argomento.

Il connettore pubblica la risposta nel corrispondente argomento di output (ad esempio, gpio/*core-thing-name*/gpio-number/state).

Proprietà dei messaggi

Nessuna. I messaggi inviati a questo argomento vengono ignorati.

Filtro di argomenti: gpio/+ /+ /write

Utilizza questo argomento per inviare richieste di scrittura a un pin GPIO. Indica al connettore di impostare il pin GPIO specificato nell'argomento sulla bassa tensione o sull'alta tensione.

- 0 imposta il pin sulla bassa tensione.
- 1 imposta il pin sull'alta tensione.

Il connettore pubblica la risposta nel corrispondente argomento /state di output (ad esempio, gpio/*core-thing-name*/gpio-number/state).

Proprietà dei messaggi

Il valore 0 o 1, come intero o stringa.

Input di esempio

0

Dati di output

Questo connettore pubblica i dati in due argomenti:

- Le modifiche allo stato alto o basso nell'argomento gpio/+ /+ /state.
- Gli errori nell'argomento gpio/+ /error.

Filtro di argomenti: gpio/+ /+ /state

Utilizza questo argomento per ascoltare le modifiche dello stato dei pin di input e risposte alle richieste di lettura. Il connettore restituisce la stringa "0" se lo stato del pin è basso oppure "1" se lo stato è alto.

Durante la pubblicazione in questo argomento, il connettore sostituisce i caratteri jolly + rispettivamente con il nome dell'oggetto core e il numero di pin di destinazione. Per esempio:

```
gpio/core-thing-name/gpio-number/state
```

Note

Attualmente, quando si crea un abbonamento che utilizza il connettore GPIO Raspberry Pi, è necessario specificare un valore per almeno uno degli + caratteri jolly presenti nell'argomento.

Output di esempio

```
0
```

Filtro di argomenti: `gpio/+/error`

Utilizza questo argomento per ascoltare gli errori. Il connettore pubblica in questo argomento in seguito a una richiesta non valida (ad esempio, quando è necessaria una modifica dello stato di un pin di input).

Durante la pubblicazione in questo argomento, il connettore sostituisce il carattere jolly + con il nome dell'oggetto core.

Output di esempio

```
{
  "topic": "gpio/my-core-thing/22/write",
  "error": "Invalid GPIO operation",
  "long_description": "GPIO 22 is configured as an INPUT GPIO. Write operations
are not permitted."
}
```

Esempio di utilizzo

Usa i seguenti passaggi di alto livello per configurare una funzione Lambda di esempio di Python 3.7 che puoi usare per provare il connettore.

 Note

- Se usi altri runtime Python, puoi creare un collegamento simbolico da Python3.x a Python 3.7.
- Gli argomenti [Nozioni di base sui connettori \(console\)](#) e [Nozioni di base sui connettori \(CLI\)](#) contengono passaggi dettagliati che illustrano come configurare e distribuire un connettore Twilio Notifications di esempio.

1. Assicurarsi di soddisfare i [requisiti](#) per il connettore.
2. Crea e pubblica una funzione Lambda che invia dati di input al connettore.

Salvare il [codice di esempio](#) come file PY. Scarica e decomprimi il [AWS IoT Greengrass Core SDK per Python](#). Quindi, crea un pacchetto zip che contiene il file PY e la cartella greengrasssdk a livello root. Questo pacchetto zip è il pacchetto di distribuzione in cui carichi AWS Lambda

Dopo aver creato la funzione Python 3.7 Lambda, pubblica una versione della funzione e crea un alias.

3. Configurare il gruppo Greengrass.
 - a. Aggiungi la funzione Lambda tramite il relativo alias (consigliato). Configura il ciclo di vita Lambda come longevo (o nella "Pinned": true CLI).
 - b. Aggiungi la risorsa del dispositivo locale richiesta e concedi l'accesso in lettura/scrittura alla funzione Lambda.
 - c. Aggiungere il connettore e configurarne i relativi [parametri](#).
 - d. Aggiungere sottoscrizioni che consentono al connettore di ricevere [i dati di input](#) e inviare [i dati di output](#) nei filtri degli argomenti supportati.
 - Imposta la funzione Lambda come origine, il connettore come destinazione e utilizza un filtro per argomenti di input supportato.
 - Imposta il connettore come origine, AWS IoT Core come destinazione e utilizza un filtro per l'argomento di output supportato. Utilizzi questo abbonamento per visualizzare i messaggi di stato nella AWS IoT console.
4. Distribuisci il gruppo.

5. Nella AWS IoT console, nella pagina Test, sottoscrivi l'argomento relativo ai dati di output per visualizzare i messaggi di stato dal connettore. La funzione Lambda di esempio è di lunga durata e inizia a inviare messaggi subito dopo l'implementazione del gruppo.

Al termine del test, puoi impostare il ciclo di vita Lambda su richiesta (o nella CLI) e "Pinned": false distribuire il gruppo. Ciò impedisce alla funzione di inviare messaggi.

Esempio

L'esempio seguente della funzione Lambda invia un messaggio di input al connettore. Questo esempio invia richieste di lettura per un set di pin GPIO di input. Illustra come costruire argomenti utilizzando il nome dell'oggetto core e il numero di pin.

```
import greengrasssdk
import json
import os

iot_client = greengrasssdk.client('iot-data')
INPUT_GPIOIS = [6, 17, 22]

thingName = os.environ['AWS_IOT_THING_NAME']

def get_read_topic(gpio_num):
    return '/'.join(['gpio', thingName, str(gpio_num), 'read'])

def get_write_topic(gpio_num):
    return '/'.join(['gpio', thingName, str(gpio_num), 'write'])

def send_message_to_connector(topic, message=''):
    iot_client.publish(topic=topic, payload=str(message))

def set_gpio_state(gpio, state):
    send_message_to_connector(get_write_topic(gpio), str(state))

def read_gpio_state(gpio):
    send_message_to_connector(get_read_topic(gpio))

def publish_basic_message():
    for i in INPUT_GPIOIS:
        read_gpio_state(i)

publish_basic_message()
```

```
def lambda_handler(event, context):  
    return
```

Licenze

Il connettore Raspberry Pi GPIO; include i seguenti software/licenze di terze parti:

- [RPI.GPIO/MIT](#)

Questo connettore è rilasciato ai sensi del contratto di [licenza del software Greengrass Core](#).

Changelog

La tabella seguente descrive le modifiche apportate a ciascuna versione del connettore.

Versione	Modifiche
3	È stato aggiornato il runtime Lambda a Python 3.7, che modifica i requisiti di runtime.
2	Arn del connettore aggiornato per Regione AWS il supporto.
1	Versione iniziale.

Un gruppo Greengrass può contenere una sola versione del connettore alla volta. Per informazioni sull'aggiornamento di una versione del connettore, consulta [the section called “Aggiornamento delle versioni dei connettori”](#).

Consulta anche

- [Integrazione con servizi e protocolli tramite i connettori](#)
- [the section called “Nozioni di base sui connettori \(console\)”](#)
- [the section called “Nozioni di base sui connettori \(CLI\)”](#)
- [GPIO](#) nella documentazione di Raspberry Pi

Connettore Serial Stream

Warning

Questo connettore è entrato nella fase di vita prolungata e AWS IoT Greengrass non rilascerà aggiornamenti che forniscano funzionalità, miglioramenti alle funzionalità esistenti, patch di sicurezza o correzioni di bug. Per ulteriori informazioni, consulta [AWS IoT Greengrass Version 1 politica di manutenzione](#).

Il [connettore](#) Serial Stream legge e scrive su una porta seriale di un dispositivo principale. AWS IoT Greengrass

Questo connettore supporta due modalità di funzionamento:

- Read-On-Demand. Riceve richieste di lettura e scrittura su argomenti MQTT e pubblica la risposta dell'operazione di lettura o lo stato dell'operazione di scrittura.
- Polling-Read. Effettua letture dalla porta seriale a intervalli regolari. Questa modalità supporta anche Read-On-Demand le richieste.

Note

Le richieste di lettura possono avere una lunghezza massima di 63994 byte. Le richieste di scrittura possono avere una lunghezza massima dei dati di 128000 byte.

Questo connettore ha le seguenti versioni.

Versione	ARN
3	arn:aws:greengrass: <i>region</i> ::/connectors/SerialStream/versions/3
2	arn:aws:greengrass: <i>region</i> ::/connectors/SerialStream/versions/2

Versione	ARN
1	arn:aws:greengrass: <i>region</i> ::/ connectors/SerialStream/ versions/1

Per informazioni sulle modifiche di ogni versione, consulta [Changelog](#).

Requisiti

Questo connettore presenta i seguenti requisiti:

Version 3

- AWS IoT Greengrass Software principale v1.9.3 o successivo.
- [Python](#) versione 3.7 o 3.8 installata sul dispositivo principale e aggiunta alla variabile di ambiente PATH.

Note

Per usare Python 3.8, esegui il seguente comando per creare un collegamento simbolico dalla cartella di installazione predefinita di Python 3.7 ai binari Python 3.8 installati.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Questo configura il dispositivo in modo che soddisfi il requisito Python per AWS IoT Greengrass.

- Una [risorsa del dispositivo locale](#) del gruppo Greengrass che punta alla porta seriale di destinazione.

Note

Prima di distribuire il connettore, ti consigliamo di configurare la porta seriale e di verificare che sia possibile effettuarvi operazioni di lettura e scrittura.

Versions 1 - 2

- AWS IoT Greengrass Software principale v1.7 o successivo.
- [Python](#) versione 2.7 installato sul dispositivo principale e aggiunto alla variabile di ambiente PATH.
- Una [risorsa del dispositivo locale](#) del gruppo Greengrass che punta alla porta seriale di destinazione.



Note

Prima di distribuire il connettore, ti consigliamo di configurare la porta seriale e di verificare che sia possibile effettuarvi operazioni di lettura e scrittura.

Parametri del connettore

Questo connettore fornisce i seguenti parametri:

BaudRate

La velocità in baud della connessione seriale.

Nome visualizzato nella AWS IoT console: baud rate

Obbligatorio: true

Tipo: string

Valori validi: 110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 56000, 57600, 115200, 230400

Schema valido: ^110\$|^300\$|^600\$|^1200\$|^2400\$|^4800\$|^9600\$|^14400\$|^19200\$|^28800\$|^38400\$|^56000\$|^57600\$|^115200\$|^230400\$

Timeout

Il timeout (in secondi) di un'operazione di lettura.

Nome visualizzato nella AWS IoT console: Timeout

Obbligatorio: true

Tipo: `string`

Valori validi: 1 - 59

Schema valido: `^([1-9]|[1-5][0-9])$`

SerialPort

Il percorso assoluto della porta seriale fisica sul dispositivo. Si tratta del percorso di origine specificato per la risorsa del dispositivo locale.

Nome visualizzato nella AWS IoT console: porta seriale

Obbligatorio: `true`

Tipo: `string`

Schema valido: `[/a-zA-Z0-9_-]+`

SerialPort-ResourceId

L'ID della risorsa del dispositivo locale fisico che rappresenta la porta seriale fisica.

Note

Al connettore è concesso l'accesso in lettura e scrittura alla risorsa.

Nome visualizzato nella AWS IoT console: risorsa della porta seriale

Richiesto: `true`

Tipo: `string`

Schema valido: `[a-zA-Z0-9_-]+`

PollingRead

Imposta la modalità di lettura: Polling-Read o. Read-On-Demand

- Per la modalità Polling-Read, specificare `true`. In questa modalità, sono obbligatorie le proprietà `PollingInterval`, `PollingReadType` e `PollingReadLength`.
- Per la Read-On-Demand modalità, specificare `false`. In questa modalità, il tipo e i valori di lunghezza sono specificati nella richiesta di lettura.

Nome visualizzato nella AWS IoT console: modalità di lettura

Obbligatorio: true

Tipo: string

Valori validi: true, false

Schema valido: `^([Tt][Rr][Uu][Ee]|[Ff][Aa][Ll][Ss][Ee])$`

PollingReadLength

La lunghezza dei dati (in byte) da leggere in ciascuna operazione di lettura di polling. Si applica solo quando si utilizza la modalità Polling-Read.

Nome visualizzato nella AWS IoT console: lunghezza della lettura del sondaggio

Obbligatorio: false. Questa proprietà è obbligatoria quando PollingRead è true.

Tipo: string

Modello valido: `^(|[1-9][0-9]{0,3}|[1-5][0-9]{4}|6[0-2][0-9]{3}|63[0-8][0-9]{2}|639[0-8][0-9]|6399[0-4])$`

PollingReadInterval

L'intervallo di tempo (in secondi) in cui il avviene la lettura di polling. Si applica solo quando si utilizza la modalità Polling-Read.

Nome visualizzato nella AWS IoT console: Intervallo di lettura dei sondaggi

Obbligatorio: false. Questa proprietà è obbligatoria quando PollingRead è true.

Tipo: string

Valori validi: 1 - 999

Modello valido: `^(|[1-9]|[1-9][0-9]|[1-9][0-9][0-9])$`

PollingReadType

Il tipo di dati letto dal thread di polling. Si applica solo quando si utilizza la modalità Polling-Read.

Nome visualizzato nella AWS IoT console: tipo di lettura del sondaggio

Obbligatorio: false. Questa proprietà è obbligatoria quando PollingRead è true.

Tipo: `string`

Valori validi: `ascii`, `hex`

Modello valido: `^(|[Aa][Ss][Cc][Ii][Ii]|[Hh][Ee][Xx])$`

RtsCts

Indica se abilitare o meno il controllo del flusso RTS/CTS. Il valore predefinito è `false`. Per ulteriori informazioni, consulta [RTS, CTS e RTR](#).

Nome visualizzato nella AWS IoT console: controllo del flusso RTS/CTS

Obbligatorio: `false`

Tipo: `string`

Valori validi: `true`, `false`

Schema valido: `^(|[Tt][Rr][Uu][Ee]|[Ff][Aa][Ll][Ss][Ee])$`

XonXoff

Indica se abilitare o meno il controllo del flusso del software. Il valore predefinito è `false`. Per ulteriori informazioni, consulta [Controllo del flusso software](#).

Nome visualizzato nella AWS IoT console: Software flow control

Obbligatorio: `false`

Tipo: `string`

Valori validi: `true`, `false`

Schema valido: `^(|[Tt][Rr][Uu][Ee]|[Ff][Aa][Ll][Ss][Ee])$`

Parity

La parità della porta seriale. Il valore predefinito è `N`. Per ulteriori informazioni, consulta [Parità](#).

Nome visualizzato nella AWS IoT console: parità della porta seriale

Obbligatorio: `false`

Tipo: `string`

Valori validi: N, E, O, S, M

Schema valido: `^(|[NEOSMneosm])$`

Esempio di creazione di un connettore (AWS CLI)

Il seguente comando CLI crea un `ConnectorDefinition` con una versione iniziale che contiene il connettore Serial Stream. Configura il connettore per la modalità Polling-Read.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
  "Connectors": [
    {
      "Id": "MySerialStreamConnector",
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/SerialStream/
versions/3",
      "Parameters": {
        "BaudRate" : "9600",
        "Timeout" : "25",
        "SerialPort" : "/dev/serial1",
        "SerialPort-ResourceId" : "my-serial-port-resource",
        "PollingRead" : "true",
        "PollingReadLength" : "30",
        "PollingReadInterval" : "30",
        "PollingReadType" : "hex"
      }
    }
  ]
}'
```

Nella AWS IoT Greengrass console, puoi aggiungere un connettore dalla pagina Connettori del gruppo. Per ulteriori informazioni, consulta [the section called “Nozioni di base sui connettori \(console\)”](#).

Dati di input

Questo connettore accetta richieste di lettura o scrittura per porte seriali su due argomenti MQTT. I messaggi di input devono essere in formato JSON.

- Richieste di lettura nell'argomento `serial/+/read/#`.
- Richieste di scrittura nell'argomento `serial/+/write/#`.

Per pubblicare in questi argomenti, sostituire il carattere jolly + con il nome oggetto del core e il carattere jolly # con il percorso della porta seriale. Per esempio:

```
serial/core-thing-name/read/dev/serial-port
```

Filtro di argomenti: serial/+/read/#

Utilizza questo argomento per inviare richieste di lettura on demand a un pin seriale. Le richieste di lettura possono avere una lunghezza massima di 63994 byte.

Proprietà dei messaggi

readLength

La lunghezza dei dati da leggere dalla porta seriale.

Obbligatorio: true

Tipo: string

Schema valido: `^[1-9][0-9]*$`

type

Il tipo di dati da leggere.

Richiesto: true

Tipo: string

Valori validi: `ascii`, `hex`

Schema valido: `(?i)^(ascii|hex)$`

id

Un ID arbitrario della richiesta. Questa proprietà viene utilizzata per associare una richiesta di input a una risposta di output.

Richiesto: false

Tipo: string

Schema valido: `.+`

Input di esempio

```
{
  "readLength": "30",
  "type": "ascii",
  "id": "abc123"
}
```

Filtro di argomenti: `serial/+/write/#`

Utilizza questo argomento per inviare richieste di scrittura a un pin seriale. Le richieste di scrittura possono avere una lunghezza massima dei dati di 128000 byte.

Proprietà dei messaggi

`data`

La stringa da scrivere nella porta seriale.

Richiesto: `true`

Tipo: `string`

Schema valido: `^[1-9][0-9]*$`

`type`

Il tipo di dati da leggere.

Richiesto: `true`

Tipo: `string`

Valori validi: `ascii`, `hex`

Schema valido: `^(ascii|hex|ASCII|HEX)$`

`id`

Un ID arbitrario della richiesta. Questa proprietà viene utilizzata per associare una richiesta di input a una risposta di output.

Richiesto: `false`

Tipo: `string`

Schema valido: `.+`

Input di esempio: richiesta ASCII

```
{
  "data": "random serial data",
  "type": "ascii",
  "id": "abc123"
}
```

Input di esempio: richiesta esadecimale

```
{
  "data": "base64 encoded data",
  "type": "hex",
  "id": "abc123"
}
```

Dati di output

Il connettore pubblica i dati di output in due argomenti:

- Le informazioni sullo stato del connettore nell'argomento `serial/+/status/#`.
- Le risposte delle richieste di lettura nell'argomento `serial/+/read_response/#`.

Durante la pubblicazione in questo argomento, il connettore sostituisce il carattere jolly + con il nome oggetto del core e il carattere jolly # con il percorso della porta seriale. Per esempio:

```
serial/core-thing-name/status/dev/serial-port
```

Filtro di argomenti: `serial/+/status/#`

Utilizza questo argomento per ascoltare lo stato delle richieste di lettura e scrittura. Se la richiesta include una proprietà `id`, verrà restituita nella risposta.

Output di esempio: Operazione riuscita

```
{
  "response": {
    "status": "success"
  },
  "id": "abc123"
}
```

```
}
```

Esempio di output: Errore

Una risposta di errore include la proprietà `error_message` che descrive l'errore o il timeout riscontrato durante l'esecuzione dell'operazione di lettura o scrittura.

```
{
  "response": {
    "status": "fail",
    "error_message": "Could not write to port"
  },
  "id": "abc123"
}
```

Filtro di argomenti: `serial/+/read_response/#`

Utilizza questo argomento per ricevere risposta dei dati da un'operazione di lettura. I dati della risposta hanno la codifica Base64 se il tipo è hex.

Output di esempio

```
{
  "data": "output of serial read operation"
  "id": "abc123"
}
```

Esempio di utilizzo

Usa i seguenti passaggi di alto livello per configurare una funzione Lambda di esempio di Python 3.7 che puoi usare per provare il connettore.

Note

- Se usi altri runtime Python, puoi creare un collegamento simbolico da Python3.x a Python 3.7.
- Gli argomenti [Nozioni di base sui connettori \(console\)](#) e [Nozioni di base sui connettori \(CLI\)](#) contengono passaggi dettagliati che illustrano come configurare e distribuire un connettore Twilio Notifications di esempio.

1. Assicurarsi di soddisfare i [requisiti](#) per il connettore.
2. Crea e pubblica una funzione Lambda che invia dati di input al connettore.

Salvare il [codice di esempio](#) come file PY. Scarica e decomprimi il [AWS IoT Greengrass Core SDK per Python](#). Quindi, crea un pacchetto zip che contiene il file PY e la cartella greengrasssdk a livello root. Questo pacchetto zip è il pacchetto di distribuzione in cui carichi AWS Lambda

Dopo aver creato la funzione Python 3.7 Lambda, pubblica una versione della funzione e crea un alias.

3. Configurare il gruppo Greengrass.
 - a. Aggiungi la funzione Lambda tramite il relativo alias (consigliato). Configura il ciclo di vita Lambda come longevo (o nella "Pinned": `true` CLI).
 - b. Aggiungi la risorsa del dispositivo locale richiesta e concedi l'accesso in lettura/scrittura alla funzione Lambda.
 - c. Aggiungere il connettore e configurarne i relativi [parametri](#).
 - d. Aggiungere sottoscrizioni al gruppo che consentono al connettore di ricevere [i dati di input](#) e inviare [i dati di output](#) nei filtri degli argomenti supportati.
 - Imposta la funzione Lambda come origine, il connettore come destinazione e utilizza un filtro per argomenti di input supportato.
 - Imposta il connettore come origine, AWS IoT Core come destinazione e utilizza un filtro per l'argomento di output supportato. Utilizzi questo abbonamento per visualizzare i messaggi di stato nella AWS IoT console.
4. Distribuisci il gruppo.
5. Nella AWS IoT console, nella pagina Test, sottoscrivi l'argomento relativo ai dati di output per visualizzare i messaggi di stato dal connettore. La funzione Lambda di esempio è di lunga durata e inizia a inviare messaggi subito dopo l'implementazione del gruppo.

Al termine del test, puoi impostare il ciclo di vita Lambda su richiesta (o nella CLI) e "Pinned": `false` distribuire il gruppo. Ciò impedisce alla funzione di inviare messaggi.

Esempio

L'esempio seguente della funzione Lambda invia un messaggio di input al connettore.

```
import greengrasssdk
import json

TOPIC_REQUEST = 'serial/CORE_THING_NAME/write/dev/serial1'

# Creating a greengrass core sdk client
iot_client = greengrasssdk.client('iot-data')

def create_serial_stream_request():
    request = {
        "data": "TEST",
        "type": "ascii",
        "id": "abc123"
    }
    return request

def publish_basic_request():
    iot_client.publish(payload=json.dumps(create_serial_stream_request()),
        topic=TOPIC_REQUEST)

publish_basic_request()

def lambda_handler(event, context):
    return
```

Licenze

Il connettore Serial Stream include i seguenti software/licenze di terze parti:

- [pyserial](#)/BSD

Questo connettore è rilasciato ai sensi del contratto di [licenza del software Greengrass Core](#).

Changelog

La tabella seguente descrive le modifiche apportate a ciascuna versione del connettore.

Versione	Modifiche
3	È stato aggiornato il runtime Lambda a Python 3.7, che modifica i requisiti di runtime.

Versione	Modifiche
2	Arn del connettore aggiornato per Regione AWS il supporto.
1	Versione iniziale.

Un gruppo Greengrass può contenere una sola versione del connettore alla volta. Per informazioni sull'aggiornamento di una versione del connettore, consulta [the section called “Aggiornamento delle versioni dei connettori”](#).

Consulta anche

- [Integrazione con servizi e protocolli tramite i connettori](#)
- [the section called “Nozioni di base sui connettori \(console\)”](#)
- [the section called “Nozioni di base sui connettori \(CLI\)”](#)

ServiceNow MetricBase Connettore di integrazione

Warning

Questo connettore è entrato nella fase di vita prolungata e AWS IoT Greengrass non rilascerà aggiornamenti che forniscano funzionalità, miglioramenti alle funzionalità esistenti, patch di sicurezza o correzioni di bug. Per ulteriori informazioni, consulta [AWS IoT Greengrass Version 1 politica di manutenzione](#).

Il [connettore ServiceNow MetricBase](#) Integration pubblica le metriche delle serie temporali dai dispositivi Greengrass su. ServiceNow MetricBase In questo modo potrai archiviare, analizzare e visualizzare i dati delle serie temporali dall'ambiente core di Greengrass e agire sugli eventi locali.

Questo connettore riceve dati di serie temporali su un argomento MQTT e li pubblica nell'API a intervalli regolari. ServiceNow

Puoi utilizzare questo connettore per supportare molti scenari, ad esempio:

- Creazione di avvisi basati su soglie e allarmi basati sui dati delle serie temporali raccolti dai dispositivi Greengrass.
- Usa i dati dei servizi temporali dei dispositivi Greengrass con applicazioni personalizzate basate sulla ServiceNow piattaforma.

Questo connettore ha le seguenti versioni.

Versione	ARN
4	arn:aws:greengrass: <i>region</i> ::/ connectors/ServiceNowMetricBaseIntegration/versions/4
3	arn:aws:greengrass: <i>region</i> ::/ connectors/ServiceNowMetricBaseIntegration/versions/3
2	arn:aws:greengrass: <i>region</i> ::/ connectors/ServiceNowMetricBaseIntegration/versions/2
1	arn:aws:greengrass: <i>region</i> ::/ connectors/ServiceNowMetricBaseIntegration/versions/1

Per informazioni sulle modifiche di ogni versione, consulta [Changelog](#).

Requisiti

Questo connettore presenta i seguenti requisiti:

Version 3 - 4

- AWS IoT Greengrass Software principale v1.9.3 o successivo. AWS IoT Greengrass [deve essere configurato per supportare i segreti locali, come descritto in Secrets Requirements](#).

 Note

Questo requisito include l'autorizzazione all'accesso ai segreti di Secrets Manager. Se utilizzi il ruolo di servizio Greengrass predefinito, Greengrass è autorizzato a ottenere i valori dei segreti con nomi che iniziano con greengrass-.

- [Python](#) versione 3.7 o 3.8 installata sul dispositivo principale e aggiunta alla variabile di ambiente PATH.

 Note

Per usare Python 3.8, esegui il seguente comando per creare un collegamento simbolico dalla cartella di installazione predefinita di Python 3.7 ai binari Python 3.8 installati.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Questo configura il dispositivo in modo che soddisfi il requisito Python per AWS IoT Greengrass.

- Un account con un abbonamento attivato a ServiceNow MetricBase. Inoltre, nell'account è necessario creare un parametro e una tabella dei parametri. Per ulteriori informazioni, [MetricBase](#) consulta la ServiceNow documentazione.
- Un tipo di testo segreto AWS Secrets Manager che memorizza il nome utente e la password per accedere all'istanza ServiceNow con l'autenticazione di base. Il segreto deve includere le chiavi "utente" e "password" con i valori corrispondenti. Per ulteriori informazioni, consulta [Creazione di un segreto di base](#) nella Guida AWS Secrets Manager per l'utente.
- Una risorsa segreta del gruppo Greengrass che fa riferimento al segreto di Secrets Manager. Per ulteriori informazioni, consulta [Distribuzione dei segreti nel core](#).

Versions 1 - 2

- AWS IoT Greengrass Software principale v1.7 o successivo. AWS IoT Greengrass deve essere configurato per supportare i segreti locali, come descritto in [Secrets Requirements](#).

 Note

Questo requisito include l'autorizzazione all'accesso ai segreti di Secrets Manager. Se utilizzi il ruolo di servizio Greengrass predefinito, Greengrass è autorizzato a ottenere i valori dei segreti con nomi che iniziano con greengrass-.

- [Python](#) versione 2.7 installato sul dispositivo principale e aggiunto alla variabile di ambiente PATH.
- Un ServiceNow account con un abbonamento attivato a MetricBase. Inoltre, nell'account è necessario creare un parametro e una tabella dei parametri. Per ulteriori informazioni, [MetricBase](#) consulta la ServiceNow documentazione.
- Un tipo di testo segreto AWS Secrets Manager che memorizza il nome utente e la password per accedere all'istanza ServiceNow con l'autenticazione di base. Il segreto deve includere le chiavi "utente" e "password" con i valori corrispondenti. Per ulteriori informazioni, consulta [Creazione di un segreto di base](#) nella Guida AWS Secrets Manager per l'utente.
- Una risorsa segreta del gruppo Greengrass che fa riferimento al segreto di Secrets Manager. Per ulteriori informazioni, consulta [Distribuzione dei segreti nel core](#).

Parametri del connettore

Questo connettore fornisce i seguenti parametri:

Version 4

`PublishInterval`

Il numero massimo di secondi di attesa tra gli eventi di pubblicazione ServiceNow. Il valore massimo è 900.

Il connettore pubblica ServiceNow quando `PublishBatchSize` viene raggiunto o `PublishInterval` scade.

Nome visualizzato nella AWS IoT console: intervallo di pubblicazione in secondi

Obbligatorio: `true`

Tipo: `string`

Valori validi: 1 - 900

Schema valido: [1-9] | [1-9]\d | [1-9]\d\d | 900

PublishBatchSize

Il numero massimo di valori delle metriche che possono essere raggruppati in batch prima di essere pubblicati su. ServiceNow

Il connettore pubblica ServiceNow quando PublishBatchSize viene raggiunto o scade.

PublishInterval

Nome visualizzato nella AWS IoT console: dimensione del batch di pubblicazione

Obbligatorio: true

Tipo: string

Schema valido: ^[0-9]+\$

InstanceName

Il nome dell'istanza utilizzata per la connessione ServiceNow.

Nome visualizzato nella AWS IoT console: nome dell' ServiceNow istanza

Obbligatorio: true

Tipo: string

Schema valido: .+

DefaultTableName

Il nome della tabella che contiene le informazioni GlideRecord associate al MetricBase database delle serie temporali. La proprietà table nel payload del messaggio di input può essere utilizzata per sostituire questo valore.

Nome visualizzato nella AWS IoT console: nome della tabella che contiene la metrica

Obbligatorio: true

Tipo: string

Schema valido: .+

MaxMetricsToRetain

Il numero massimo di parametri da salvare in memoria prima che vengano sostituiti con nuovi parametri.

Questo limite è valido quando non è presente una connessione a Internet e il connettore inizia il buffering dei parametri da pubblicare successivamente. Quando il buffer è pieno, i parametri meno recenti vengono sostituiti da quelli nuovi.

Note

I parametri non vengono salvati se si interrompe il processo host del connettore. Ad esempio, ciò potrebbe verificarsi durante la distribuzione dei gruppi o al riavvio del dispositivo.

Il valore deve essere superiore alle dimensioni del batch e abbastanza grande da contenere messaggi in base alla velocità in entrata dei messaggi MQTT.

Nome visualizzato nella AWS IoT console: numero massimo di metriche da conservare in memoria

Obbligatorio: `true`

Tipo: `string`

Schema valido: `^[0-9]+$`

AuthSecretArn

Il segreto AWS Secrets Manager che memorizza il nome ServiceNow utente e la password. Deve essere un segreto in formato testo. Il segreto deve includere le chiavi "utente" e "password" con i valori corrispondenti.

Nome visualizzato nella AWS IoT console: ARN del segreto di autenticazione

Obbligatorio: `true`

Tipo: `string`

Schema valido: `arn:aws:secretsmanager:[a-z0-9\-\_]+:[0-9]{12}:secret:([a-zA-Z0-9\-\_]+/)+/[a-zA-Z0-9\-\_]+`

AuthSecretArn-ResourceId

La risorsa segreta del gruppo che fa riferimento al segreto di Secrets Manager per le ServiceNow credenziali.

Nome visualizzato nella AWS IoT console: risorsa token di autenticazione

Obbligatorio: true

Tipo: string

Schema valido: .+

IsolationMode

Modalità di [containerizzazione](#) per questo connettore. L'impostazione predefinita è `GreengrassContainer`, il che significa che il connettore viene eseguito in un ambiente di runtime isolato all'interno del AWS IoT Greengrass contenitore.

Note

L'impostazione predefinita della containerizzazione per il gruppo non si applica ai connettori.

Nome visualizzato nella AWS IoT console: modalità di isolamento del contenitore

Obbligatorio: false

Tipo: string

Valori validi: `GreengrassContainer` o `NoContainer`

Schema valido: `^NoContainer$|^GreengrassContainer$`

Version 1 - 3

PublishInterval

Il numero massimo di secondi di attesa tra gli eventi di pubblicazione ServiceNow. Il valore massimo è 900.

Il connettore pubblica ServiceNow quando PublishBatchSize viene raggiunto o PublishInterval scade.

Nome visualizzato nella AWS IoT console: intervallo di pubblicazione in secondi

Obbligatorio: true

Tipo: string

Valori validi: 1 - 900

Schema valido: [1-9] | [1-9]\d | [1-9]\d\d | 900

PublishBatchSize

Il numero massimo di valori delle metriche che possono essere raggruppati in batch prima di essere pubblicati su. ServiceNow

Il connettore pubblica ServiceNow quando PublishBatchSize viene raggiunto o scade. PublishInterval

Nome visualizzato nella AWS IoT console: dimensione del batch di pubblicazione

Obbligatorio: true

Tipo: string

Schema valido: ^[0-9]+\$

InstanceName

Il nome dell'istanza utilizzata per la connessione ServiceNow.

Nome visualizzato nella AWS IoT console: nome dell' ServiceNow istanza

Obbligatorio: true

Tipo: string

Schema valido: .+

DefaultTableName

Il nome della tabella che contiene le informazioni GlideRecord associate al MetricBase database delle serie temporali. La proprietà table nel payload del messaggio di input può essere utilizzata per sostituire questo valore.

Nome visualizzato nella AWS IoT console: nome della tabella che contiene la metrica

Obbligatorio: true

Tipo: string

Schema valido: .+

MaxMetricsToRetain

Il numero massimo di parametri da salvare in memoria prima che vengano sostituiti con nuovi parametri.

Questo limite è valido quando non è presente una connessione a Internet e il connettore inizia il buffering dei parametri da pubblicare successivamente. Quando il buffer è pieno, i parametri meno recenti vengono sostituiti da quelli nuovi.

 Note

I parametri non vengono salvati se si interrompe il processo host del connettore. Ad esempio, ciò potrebbe verificarsi durante la distribuzione dei gruppi o al riavvio del dispositivo.

Il valore deve essere superiore alle dimensioni del batch e abbastanza grande da contenere messaggi in base alla velocità in entrata dei messaggi MQTT.

Nome visualizzato nella AWS IoT console: numero massimo di metriche da conservare in memoria

Obbligatorio: true

Tipo: string

Schema valido: ^[0-9]+\$

AuthSecretArn

Il segreto AWS Secrets Manager che memorizza il nome ServiceNow utente e la password. Deve essere un segreto in formato testo. Il segreto deve includere le chiavi "utente" e "password" con i valori corrispondenti.

Nome visualizzato nella AWS IoT console: ARN del segreto di autenticazione

Obbligatorio: true

Tipo: string

Schema valido: `arn:aws:secretsmanager:[a-z0-9\-\_]+:[0-9]{12}:secret:([a-zA-Z0-9\-\_]+/)*[a-zA-Z0-9/_+=,.\@-\_]+-[a-zA-Z0-9]+`

AuthSecretArn-ResourceId

La risorsa segreta del gruppo che fa riferimento al segreto di Secrets Manager per le ServiceNow credenziali.

Nome visualizzato nella AWS IoT console: risorsa token di autenticazione

Obbligatorio: true

Tipo: string

Schema valido: `.*`

Esempio di creazione di un connettore (AWS CLI)

Il seguente comando CLI crea un `ConnectorDefinition` con una versione iniziale che contiene il connettore di ServiceNow MetricBase integrazione.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
  "Connectors": [
    {
      "Id": "MyServiceNowMetricBaseIntegrationConnector",
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/
ServiceNowMetricBaseIntegration/versions/4",
      "Parameters": {
        "PublishInterval" : "10",
        "PublishBatchSize" : "50",
        "InstanceName" : "myinstance",
        "DefaultTableName" : "u_greengrass_app",
        "MaxMetricsToRetain" : "20000",
        "AuthSecretArn" : "arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret-hash",
```

```
        "AuthSecretArn-ResourceId" : "MySecretResource",  
        "IsolationMode" : "GreengrassContainer"  
    }  
]  
'
```

Note

La funzione Lambda in questo connettore ha un ciclo di vita di [lunga durata](#).

Nella AWS IoT Greengrass console, puoi aggiungere un connettore dalla pagina Connettori del gruppo. Per ulteriori informazioni, consulta [the section called “Nozioni di base sui connettori \(console\)”](#).

Dati di input

Questo connettore accetta le metriche delle serie temporali su un argomento MQTT e le pubblica su ServiceNow. I messaggi di input devono essere in formato JSON.

Filtro argomento in sottoscrizione

`servicenow/metricbase/metric`

Proprietà dei messaggi

`request`

Informazioni sulla tabella, sul record e sul parametro. Questa richiesta rappresenta l'oggetto `seriesRef` in una richiesta POST delle serie temporali. Per ulteriori informazioni, consulta [API delle serie temporali Clotho - POST](#).

Obbligatorio: `true`

Tipo: `object` che include le seguenti proprietà:

`subject`

Il valore `sys_id` del record specifico della tabella.

Obbligatorio: `true`

Tipo: string

`metric_name`

Il nome del campo del parametro.

Richiesto: true

Tipo: string

`table`

Il nome della tabella in cui memorizzare il record. Specifica questo valore per sostituire il parametro `DefaultTableName`.

Richiesto: false

Tipo: string

`value`

Il valore del punto dati individuale.

Richiesto: true

Tipo: float

`timestamp`

Il timestamp del punto dati individuale. Il valore predefinito è l'ora corrente.

Richiesto: false

Tipo: string

Input di esempio

```
{
  "request": {
    "subject": "ef43c6d40a0a0b5700c77f9bf387afe3",
    "metric_name": "u_count",
    "table": "u_greengrass_app"
    "value": 1.0,
    "timestamp": "2018-10-14T10:30:00"
  }
}
```

```
}
```

Dati di output

Questo connettore pubblica le informazioni di stato come dati di output su un argomento MQTT.

Filtro argomento in sottoscrizione

`servicenow/metricbase/metric/status`

Output di esempio: Operazione riuscita

```
{
  "response": {
    "metric_name": "Errors",
    "table_name": "GliderProd",
    "processed_on": "2018-10-14T10:35:00",
    "response_id": "khjKSkj132qwr23fcba",
    "status": "success",
    "values": [
      {
        "timestamp": "2016-10-14T10:30:00",
        "value": 1.0
      },
      {
        "timestamp": "2016-10-14T10:31:00",
        "value": 1.1
      }
    ]
  }
}
```

Esempio di output: Errore

```
{
  "response": {
    "error": "InvalidInputException",
    "error_message": "metric value is invalid",
    "status": "fail"
  }
}
```

 Note

Se il connettore rileva un errore riutilizzabile (ad esempio errori di connessione), riprova la pubblicazione nel batch successivo.

Esempio di utilizzo

Usa i seguenti passaggi di alto livello per configurare una funzione Lambda di esempio di Python 3.7 che puoi usare per provare il connettore.

 Note

- Se usi altri runtime Python, puoi creare un collegamento simbolico da Python3.x a Python 3.7.
- Gli argomenti [Nozioni di base sui connettori \(console\)](#) e [Nozioni di base sui connettori \(CLI\)](#) contengono passaggi dettagliati che illustrano come configurare e distribuire un connettore Twilio Notifications di esempio.

1. Assicurarsi di soddisfare i [requisiti](#) per il connettore.
2. Crea e pubblica una funzione Lambda che invia dati di input al connettore.

Salvare il [codice di esempio](#) come file PY. Scarica e decomprimi il [AWS IoT Greengrass Core SDK per Python](#). Quindi, crea un pacchetto zip che contiene il file PY e la cartella greengrasssdk a livello root. Questo pacchetto zip è il pacchetto di distribuzione in cui carichi AWS Lambda

Dopo aver creato la funzione Python 3.7 Lambda, pubblica una versione della funzione e crea un alias.

3. Configurare il gruppo Greengrass.
 - a. Aggiungi la funzione Lambda tramite il relativo alias (consigliato). Configura il ciclo di vita Lambda come longevo (o nella "Pinned": `true` CLI).
 - b. Aggiungi la risorsa segreta richiesta e concedi l'accesso in lettura alla funzione Lambda.
 - c. Aggiungere il connettore e configurarne i relativi [parametri](#).

- d. Aggiungere sottoscrizioni che consentono al connettore di ricevere [i dati di input](#) e inviare [i dati di output](#) nei filtri degli argomenti supportati.
 - Imposta la funzione Lambda come origine, il connettore come destinazione e utilizza un filtro per argomenti di input supportato.
 - Imposta il connettore come origine, AWS IoT Core come destinazione e utilizza un filtro per l'argomento di output supportato. Utilizzi questo abbonamento per visualizzare i messaggi di stato nella AWS IoT console.
4. Distribuisci il gruppo.
5. Nella AWS IoT console, nella pagina Test, sottoscrivi l'argomento relativo ai dati di output per visualizzare i messaggi di stato dal connettore. La funzione Lambda di esempio è di lunga durata e inizia a inviare messaggi subito dopo l'implementazione del gruppo.

Al termine del test, puoi impostare il ciclo di vita Lambda su richiesta (o nella CLI) e "Pinned": `false` distribuire il gruppo. Ciò impedisce alla funzione di inviare messaggi.

Esempio

L'esempio seguente della funzione Lambda invia un messaggio di input al connettore.

```
import greengrasssdk
import json

iot_client = greengrasssdk.client('iot-data')
SEND_TOPIC = 'servicenow/metricbase/metric'

def create_request_with_all_fields():
    return {
        "request": {
            "subject": '2efdf6badbd523803acfae441b961961',
            "metric_name": 'u_count',
            "value": 1234,
            "timestamp": '2018-10-20T20:22:20',
            "table": 'u_greengrass_metricbase_test'
        }
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
```

```
iot_client.publish(topic=SEND_TOPIC,  
                  payload=json.dumps(messageToPublish))  
  
publish_basic_message()  
  
def lambda_handler(event, context):  
    return
```

Licenze

Il connettore di ServiceNow MetricBase integrazione include i seguenti software/licenze di terze parti:

- [pysnow](#)/MIT

Questo connettore è rilasciato ai sensi del contratto di [licenza del software Greengrass Core](#).

Changelog

La tabella seguente descrive le modifiche apportate a ciascuna versione del connettore.

Versione	Modifiche
4	Aggiunto il parametro <code>IsolationMode</code> per configurare la modalità di containerizzazione per il connettore.
3	È stato aggiornato il runtime Lambda a Python 3.7, che modifica i requisiti di runtime.
2	Correggere per ridurre l'eccessiva registrazione di log.
1	Versione iniziale.

Un gruppo Greengrass può contenere una sola versione del connettore alla volta. Per informazioni sull'aggiornamento di una versione del connettore, consulta [the section called “Aggiornamento delle versioni dei connettori”](#).

Consulta anche

- [Integrazione con servizi e protocolli tramite i connettori](#)
- [the section called “Nozioni di base sui connettori \(console\)”](#)
- [the section called “Nozioni di base sui connettori \(CLI\)”](#)

Connettore SNS

Il [connettore](#) SNS pubblica messaggi su un argomento di Amazon SNS. Ciò consente ai server Web, agli indirizzi e-mail e agli altri abbonati al messaggio di rispondere agli eventi nel gruppo Greengrass.

Questo connettore riceve informazioni sui messaggi SNS in un argomento MQTT, quindi invia il messaggio a un argomento SNS specificato. Facoltativamente, puoi utilizzare funzioni Lambda personalizzate per implementare la logica di filtraggio o formattazione sui messaggi prima che vengano pubblicati su questo connettore.

Questo connettore ha le seguenti versioni.

Versione	ARN
4	arn:aws:greengrass: <i>region</i> ::/connectors/SNS/versions/4
3	arn:aws:greengrass: <i>region</i> ::/connectors/SNS/versions/3
2	arn:aws:greengrass: <i>region</i> ::/connectors/SNS/versions/2
1	arn:aws:greengrass: <i>region</i> ::/connectors/SNS/versions/1

Per informazioni sulle modifiche di ogni versione, consulta [Changelog](#).

Requisiti

Questo connettore presenta i seguenti requisiti:

Version 3 - 4

- AWS IoT Greengrass Software principale v1.9.3 o versione successiva.
- [Python](#) versione 3.7 o 3.8 installata sul dispositivo principale e aggiunta alla variabile di ambiente PATH.

 Note

Per usare Python 3.8, esegui il seguente comando per creare un collegamento simbolico dalla cartella di installazione predefinita di Python 3.7 ai binari Python 3.8 installati.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Questo configura il dispositivo in modo che soddisfi il requisito Python per AWS IoT Greengrass.

- Un argomento SNS configurato. Per le istruzioni, consulta [Creazione di un argomento Amazon SNS](#) nella Guida per lo Sviluppatore di Amazon Simple Notification Service.
- Il [ruolo del gruppo Greengrass](#) è configurato per consentire l'`sns:Publish` sull'Amazon di destinazione `SNSTopic`, come mostrato nel seguente esempio di policy IAM.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1528133056761",
      "Action": [
        "sns:Publish"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:sns:region:account-id:topic-name"
      ]
    }
  ]
}
```

Questo connettore consente di sostituire dinamicamente l'argomento predefinito nel payload del messaggio di input. Se l'implementazione utilizza questa funzionalità, la policy IAM deve consentire l'`sns:Publish` su tutti gli argomenti di destinazione. Puoi concedere alle risorse un accesso granulare o condizionale (ad esempio, utilizzando uno schema di denominazione con il carattere jolly *).

Per il requisito del ruolo di gruppo, è necessario configurare il ruolo in modo da concedere le autorizzazioni necessarie e assicurarsi che il ruolo sia stato aggiunto al gruppo. Per ulteriori informazioni, consulta [the section called “Gestione del ruolo del gruppo \(console\)”](#) o [the section called “Gestire il ruolo del gruppo \(CLI\)”](#).

Versions 1 - 2

- AWS IoT Greengrass Software di base v1.7 o successivo.
- [Python](#) versione 2.7 installato sul dispositivo principale e aggiunto alla variabile di ambiente PATH.
- Un argomento SNS configurato. Per le istruzioni, consulta [Creazione di un argomento Amazon SNS](#) nella Guida per lo Sviluppatore di Amazon Simple Notification Service.
- Il [ruolo del gruppo Greengrass](#) è configurato per consentire l'`sns:Publish` sull'Amazon di destinazione SNS topic, come mostrato nel seguente esempio di policy IAM.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1528133056761",
      "Action": [
        "sns:Publish"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:sns:region:account-id:topic-name"
      ]
    }
  ]
}
```

Questo connettore consente di sostituire dinamicamente l'argomento predefinito nel payload del messaggio di input. Se l'implementazione utilizza questa funzionalità, la policy IAM deve consentire l'`sns:Publish` autorizzazione su tutti gli argomenti di destinazione. Puoi concedere alle risorse un accesso granulare o condizionale (ad esempio, utilizzando uno schema di denominazione con il carattere jolly `*`).

Per il requisito del ruolo di gruppo, è necessario configurare il ruolo in modo da concedere le autorizzazioni necessarie e assicurarsi che il ruolo sia stato aggiunto al gruppo. Per ulteriori informazioni, consulta [the section called “Gestione del ruolo del gruppo \(console\)”](#) o [the section called “Gestire il ruolo del gruppo \(CLI\)”](#).

Parametri del connettore

Questo connettore fornisce i seguenti parametri:

Version 4

`DefaultSNSArn`

L'ARN dell'argomento SNS predefinito in cui pubblicare i messaggi. L'argomento di destinazione può essere ignorato con la proprietà `sns_topic_arn` del payload del messaggio di input.

Note

Il ruolo del gruppo deve permettere l'autorizzazione `sns:Publish` a tutti gli argomenti di destinazione. Per ulteriori informazioni, consulta [the section called “Requisiti”](#).

Nome visualizzato nella AWS IoT console: argomento SNS predefinito ARN

Obbligatorio: `true`

Tipo: `string`

Schema valido: `arn:aws:sns:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):([a-zA-Z0-9-_\+]*)$`

IsolationMode

Modalità di [containerizzazione](#) per questo connettore. L'impostazione predefinita è `GreengrassContainer`, il che significa che il connettore viene eseguito in un ambiente di runtime isolato all'interno del AWS IoT Greengrass contenitore.

Note

L'impostazione predefinita della containerizzazione per il gruppo non si applica ai connettori.

Nome visualizzato nella AWS IoT console: modalità di isolamento del contenitore

Obbligatorio: `false`

Tipo: `string`

Valori validi: `GreengrassContainer` o `NoContainer`

Schema valido: `^NoContainer$|^GreengrassContainer$`

Versions 1 - 3

DefaultSNSArn

L'ARN dell'argomento SNS predefinito in cui pubblicare i messaggi. L'argomento di destinazione può essere ignorato con la proprietà `sns_topic_arn` del payload del messaggio di input.

Note

Il ruolo del gruppo deve permettere l'autorizzazione `sns:Publish` a tutti gli argomenti di destinazione. Per ulteriori informazioni, consulta [the section called "Requisiti"](#).

Nome visualizzato nella AWS IoT console: argomento SNS predefinito ARN

Obbligatorio: `true`

Tipo: string

Schema valido: `arn:aws:sns:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):([a-zA-Z0-9-_]+)$`

Esempio di creazione di un connettore (AWS CLI)

Il seguente comando CLI crea un file `ConnectorDefinition` con una versione iniziale che contiene il connettore SNS.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-version '{
  "Connectors": [
    {
      "Id": "MySNSConnector",
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/SNS/versions/4",
      "Parameters": {
        "DefaultSNSArn": "arn:aws:sns:region:account-id:topic-name",
        "IsolationMode" : "GreengrassContainer"
      }
    }
  ]
}'
```

Nella AWS IoT Greengrass console, puoi aggiungere un connettore dalla pagina Connettori del gruppo. Per ulteriori informazioni, consulta [the section called “Nozioni di base sui connettori \(console\)”](#).

Dati di input

Questo connettore accetta informazioni sui messaggi SNS su un argomento MQTT, quindi pubblica il messaggio così com'è nell'argomento SNS di destinazione. I messaggi di input devono essere in formato JSON.

Filtro argomento in sottoscrizione

`sns/message`

Proprietà dei messaggi

`request`

Informazioni sul messaggio da inviare all'argomento SNS.

Obbligatorio: `true`

Tipo: `object` che include le seguenti proprietà:

`message`

Il contenuto del messaggio come stringa o in formato JSON. Per degli esempi, vedi [Input di esempio](#).

Per inviare JSON, la proprietà `message_structure` deve essere impostata su `json` e il messaggio deve essere un oggetto JSON codificato come stringa che contiene una chiave `default`.

Obbligatorio: `true`

Tipo: `string`

Schema valido: `.*`

`subject`

L'oggetto del messaggio.

Richiesto: `false`

Tipo: testo ASCII, fino a 100 caratteri. Deve iniziare con una lettera, un numero o un segno di punteggiatura. Non deve contenere interruzioni di riga né caratteri di controllo.

Modello valido: `.*`

`sns_topic_arn`

L'ARN dell'argomento SNS in cui pubblicare i messaggi. Se specificato, il connettore effettua la pubblicazione in questo argomento anziché in quello predefinito.

 Note

Il ruolo del gruppo deve permettere l'autorizzazione `sns:Publish` a qualsiasi argomento di destinazione. Per ulteriori informazioni, consulta [the section called "Requisiti"](#).

Richiesto: `false`

Tipo: `string`

Schema valido: `arn:aws:sns:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):([a-zA-Z0-9-_\])+)`

`message_structure`

La struttura del messaggio.

Obbligatorio: `false`. Questo deve essere specificato per inviare un messaggio JSON.

Tipo: `string`

Valori validi: `json`

`id`

Un ID arbitrario della richiesta. Questa proprietà viene utilizzata per associare una richiesta di input a una risposta di output. Quando specificato, la proprietà `id` nell'oggetto della risposta è impostata su questo valore. Se non utilizzi questa funzione, puoi omettere la proprietà oppure specificare una stringa vuota.

Richiesto: `false`

Tipo: `string`

Schema valido: `.*`

Limiti

Le dimensioni dei messaggi SNS non devono superare i 256 KB.

Input di esempio: messaggio in formato stringa

In questo esempio viene inviato un messaggio in formato stringa. Specifica la proprietà facoltativa `sns_topic_arn`, che sostituisce l'argomento di destinazione predefinito.

```
{
  "request": {
    "subject": "Message subject",
    "message": "Message data",
    "sns_topic_arn": "arn:aws:sns:region:account-id:topic2-name"
  },
  "id": "request123"
```

```
}
```

Input di esempio: messaggio JSON

Questo esempio invia un messaggio come oggetto JSON codificato come stringa che include la chiave default.

```
{
  "request": {
    "subject": "Message subject",
    "message": "{ \"default\": \"Message data\" }",
    "message_structure": "json"
  },
  "id": "request123"
}
```

Dati di output

Questo connettore pubblica le informazioni di stato come dati di output su un argomento MQTT.

Filtro argomento in sottoscrizione

sns/message/status

Output di esempio: Operazione riuscita

```
{
  "response": {
    "sns_message_id": "f80a81bc-f44c-56f2-a0f0-d5af6a727c8a",
    "status": "success"
  },
  "id": "request123"
}
```

Esempio di output: Errore

```
{
  "response" : {
    "error": "InvalidInputException",
    "error_message": "SNS Topic Arn is invalid",
    "status": "fail"
  },
}
```



```
"id": "request123"  
}
```

Esempio di utilizzo

Usa i seguenti passaggi di alto livello per configurare una funzione Lambda di esempio di Python 3.7 che puoi usare per provare il connettore.

Note

- Se usi altri runtime Python, puoi creare un collegamento simbolico da Python3.x a Python 3.7.
- Gli argomenti [Nozioni di base sui connettori \(console\)](#) e [Nozioni di base sui connettori \(CLI\)](#) contengono passaggi dettagliati che illustrano come configurare e distribuire un connettore Twilio Notifications di esempio.

1. Assicurarsi di soddisfare i [requisiti](#) per il connettore.

Per il requisito del ruolo di gruppo, è necessario configurare il ruolo in modo da concedere le autorizzazioni necessarie e assicurarsi che il ruolo sia stato aggiunto al gruppo. Per ulteriori informazioni, consulta [the section called “Gestione del ruolo del gruppo \(console\)”](#) o [the section called “Gestire il ruolo del gruppo \(CLI\)”](#).

2. Crea e pubblica una funzione Lambda che invia dati di input al connettore.

Salvare il [codice di esempio](#) come file PY. Scarica e decomprimi il [AWS IoT Greengrass Core SDK per Python](#). Quindi, crea un pacchetto zip che contiene il file PY e la cartella greengrasssdk a livello root. Questo pacchetto zip è il pacchetto di distribuzione in cui carichi. AWS Lambda

Dopo aver creato la funzione Python 3.7 Lambda, pubblica una versione della funzione e crea un alias.

3. Configurare il gruppo Greengrass.
 - a. Aggiungi la funzione Lambda tramite il relativo alias (consigliato). Configura il ciclo di vita Lambda come longevo (o nella "Pinned": true CLI).
 - b. Aggiungere il connettore e configurarne i relativi [parametri](#).

- c. Aggiungere sottoscrizioni che consentono al connettore di ricevere [i dati di input](#) e inviare [i dati di output](#) nei filtri degli argomenti supportati.
 - Imposta la funzione Lambda come origine, il connettore come destinazione e utilizza un filtro per argomenti di input supportato.
 - Imposta il connettore come origine, AWS IoT Core come destinazione e utilizza un filtro per l'argomento di output supportato. Utilizzi questo abbonamento per visualizzare i messaggi di stato nella AWS IoT console.
4. Distribuisci il gruppo.
5. Nella AWS IoT console, nella pagina Test, sottoscrivi l'argomento relativo ai dati di output per visualizzare i messaggi di stato dal connettore. La funzione Lambda di esempio è di lunga durata e inizia a inviare messaggi subito dopo l'implementazione del gruppo.

Al termine del test, puoi impostare il ciclo di vita Lambda su richiesta (o nella CLI) e "Pinned": `false` distribuire il gruppo. Ciò impedisce alla funzione di inviare messaggi.

Esempio

L'esempio seguente della funzione Lambda invia un messaggio di input al connettore.

```
import greengrasssdk
import time
import json

iot_client = greengrasssdk.client('iot-data')
send_topic = 'sns/message'

def create_request_with_all_fields():
    return {
        "request": {
            "message": "Message from SNS Connector Test"
        },
        "id" : "req_123"
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
        payload=json.dumps(messageToPublish))
```

```
publish_basic_message()  
  
def lambda_handler(event, context):  
    return
```

Licenze

Il connettore SNS include i seguenti software/licenze di terze parti:

- [AWS SDK per Python \(Boto3\)](#)/Apache License 2.0
- [botocore](#)/Apache License 2.0
- [dateutil](#)/PSF License
- [docutils](#)/BSD License, GNU General Public License (GPL), Python Software Foundation License, Public Domain
- [jmespath](#)/MIT License
- [s3transfer](#)/Apache License 2.0
- [urllib3](#)/MIT License

Questo connettore è rilasciato ai sensi del contratto di [licenza del software Greengrass Core](#).

Changelog

La tabella seguente descrive le modifiche apportate a ciascuna versione del connettore.

Versione	Modifiche
4	Aggiunto il parametro <code>IsolationMode</code> per configurare la modalità di containerizzazione per il connettore.
3	È stato aggiornato il runtime Lambda a Python 3.7, che modifica i requisiti di runtime.
2	Correggere per ridurre l'eccessiva registrazione di log.
1	Versione iniziale.

Un gruppo Greengrass può contenere una sola versione del connettore alla volta. Per informazioni sull'aggiornamento di una versione del connettore, consulta [the section called “Aggiornamento delle versioni dei connettori”](#).

Consulta anche

- [Integrazione con servizi e protocolli tramite i connettori](#)
- [the section called “Nozioni di base sui connettori \(console\)”](#)
- [the section called “Nozioni di base sui connettori \(CLI\)”](#)
- [Operazione di pubblicazione](#) nella documentazione Boto 3
- [Cos'è Amazon Simple Notification Service?](#) nella Guida per gli sviluppatori di Amazon Simple Notification Service

Connettore di integrazione Splunk

Warning

Questo connettore è entrato nella fase di vita prolungata e AWS IoT Greengrass non rilascerà aggiornamenti che forniscano funzionalità, miglioramenti alle funzionalità esistenti, patch di sicurezza o correzioni di bug. Per ulteriori informazioni, consulta [AWS IoT Greengrass Version 1 politica di manutenzione](#).

Il [connettore](#) Splunk Integration pubblica i dati dai dispositivi Greengrass su Splunk. In questo modo potrai utilizzare Splunk per monitorare e analizzare l'ambiente core di Greengrass e agire sugli eventi locali. Il connettore si integra con HTTP Event Collector (HEC). Per ulteriori informazioni, consulta [Introduzione a Splunk HTTP Event Collector](#) nella documentazione di Splunk.

Questo connettore riceve dati di eventi e di logging in un argomento MQTT e pubblica i dati così come sono nell'API Splunk.

Puoi utilizzare questo connettore per supportare scenari industriali, ad esempio:

- Gli operatori possono utilizzare i dati periodici di attuatori e sensori (ad esempio, letture di temperatura, pressione e acqua) per avviare allarmi quando i valori superano determinate soglie.
- Gli sviluppatori utilizzano i dati raccolti dai macchinari industriali per creare modelli ML in grado di monitorare le apparecchiature in vista di potenziali problemi.

Questo connettore ha le seguenti versioni.

Versione	ARN
4	<code>arn:aws:greengrass: <i>region</i>::/connectors/SplunkIntegration/versions/4</code>
3	<code>arn:aws:greengrass: <i>region</i>::/connectors/SplunkIntegration/versions/3</code>
2	<code>arn:aws:greengrass: <i>region</i>::/connectors/SplunkIntegration/versions/2</code>
1	<code>arn:aws:greengrass: <i>region</i>::/connectors/SplunkIntegration/versions/1</code>

Per informazioni sulle modifiche di ogni versione, consulta [Changelog](#).

Requisiti

Questo connettore presenta i seguenti requisiti:

Version 3 - 4

- AWS IoT Greengrass Software principale v1.9.3 o versione successiva. AWS IoT Greengrass [deve essere configurato per supportare i segreti locali, come descritto in Secrets Requirements](#).



Note

Questo requisito include l'autorizzazione all'accesso ai segreti di Secrets Manager. Se utilizzi il ruolo di servizio Greengrass predefinito, Greengrass è autorizzato a ottenere i valori dei segreti con nomi che iniziano con greengrass-.

- [Python](#) versione 3.7 o 3.8 installata sul dispositivo principale e aggiunta alla variabile di ambiente PATH.

 Note

Per usare Python 3.8, esegui il seguente comando per creare un collegamento simbolico dalla cartella di installazione predefinita di Python 3.7 ai binari Python 3.8 installati.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Questo configura il dispositivo in modo che soddisfi il requisito Python per AWS IoT Greengrass.

- La funzionalità HTTP Event Collector deve essere abilitata in Splunk. Per ulteriori informazioni, consulta [Configurazione e utilizzo di HTTP Event Collector in Splunk Web](#) nella documentazione di Splunk.
- Un tipo di testo segreto AWS Secrets Manager che memorizza il token Splunk HTTP Event Collector. Per maggiori informazioni, consulta [About event collector tokens](#) nella documentazione di Splunk e [Creazione di un segreto di base](#) nella Guida per l'utente AWS Secrets Manager

 Note

Per creare il segreto nella console Secrets Manager, inserisci il tuo token nella scheda Plaintext. Non includere virgolette o altri tipi di formattazione. Nell'API, specifica il token come valore per la `SecretString` proprietà.

- Una risorsa segreta del gruppo Greengrass che fa riferimento al segreto di Secrets Manager. Per ulteriori informazioni, consulta [Distribuzione dei segreti nel core](#).

Versions 1 - 2

- AWS IoT Greengrass Software principale v1.7 o successivo. AWS IoT Greengrass deve essere configurato per supportare i segreti locali, come descritto in [Secrets Requirements](#).

 Note

Questo requisito include l'autorizzazione all'accesso ai segreti di Secrets Manager. Se utilizzi il ruolo di servizio Greengrass predefinito, Greengrass è autorizzato a ottenere i valori dei segreti con nomi che iniziano con greengrass-.

- [Python](#) versione 2.7 installato sul dispositivo principale e aggiunto alla variabile di ambiente PATH.
- La funzionalità HTTP Event Collector deve essere abilitata in Splunk. Per ulteriori informazioni, consulta [Configurazione e utilizzo di HTTP Event Collector in Splunk Web](#) nella documentazione di Splunk.
- Un tipo di testo segreto AWS Secrets Manager che memorizza il token Splunk HTTP Event Collector. Per maggiori informazioni, consulta [About event collector tokens](#) nella documentazione di Splunk e [Creazione di un segreto di base](#) nella Guida per l'utente AWS Secrets Manager

 Note

Per creare il segreto nella console Secrets Manager, inserisci il tuo token nella scheda Plaintext. Non includere virgolette o altri tipi di formattazione. Nell'API, specifica il token come valore per la `SecretString` proprietà.

- Una risorsa segreta del gruppo Greengrass che fa riferimento al segreto di Secrets Manager. Per ulteriori informazioni, consulta [Distribuzione dei segreti nel core](#).

Parametri del connettore

Questo connettore fornisce i seguenti parametri:

Version 4

`SplunkEndpoint`

L'endpoint dell'istanza database Splunk. Questo valore deve contenere il protocollo, il nome host e la porta.

Nome visualizzato nella AWS IoT console: Splunk endpoint

Obbligatorio: true

Tipo: string

Schema valido: `^(http:\\\\|https:\\\\)?[a-z0-9]+([-\\.]{1}[a-z0-9]+)*.[a-z]{2,5}(:[0-9]{1,5})?(\\/.*)?$`

MemorySize

La quantità di memoria (in KB) da allocare al connettore.

Nome visualizzato nella AWS IoT console: dimensione della memoria

Obbligatorio: true

Tipo: string

Schema valido: `^[0-9]+$`

SplunkQueueSize

Il numero massimo di voci da salvare in memoria prima che vengono inviate o eliminate. Quando si raggiunge questo limite, le voci meno recenti in coda vengono sostituite da quelle più recenti. Questo limite si applica in genere in assenza di una connessione a Internet.

Nome visualizzato nella AWS IoT console: numero massimo di elementi da conservare

Obbligatorio: true

Tipo: string

Schema valido: `^[0-9]+$`

SplunkFlushIntervalSeconds

L'intervallo di tempo (in secondi) per la pubblicazione dei dati ricevuti in Splunk HEC. Il valore massimo è 900. Per configurare il connettore in modo da pubblicare gli elementi man mano che vengono ricevuti (senza raggruppamento), specificare 0.

Nome visualizzato nella AWS IoT console: intervallo di pubblicazione Splunk

Obbligatorio: true

Tipo: string

Schema valido: `[0-9] | [1-9]\\d | [1-9]\\d\\d | 900`

SplunkTokenSecretArn

Il segreto AWS Secrets Manager che contiene il token Splunk. Deve essere un segreto in formato testo.

Nome visualizzato nella AWS IoT console: ARN del token di autenticazione Splunk secret

Obbligatorio: true

Tipo: string

Schema valido: `arn:aws:secretsmanager:[a-z]{2}-[a-z]+-\d{1}:\d{12}?:secret:[a-zA-Z0-9-_\d]{1,64}`

SplunkTokenSecretArn-ResourceId

La risorsa segreta nel gruppo Greengrass che fa riferimento al segreto Splunk.

Nome visualizzato nella AWS IoT console: risorsa del token di autenticazione Splunk

Obbligatorio: true

Tipo: string

Schema valido: `.*`

SplunkCustomCALocation

Il percorso del file dell'autorità di certificazione (CA) personalizzata per Splunk (ad esempio `/etc/ssl/certs/splunk.crt`).

Nome visualizzato nella AWS IoT console: posizione personalizzata dell'autorità di certificazione Splunk

Obbligatorio: false

Tipo: string

Schema valido: `^[a-zA-Z0-9_/.]*`

IsolationMode

Modalità di [containerizzazione](#) per questo connettore. L'impostazione predefinita è `GreengrassContainer`, il che significa che il connettore viene eseguito in un ambiente di runtime isolato all'interno del AWS IoT Greengrass contenitore.

 Note

L'impostazione predefinita della containerizzazione per il gruppo non si applica ai connettori.

Nome visualizzato nella AWS IoT console: modalità di isolamento del contenitore

Obbligatorio: `false`

Tipo: `string`

Valori validi: `GreengrassContainer` o `NoContainer`

Schema valido: `^NoContainer$|^GreengrassContainer$`

Version 1 - 3

SplunkEndpoint

L'endpoint dell'istanza database Splunk. Questo valore deve contenere il protocollo, il nome host e la porta.

Nome visualizzato nella AWS IoT console: Splunk endpoint

Obbligatorio: `true`

Tipo: `string`

Schema valido: `^(http:\\\\|https:\\\\)?[a-z0-9]+([-.]{1}[a-z0-9]+)*.[a-z]{2,5}(:[0-9]{1,5})?(\\.|*)?$`

MemorySize

La quantità di memoria (in KB) da allocare al connettore.

Nome visualizzato nella AWS IoT console: dimensione della memoria

Obbligatorio: `true`

Tipo: `string`

Schema valido: `^[0-9]+$`

SplunkQueueSize

Il numero massimo di voci da salvare in memoria prima che vengono inviate o eliminate. Quando si raggiunge questo limite, le voci meno recenti in coda vengono sostituite da quelle più recenti. Questo limite si applica in genere in assenza di una connessione a Internet.

Nome visualizzato nella AWS IoT console: numero massimo di elementi da conservare

Obbligatorio: true

Tipo: string

Schema valido: `^[0-9]+$`

SplunkFlushIntervalSeconds

L'intervallo di tempo (in secondi) per la pubblicazione dei dati ricevuti in Splunk HEC. Il valore massimo è 900. Per configurare il connettore in modo da pubblicare gli elementi man mano che vengono ricevuti (senza raggruppamento), specificare 0.

Nome visualizzato nella AWS IoT console: Intervallo di pubblicazione di Splunk

Obbligatorio: true

Tipo: string

Schema valido: `[0-9] | [1-9]\d | [1-9]\d\d | 900`

SplunkTokenSecretArn

Il segreto AWS Secrets Manager che contiene il token Splunk. Deve essere un segreto in formato testo.

Nome visualizzato nella AWS IoT console: ARN del token di autenticazione Splunk secret

Obbligatorio: true

Tipo: string

Schema valido: `arn:aws:secretsmanager:[a-z]{2}-[a-z]+-\d{1}:\d{12}?:secret:[a-zA-Z0-9-_\d]{32}`

SplunkTokenSecretArn-ResourceId

La risorsa segreta nel gruppo Greengrass che fa riferimento al segreto Splunk.

Nome visualizzato nella AWS IoT console: risorsa del token di autenticazione Splunk

Obbligatorio: true

Tipo: string

Schema valido: .+

SplunkCustomCALocation

Il percorso del file dell'autorità di certificazione (CA) personalizzata per Splunk (ad esempio /etc/ssl/certs/splunk.crt).

Nome visualizzato nella AWS IoT console: posizione personalizzata dell'autorità di certificazione Splunk

Obbligatorio: false

Tipo: string

Schema valido: ^\$|/.*

Esempio di creazione di un connettore (AWS CLI)

Il seguente comando CLI crea una `ConnectorDefinition` versione iniziale che contiene il connettore Splunk Integration.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-  
version '{  
  "Connectors": [  
    {  
      "Id": "MySplunkIntegrationConnector",  
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/SplunkIntegration/  
versions/4",  
      "Parameters": {  
        "SplunkEndpoint": "https://myinstance.cloud.splunk.com:8088",  
        "MemorySize": 200000,  
        "SplunkQueueSize": 10000,  
        "SplunkFlushIntervalSeconds": 5,  
        "SplunkTokenSecretArn": "arn:aws:secretsmanager:region:account-  
id:secret:greengrass-secret-hash",  
        "SplunkTokenSecretArn-ResourceId": "MySplunkResource",  
        "IsolationMode" : "GreengrassContainer"
```

```
}  
  }  
  ]  
}'
```

Note

La funzione Lambda in questo connettore ha un ciclo di vita di [lunga durata](#).

Nella AWS IoT Greengrass console, puoi aggiungere un connettore dalla pagina Connettori del gruppo. Per ulteriori informazioni, consulta [the section called “Nozioni di base sui connettori \(console\)”](#).

Dati di input

Questo connettore accetta dati di registrazione ed eventi su un argomento MQTT e pubblica i dati ricevuti così come sono nell'API Splunk. I messaggi di input devono essere in formato JSON.

Filtro argomento in sottoscrizione

splunk/logs/put

Proprietà dei messaggi

request

I dati dell'evento da inviare all'API Splunk. Gli eventi devono soddisfare le specifiche dell'API [services/collector](#).

Obbligatorio: true

Tipo: object. È obbligatoria solo la proprietà event.

id

Un ID arbitrario della richiesta. Questa proprietà viene utilizzata per associare una richiesta di input a uno stato di output.

Richiesto: false

Tipo: string

Limiti

Tutti i limiti imposti dall'API Splunk vengono applicati quando si utilizza questo connettore. Per ulteriori informazioni, consulta [services/collector](#).

Input di esempio

```
{
  "request": {
    "event": "some event",
    "fields": {
      "severity": "INFO",
      "category": [
        "value1",
        "value2"
      ]
    }
  },
  "id": "request123"
}
```

Dati di output

Questo connettore pubblica i dati di output in due argomenti:

- Informazioni di stato nell'argomento `splunk/logs/put/status`.
- Gli errori nell'argomento `splunk/logs/put/error`.

Filtro di argomenti: `splunk/logs/put/status`

Utilizza questo argomento per ascoltare lo stato delle richieste. Ogni volta che il connettore invia un batch di dati ricevuti all'API Splunk, pubblica un elenco IDs delle richieste riuscite e non riuscite.

Output di esempio

```
{
  "response": {
    "succeeded": [
      "request123",
      ...
    ],
  },
}
```

```
        "failed": [  
            "request789",  
            ...  
        ]  
    }  
}
```

Filtro di argomenti: `splunk/logs/put/error`

Utilizza questo argomento per ascoltare gli errori del connettore. La proprietà `error_message` che descrive l'errore o il timeout rilevato durante l'elaborazione della richiesta.

Output di esempio

```
{  
  "response": {  
    "error": "UnauthorizedException",  
    "error_message": "invalid splunk token",  
    "status": "fail"  
  }  
}
```

Note

Se il connettore rileva un errore riutilizzabile (ad esempio errori di connessione), riprova la pubblicazione nel batch successivo.

Esempio di utilizzo

Usa i seguenti passaggi di alto livello per configurare una funzione Lambda di esempio di Python 3.7 che puoi usare per provare il connettore.

Note

- Se usi altri runtime Python, puoi creare un collegamento simbolico da Python3.x a Python 3.7.
- Gli argomenti [Nozioni di base sui connettori \(console\)](#) e [Nozioni di base sui connettori \(CLI\)](#) contengono passaggi dettagliati che illustrano come configurare e distribuire un connettore Twilio Notifications di esempio.

1. Assicurarsi di soddisfare i [requisiti](#) per il connettore.
2. Crea e pubblica una funzione Lambda che invia dati di input al connettore.

Salvare il [codice di esempio](#) come file PY. Scarica e decomprimi il [AWS IoT Greengrass Core SDK per Python](#). Quindi, crea un pacchetto zip che contiene il file PY e la cartella greengrasssdk a livello root. Questo pacchetto zip è il pacchetto di distribuzione in cui carichi AWS Lambda

Dopo aver creato la funzione Python 3.7 Lambda, pubblica una versione della funzione e crea un alias.

3. Configurare il gruppo Greengrass.
 - a. Aggiungi la funzione Lambda tramite il relativo alias (consigliato). Configura il ciclo di vita Lambda come longevo (o nella "Pinned": true CLI).
 - b. Aggiungi la risorsa segreta richiesta e concedi l'accesso in lettura alla funzione Lambda.
 - c. Aggiungere il connettore e configurarne i relativi [parametri](#).
 - d. Aggiungere sottoscrizioni che consentono al connettore di ricevere [i dati di input](#) e inviare [i dati di output](#) nei filtri degli argomenti supportati.
 - Imposta la funzione Lambda come origine, il connettore come destinazione e utilizza un filtro per argomenti di input supportato.
 - Imposta il connettore come origine, AWS IoT Core come destinazione e utilizza un filtro per l'argomento di output supportato. Utilizzi questo abbonamento per visualizzare i messaggi di stato nella AWS IoT console.
4. Distribuisci il gruppo.
5. Nella AWS IoT console, nella pagina Test, sottoscrivi l'argomento relativo ai dati di output per visualizzare i messaggi di stato dal connettore. La funzione Lambda di esempio è di lunga durata e inizia a inviare messaggi subito dopo l'implementazione del gruppo.

Al termine del test, puoi impostare il ciclo di vita Lambda su richiesta (o nella CLI) e "Pinned": false distribuire il gruppo. Ciò impedisce alla funzione di inviare messaggi.

Esempio

L'esempio seguente della funzione Lambda invia un messaggio di input al connettore.

```
import greengrasssdk
```



```
import time
import json

iot_client = greengrasssdk.client('iot-data')
send_topic = 'splunk/logs/put'

def create_request_with_all_fields():
    return {
        "request": {
            "event": "Access log test message."
        },
        "id" : "req_123"
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
        payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

Licenze

Questo connettore è rilasciato ai sensi del contratto di [licenza del software Greengrass Core](#).

Changelog

La tabella seguente descrive le modifiche apportate a ciascuna versione del connettore.

Versione	Modifiche
4	Aggiunto il parametro <code>IsolationMode</code> per configurare la modalità di containerizzazione per il connettore.
3	È stato aggiornato il runtime Lambda a Python 3.7, che modifica i requisiti di runtime.

Versione	Modifiche
2	Correggere per ridurre l'eccessiva registrazione di log.
1	Versione iniziale.

Un gruppo Greengrass può contenere una sola versione del connettore alla volta. Per informazioni sull'aggiornamento di una versione del connettore, consulta [the section called “Aggiornamento delle versioni dei connettori”](#).

Consulta anche

- [Integrazione con servizi e protocolli tramite i connettori](#)
- [the section called “Nozioni di base sui connettori \(console\)”](#)
- [the section called “Nozioni di base sui connettori \(CLI\)”](#)

Connettore di notifica Twilio

Warning

Questo connettore è entrato nella fase di vita prolungata e AWS IoT Greengrass non rilascerà aggiornamenti che forniscano funzionalità, miglioramenti alle funzionalità esistenti, patch di sicurezza o correzioni di bug. Per ulteriori informazioni, consulta [AWS IoT Greengrass Version 1 politica di manutenzione](#).

Il [connettore](#) Twilio Notifications effettua chiamate telefoniche automatiche o invia messaggi di testo tramite Twilio. Puoi utilizzare questo connettore per inviare notifiche in risposta a eventi nel gruppo Greengrass. Per le telefonate, il connettore è in grado di inoltrare un messaggio vocale al destinatario.

Questo connettore riceve le informazioni sui messaggi Twilio in un argomento MQTT, quindi attiva una notifica Twilio.

 Note

Per un tutorial che mostra come utilizzare il connettore Twilio Notifications, consulta o. [the section called “Nozioni di base sui connettori \(console\)”](#) [the section called “Nozioni di base sui connettori \(CLI\)”](#)

Questo connettore ha le seguenti versioni.

Versione	ARN
5	<code>arn:aws:greengrass: <i>region</i>::/connectors/TwilioNotifications/versions/5</code>
4	<code>arn:aws:greengrass: <i>region</i>::/connectors/TwilioNotifications/versions/4</code>
3	<code>arn:aws:greengrass: <i>region</i>::/connectors/TwilioNotifications/versions/3</code>
2	<code>arn:aws:greengrass: <i>region</i>::/connectors/TwilioNotifications/versions/2</code>
1	<code>arn:aws:greengrass: <i>region</i>::/connectors/TwilioNotifications/versions/1</code>

Per informazioni sulle modifiche di ogni versione, consulta [Changelog](#).

Requisiti

Questo connettore presenta i seguenti requisiti:

Version 4 - 5

- AWS IoT Greengrass Software principale v1.9.3 o successivo. AWS IoT Greengrass [deve essere configurato per supportare i segreti locali, come descritto in Secrets Requirements](#).

 Note

Questo requisito include l'autorizzazione all'accesso ai segreti di Secrets Manager. Se utilizzi il ruolo di servizio Greengrass predefinito, Greengrass è autorizzato a ottenere i valori dei segreti con nomi che iniziano con greengrass-.

- [Python](#) versione 3.7 o 3.8 installata sul dispositivo principale e aggiunta alla variabile di ambiente PATH.

 Note

Per usare Python 3.8, esegui il seguente comando per creare un collegamento simbolico dalla cartella di installazione predefinita di Python 3.7 ai binari Python 3.8 installati.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Questo configura il dispositivo in modo che soddisfi il requisito Python per AWS IoT Greengrass.

- SID dell'account Twilio, token di autorizzazione e numero di telefono abilitato per Twilio. Dopo avere creato un progetto Twilio, questi valori sono disponibili nel pannello di controllo del progetto.

 Note

Puoi utilizzare un account di prova Twilio. Se utilizzi un account di prova, devi aggiungere i numeri di telefono dei destinatari diversi da Twilio a un elenco di numeri di telefono verificati. Per ulteriori informazioni, vedi [Come lavorare con il tuo account di prova Twilio gratuito](#).

- Un tipo di testo segreto AWS Secrets Manager che memorizza il token di autenticazione Twilio. Per ulteriori informazioni, consulta [Creazione di un segreto di base nella Guida](#) per l'AWS Secrets Manager utente.

 Note

Per creare il segreto nella console Secrets Manager, inserisci il tuo token nella scheda Plaintext. Non includere virgolette o altri tipi di formattazione. Nell'API, specifica il token come valore per la `SecretString` proprietà.

- Una risorsa segreta del gruppo Greengrass che fa riferimento al segreto di Secrets Manager. Per ulteriori informazioni, consulta [Distribuzione dei segreti nel core](#).

Versions 1 - 3

- AWS IoT Greengrass Software principale v1.7 o successivo. AWS IoT Greengrass deve essere configurato per supportare i segreti locali, come descritto in [Secrets Requirements](#).

 Note

Questo requisito include l'autorizzazione all'accesso ai segreti di Secrets Manager. Se utilizzi il ruolo di servizio Greengrass predefinito, Greengrass è autorizzato a ottenere i valori dei segreti con nomi che iniziano con `greengrass-`.

- [Python](#) versione 2.7 installato sul dispositivo principale e aggiunto alla variabile di ambiente `PATH`.
- SID dell'account Twilio, token di autorizzazione e numero di telefono abilitato per Twilio. Dopo avere creato un progetto Twilio, questi valori sono disponibili nel pannello di controllo del progetto.

 Note

Puoi utilizzare un account di prova Twilio. Se utilizzi un account di prova, devi aggiungere numeri di telefono dei destinatari diversi da Twilio a un elenco di numeri di telefono verificati. Per ulteriori informazioni, vedi [Come lavorare con il tuo account di prova Twilio gratuito](#).

- Un tipo di testo segreto AWS Secrets Manager che memorizza il token di autenticazione Twilio. Per ulteriori informazioni, consulta [Creazione di un segreto di base nella Guida](#) per l'AWS Secrets Manager utente.

Note

Per creare il segreto nella console Secrets Manager, inserisci il tuo token nella scheda Plaintext. Non includere virgolette o altri tipi di formattazione. Nell'API, specifica il token come valore per la `SecretString` proprietà.

- Una risorsa segreta del gruppo Greengrass che fa riferimento al segreto di Secrets Manager. Per ulteriori informazioni, consulta [Distribuzione dei segreti nel core](#).

Parametri del connettore

Questo connettore fornisce i seguenti parametri.

Version 5

`TWILIO_ACCOUNT_SID`

Il SID dell'account Twilio utilizzato per chiamare l'API Twilio.

Nome visualizzato nella AWS IoT console: SID dell'account Twilio

Obbligatorio: `true`

Tipo: `string`

Modello valido: `.+`

`TwilioAuthTokenSecretArn`

L'ARN del segreto Secrets Manager che memorizza il token di autenticazione Twilio.

Note

Viene utilizzato per accedere al valore del segreto locale nel core.

Nome visualizzato nella AWS IoT console: ARN del token di autenticazione segreto di Twilio

Obbligatorio: true

Tipo: string

Modello valido: `arn:aws:secretsmanager:[a-z0-9\-\+]:[0-9]{12}:secret:([a-zA-Z0-9\\]+/)*[a-zA-Z0-9/_+=,.\@-]+-[a-zA-Z0-9]+`

TwilioAuthTokenSecretArn-ResourceId

L'ID della risorsa segreta del gruppo Greengrass che fa riferimento al segreto del token di autorizzazione Twilio.

Nome visualizzato nella AWS IoT console: risorsa del token di autenticazione Twilio

Obbligatorio: true

Tipo: string

Modello valido: `.*`

DefaultFromPhoneNumber

Il numero di telefono predefinito abilitato per Twilio che Twilio utilizza per l'invio di messaggi. Twilio utilizza questo numero per iniziare il messaggio di testo o la chiamata.

- Se non configuri un numero di telefono predefinito, dovrai specificare un numero di telefono nella proprietà `from_number` nel corpo del messaggio di input.
- Se configuri un numero di telefono predefinito, potrai ignorarlo specificando la proprietà `from_number` nel corpo del messaggio di input.

Nome visualizzato nella AWS IoT console: predefinito dal numero di telefono

Obbligatorio: false

Tipo: string

Modello valido: `^\$|\+[0-9]+`

IsolationMode

Modalità di [containerizzazione](#) per questo connettore. L'impostazione predefinita è `GreengrassContainer`, il che significa che il connettore viene eseguito in un ambiente di runtime isolato all'interno del AWS IoT Greengrass contenitore.

 Note

L'impostazione predefinita della containerizzazione per il gruppo non si applica ai connettori.

Nome visualizzato nella AWS IoT console: modalità di isolamento del contenitore

Obbligatorio: `false`

Tipo: `string`

Valori validi: `GreengrassContainer` o `NoContainer`

Modello valido: `^NoContainer$|^GreengrassContainer$`

Version 1 - 4

`TWILIO_ACCOUNT_SID`

Il SID dell'account Twilio utilizzato per chiamare l'API Twilio.

Nome visualizzato nella AWS IoT console: SID dell'account Twilio

Obbligatorio: `true`

Tipo: `string`

Modello valido: `.+`

`TwilioAuthTokenSecretArn`

L'ARN del segreto Secrets Manager che memorizza il token di autenticazione Twilio.

 Note

Viene utilizzato per accedere al valore del segreto locale nel core.

Nome visualizzato nella AWS IoT console: ARN del token di autenticazione segreto di Twilio

Obbligatorio: `true`

Tipo: `string`

Modello valido: `arn:aws:secretsmanager:[a-z0-9\-\+]:[0-9]{12}:secret:([a-zA-Z0-9\-\+\/]*[a-zA-Z0-9/_+=,.\@-]+-[a-zA-Z0-9]+`

`TwilioAuthTokenSecretArn-ResourceId`

L'ID della risorsa segreta del gruppo Greengrass che fa riferimento al segreto del token di autorizzazione Twilio.

Nome visualizzato nella AWS IoT console: risorsa del token di autenticazione Twilio

Obbligatorio: `true`

Tipo: `string`

Modello valido: `.\+`

`DefaultFromPhoneNumber`

Il numero di telefono predefinito abilitato per Twilio che Twilio utilizza per l'invio di messaggi. Twilio utilizza questo numero per iniziare il messaggio di testo o la chiamata.

- Se non configuri un numero di telefono predefinito, dovrai specificare un numero di telefono nella proprietà `from_number` nel corpo del messaggio di input.
- Se configuri un numero di telefono predefinito, potrai ignorarlo specificando la proprietà `from_number` nel corpo del messaggio di input.

Nome visualizzato nella AWS IoT console: predefinito dal numero di telefono

Obbligatorio: `false`

Tipo: `string`

Modello valido: `^\$|\+[0-9]+`

Esempio di creazione di un connettore (AWS CLI)

Il seguente comando CLI di esempio crea un comando `ConnectorDefinition` con una versione iniziale che contiene il connettore Twilio Notifications.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
  "Connectors": [
    {
```

```

    "Id": "MyTwilioNotificationsConnector",
    "ConnectorArn": "arn:aws:greengrass:region::/connectors/
TwilioNotifications/versions/5",
    "Parameters": {
        "TWILIO_ACCOUNT_SID": "abcd12345xyz",
        "TwilioAuthTokenSecretArn": "arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret-hash",
        "TwilioAuthTokenSecretArn-ResourceId": "MyTwilioSecret",
        "DefaultFromPhoneNumber": "+19999999999",
        "IsolationMode" : "GreengrassContainer"
    }
}
]
}'

```

Per i tutorial che mostrano come aggiungere il connettore Twilio Notifications a un gruppo, consulta e. [the section called “Nozioni di base sui connettori \(CLI\)”](#) [the section called “Nozioni di base sui connettori \(console\)”](#)

Dati di input

Questo connettore accetta informazioni sui messaggi Twilio su due argomenti MQTT. I messaggi di input devono essere in formato JSON.

- Informazioni sui messaggi di testo nell'argomento `twilio/txt`.
- Informazioni sui messaggi vocali nell'argomento `twilio/call`.

Note

Il payload del messaggio di input può includere un messaggio di testo (`message`) o un messaggio vocale (`voice_message_location`), ma non entrambi.

Filtro di argomenti: **`twilio/txt`**

Proprietà dei messaggi

`request`

Informazioni sulla notifica Twilio.

Obbligatorio: `true`

Tipo: object che include le seguenti proprietà:

`recipient`

Il destinatario del messaggio. È supportato solo un destinatario.

Richiesto: `true`

Tipo: object che include le seguenti proprietà:

`name`

Il nome del destinatario.

Obbligatorio: `true`

Tipo: `string`

Modello valido: `.*`

`phone_number`

Il numero di telefono del destinatario.

Richiesto: `true`

Tipo: `string`

Modello valido: `\+[1-9]+`

`message`

Il contenuto del messaggio di testo. In questo argomento sono supportati solo i messaggi di testo. Per i messaggi vocali, utilizzare `twilio/call`.

Richiesto: `true`

Tipo: `string`

Modello valido: `.+`

`from_number`

Il numero di telefono del mittente. Twilio utilizza questo numero di telefono per iniziare il messaggio. Questa proprietà è obbligatoria se il parametro `DefaultFromPhoneNumber` non è configurato. Se `DefaultFromPhoneNumber` è stato configurato, puoi utilizzare questa proprietà per ignorare quello predefinito.

Richiesto: false

Tipo: string

Modello valido: \+[1-9]+

retries

Il numero di tentativi. Il valore predefinito è 0.

Richiesto: false

Tipo: integer

id

Un ID arbitrario della richiesta. Questa proprietà viene utilizzata per associare una richiesta di input a una risposta di output.

Richiesto: true

Tipo: string

Modello valido: .+

Input di esempio

```
{
  "request": {
    "recipient": {
      "name": "Darla",
      "phone_number": "+12345000000",
      "message": "Hello from the edge"
    },
    "from_number": "+19999999999",
    "retries": 3
  },
  "id": "request123"
}
```

Filtro di argomenti: **twilio/call**

Proprietà dei messaggi

request

Informazioni sulla notifica Twilio.

Richiesto: true

Tipo: object che include le seguenti proprietà:

`recipient`

Il destinatario del messaggio. È supportato solo un destinatario.

Richiesto: true

Tipo: object che include le seguenti proprietà:

`name`

Il nome del destinatario.

Obbligatorio: true

Tipo: string

Modello valido: .+

`phone_number`

Il numero di telefono del destinatario.

Richiesto: true

Tipo: string

Modello valido: \+[1-9]+

`voice_message_location`

L'URL del contenuto audio del messaggio vocale. Deve essere in formato TwiML. In questo argomento sono supportati solo i messaggi vocali. Per i messaggi di testo, utilizzare `twilio/txt`.

Richiesto: true

Tipo: string

Modello valido: .+

`from_number`

Il numero di telefono del mittente. Twilio utilizza questo numero di telefono per iniziare il messaggio. Questa proprietà è obbligatoria se il parametro `DefaultFromPhoneNumber` non è configurato. Se `DefaultFromPhoneNumber` è stato configurato, puoi utilizzare questa proprietà per ignorare quello predefinito.

Richiesto: `false`

Tipo: `string`

Modello valido: `\+[1-9]+`

`retries`

Il numero di tentativi. Il valore predefinito è 0.

Richiesto: `false`

Tipo: `integer`

`id`

Un ID arbitrario della richiesta. Questa proprietà viene utilizzata per associare una richiesta di input a una risposta di output.

Richiesto: `true`

Tipo: `string`

Modello valido: `.+`

Input di esempio

```
{
  "request": {
    "recipient": {
      "name": "Darla",
      "phone_number": "+12345000000",
      "voice_message_location": "https://some-public-TwiML"
    },
    "from_number": "+19999999999",
    "retries": 3
  },
  "id": "request123"
}
```

Dati di output

Questo connettore pubblica le informazioni di stato come dati di output su un argomento MQTT.

Filtro argomento in sottoscrizione

`twilio/message/status`

Output di esempio: Operazione riuscita

```
{
  "response": {
    "status": "success",
    "payload": {
      "from_number": "+19999999999",
      "messages": {
        "message_status": "queued",
        "to_number": "+12345000000",
        "name": "Darla"
      }
    }
  },
  "id": "request123"
}
```

Esempio di output: Errore

```
{
  "response": {
    "status": "fail",
    "error_message": "Recipient name cannot be None",
    "error": "InvalidParameter",
    "payload": None
  },
  "id": "request123"
}
```

La proprietà `payload` nell'output è la risposta dall'API Twilio al momento dell'invio del messaggio. Se il connettore rileva che i dati di input non sono validi (ad esempio, non è specificato un campo di input obbligatorio), il connettore restituisce un errore e imposta il valore su `None`. Di seguito vengono riportati payload di esempio:

```
{
  'from_number': '+19999999999',
  'messages': {
    'name': 'Darla',
    'to_number': '+12345000000',
    'message_status': 'undelivered'
  }
}
```

```
{
  'from_number': '+19999999999',
  'messages': {
    'name': 'Darla',
    'to_number': '+12345000000',
    'message_status': 'queued'
  }
}
```

Esempio di utilizzo

Usa i seguenti passaggi di alto livello per configurare una funzione Lambda di esempio di Python 3.7 che puoi usare per provare il connettore.

Note

[the section called “Nozioni di base sui connettori \(CLI\)”](#) Gli argomenti [the section called “Nozioni di base sui connettori \(console\)”](#) e contengono end-to-end passaggi che mostrano come configurare, distribuire e testare il connettore Twilio Notifications.

1. Assicurarsi di soddisfare i [requisiti](#) per il connettore.
2. Crea e pubblica una funzione Lambda che invia dati di input al connettore.

Salvare il [codice di esempio](#) come file PY. Scarica e decomprimi il [AWS IoT Greengrass Core SDK per Python](#). Quindi, crea un pacchetto zip che contiene il file PY e la cartella greengrasssdk a livello root. Questo pacchetto zip è il pacchetto di distribuzione in cui carichi. AWS Lambda

Dopo aver creato la funzione Python 3.7 Lambda, pubblica una versione della funzione e crea un alias.

3. Configurare il gruppo Greengrass.

- a. Aggiungi la funzione Lambda tramite il relativo alias (consigliato). Configura il ciclo di vita Lambda come longevo (o nella "Pinned": `true` CLI).
- b. Aggiungi la risorsa segreta richiesta e concedi l'accesso in lettura alla funzione Lambda.
- c. Aggiungere il connettore e configurarne i relativi [parametri](#).
- d. Aggiungere sottoscrizioni che consentono al connettore di ricevere [i dati di input](#) e inviare [i dati di output](#) nei filtri degli argomenti supportati.
 - Imposta la funzione Lambda come origine, il connettore come destinazione e utilizza un filtro per argomenti di input supportato.
 - Imposta il connettore come origine, AWS IoT Core come destinazione e utilizza un filtro per l'argomento di output supportato. Utilizzi questo abbonamento per visualizzare i messaggi di stato nella AWS IoT console.

4. Distribuisci il gruppo.

5. Nella AWS IoT console, nella pagina Test, sottoscrivi l'argomento relativo ai dati di output per visualizzare i messaggi di stato dal connettore. La funzione Lambda di esempio è di lunga durata e inizia a inviare messaggi subito dopo l'implementazione del gruppo.

Al termine del test, puoi impostare il ciclo di vita Lambda su richiesta (o nella CLI) e "Pinned": `false` distribuire il gruppo. Ciò impedisce alla funzione di inviare messaggi.

Esempio

L'esempio seguente della funzione Lambda invia un messaggio di input al connettore. Questo esempio attiva un messaggio di testo.

```
import greengrasssdk
import json

iot_client = greengrasssdk.client('iot-data')
TXT_INPUT_TOPIC = 'twilio/txt'
CALL_INPUT_TOPIC = 'twilio/call'

def publish_basic_message():
```

```
txt = {
    "request": {
        "recipient" : {
            "name": "Darla",
            "phone_number": "+12345000000",
            "message": 'Hello from the edge'
        },
        "from_number" : "+19999999999"
    },
    "id" : "request123"
}

print("Message To Publish: ", txt)

client.publish(topic=TXT_INPUT_TOPIC,
               payload=json.dumps(txt))

publish_basic_message()

def lambda_handler(event, context):
    return
```

Licenze

Il connettore Twilio Notifications include i seguenti software/licenze di terze parti:

- [twilio-python](#)/MIT

Questo connettore è rilasciato ai sensi del contratto di [licenza del software Greengrass Core](#).

Changelog

La tabella seguente descrive le modifiche apportate a ciascuna versione del connettore.

Versione	Modifiche
5	Aggiunto il parametro <code>IsolationMode</code> per configurare la modalità di containerizzazione per il connettore.

Versione	Modifiche
4	È stato aggiornato il runtime Lambda a Python 3.7, che modifica i requisiti di runtime.
3	Correggere per ridurre l'eccessiva registrazione di log.
2	Miglioramenti e correzioni di bug minori.
1	Versione iniziale.

Un gruppo Greengrass può contenere una sola versione del connettore alla volta. Per informazioni sull'aggiornamento di una versione del connettore, consulta [the section called “Aggiornamento delle versioni dei connettori”](#).

Consulta anche

- [Integrazione con servizi e protocolli tramite i connettori](#)
- [the section called “Nozioni di base sui connettori \(console\)”](#)
- [the section called “Nozioni di base sui connettori \(CLI\)”](#)
- [Riferimento all'API Twilio](#)

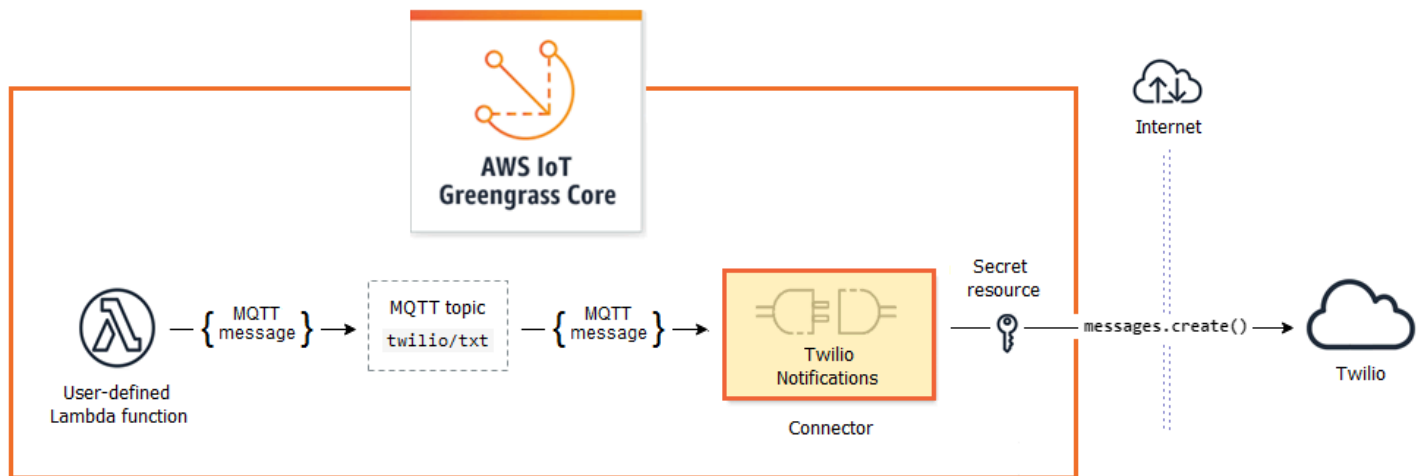
Nozioni di base sui connettori Greengrass (console)

Questa funzionalità è disponibile per AWS IoT Greengrass Core v1.7 e versioni successive.

Questo tutorial mostra come utilizzarlo per AWS Management Console lavorare con i connettori.

Utilizza i connettori per accelerare il ciclo di vita dello sviluppo. I connettori sono moduli precostituiti e riutilizzabili che semplificano l'interazione con servizi, protocolli e risorse. Possono aiutarti a distribuire più rapidamente la logica di business ai dispositivi Greengrass. Per ulteriori informazioni, consulta [Integrazione con servizi e protocolli tramite i connettori](#).

In questo tutorial, configuri e distribuisce il connettore [Twilio Notifications](#). Il connettore riceve le informazioni sui messaggi Twilio come dati di input, quindi attiva un messaggio di testo Twilio. Il flusso di dati viene mostrato nel seguente schema.



Dopo aver configurato il connettore, crei una funzione Lambda e un abbonamento.

- La funzione valuta i dati simulati da un sensore di temperatura. Pubblica in base a condizioni le informazioni sul messaggio Twilio in un argomento MQTT. Questo è l'argomento a cui il connettore effettua la sottoscrizione.
- La sottoscrizione consente alla funzione di effettuare la pubblicazione nell'argomento e al connettore di ricevere i dati dall'argomento.

Il connettore Twilio Notifications richiede un token di autenticazione Twilio per interagire con l'API Twilio. Il token è un tipo di testo segreto creato AWS Secrets Manager e referenziato da una risorsa di gruppo. Ciò consente di AWS IoT Greengrass creare una copia locale del segreto sul core di Greengrass, dove viene crittografato e reso disponibile al connettore. Per ulteriori informazioni, consulta [Distribuzione dei segreti nel core](#).

Il tutorial include le seguenti fasi di alto livello:

1. [Crea un segreto di Secrets Manager](#)
2. [Aggiunta di una risorsa segreta a un gruppo](#)
3. [Aggiunta di un connettore al gruppo](#)
4. [Creare un pacchetto di distribuzione della funzione Lambda](#)
5. [Creazione di una funzione Lambda](#)
6. [Aggiunta di una funzione al gruppo](#)
7. [Aggiunta di sottoscrizioni al gruppo](#)

8. [Distribuzione del gruppo.](#)
9. [the section called “Test della soluzione”](#)

Il completamento di questo tutorial richiede circa 20 minuti.

Prerequisiti

Per completare questo tutorial, è necessario quanto segue:

- Un gruppo Greengrass e un core Greengrass (v1.9.3 or later). Per informazioni su come creare un gruppo e un core Greengrass, consulta [Iniziare con AWS IoT Greengrass](#). Nel tutorial Nozioni di base sono descritte anche le fasi per l'installazione del software AWS IoT Greengrass Core.
- Python 3.7 installato sul dispositivo principale AWS IoT Greengrass .
- AWS IoT Greengrass deve essere configurato per supportare i segreti locali, come descritto in [Secrets Requirements](#).

Note

Questo requisito include l'autorizzazione all'accesso ai segreti di Secrets Manager. Se utilizzi il ruolo di servizio Greengrass predefinito, Greengrass è autorizzato a ottenere i valori dei segreti con nomi che iniziano con greengrass-.

- SID dell'account Twilio, token di autorizzazione e numero di telefono abilitato per Twilio. Dopo avere creato un progetto Twilio, questi valori sono disponibili nel pannello di controllo del progetto.

Note

Puoi utilizzare un account di prova Twilio. Se utilizzi un account di prova, devi aggiungere i numeri di telefono dei destinatari diversi da Twilio a un elenco di numeri di telefono verificati. Per ulteriori informazioni, vedi [Come lavorare con il tuo account di prova Twilio gratuito](#).

Fase 1: Creare un segreto di Secrets Manager

In questo passaggio, utilizzi la AWS Secrets Manager console per creare un tipo di testo segreto per il tuo token di autenticazione Twilio.

1. Accedere alla [console AWS Secrets Manager](#).

 Note

Per ulteriori informazioni su questo processo, consulta [Fase 1: Creare e archiviare il segreto AWS Secrets Manager nella Guida](#) per l'AWS Secrets Manager utente.

2. Scegli Archivia un nuovo segreto.
3. In Scegli il tipo di segreto, scegli Altro tipo di segreto.
4. In Specify the key/value pairs to be stored for this secret (Specifica le coppie chiave/valore da archiviare per il segreto), nella scheda Plaintext (Testo normale), immettere il token di autorizzazione Twilio. Rimuovi tutta la formattazione JSON e immetti solo il valore del token.
5. Mantieni aws/secretsmanager selezionato per la chiave di crittografia, quindi scegli Avanti.

 Note

Non ti viene addebitato alcun costo AWS KMS se utilizzi la chiave AWS gestita predefinita creata da Secrets Manager nel tuo account.

6. In Secret name (Nome segreto), immetti **greengrass-TwilioAuthToken**, quindi seleziona Next (Avanti).

 Note

Per impostazione predefinita, il ruolo di servizio Greengrass consente di AWS IoT Greengrass ottenere il valore dei segreti con nomi che iniziano con greengrass -. Per ulteriori informazioni, consulta [Requisiti dei segreti](#).

7. Questo tutorial non richiede la rotazione, quindi scegli disabilita la rotazione automatica, quindi scegli Avanti.
8. Nella pagina Review (Revisione), rivedi le impostazioni e quindi scegli Store (Archivia).

Dovrai quindi creare una risorsa segreta nel gruppo Greengrass che faccia riferimento al segreto.

Fase 2: aggiunta di una risorsa segreta a un gruppo Greengrass

In questa fase, aggiungerai una risorsa segreta al gruppo Greengrass. Questa risorsa è un riferimento al segreto creato nella fase precedente.

1. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).
2. Scegli il gruppo a cui aggiungere la risorsa segreta.
3. Nella pagina di configurazione del gruppo, scegli la scheda Risorse, quindi scorri verso il basso fino alla sezione Segreti. La sezione Segreti mostra le risorse segrete che appartengono al gruppo. È possibile aggiungere, modificare e rimuovere risorse segrete da questa sezione.

Note

In alternativa, la console consente di creare una risorsa segreta e segreta quando si configura un connettore o una funzione Lambda. È possibile eseguire questa operazione dalla pagina Configura parametri del connettore o dalla pagina Risorse della funzione Lambda.

4. Scegli Aggiungi nella sezione Segreti.
5. Nella pagina Aggiungi una risorsa segreta, inserisci **MyTwilioAuthToken** il nome della risorsa.
6. Per il Segreto, scegli greengrass-. TwilioAuthToken
7. Nella sezione Seleziona etichette (opzionale), l'etichetta AWSCURRENT staging rappresenta la versione più recente del segreto. Questa etichetta è sempre inclusa in una risorsa segreta.

Note

Questo tutorial richiede solo l' AWSCURRENT etichetta. Facoltativamente, puoi includere le etichette richieste dalla funzione o dal connettore Lambda.

8. Scegliere Add resource (Aggiungi risorsa).

Fase 3: aggiunta di un connettore al gruppo Greengrass

In questo passaggio, configuri i parametri per il [connettore Twilio Notifications](#) e lo aggiungi al gruppo.

1. Nella pagina di configurazione del gruppo, scegliere Connectors (Connettori), quindi Add a connector (Aggiungi un connettore).
2. Nella pagina Aggiungi connettore, scegli Notifiche Twilio.
3. Scegliere la versione .
4. Nella sezione Configurazione:
 - Per la risorsa del token di autenticazione Twilio, inserisci la risorsa che hai creato nel passaggio precedente.

 Note

Quando si immette la risorsa, l'ARN della proprietà segreta del token di autenticazione Twilio viene compilato automaticamente.

- In Default from phone number (Impostazione predefinita da numero di telefono), immettere il numero di telefono abilitato per Twilio.
 - In Twilio account SID (SID account Twilio), immettere il SID dell'account Twilio.
5. Scegliere Add resource (Aggiungi risorsa).

Fase 4: Creare un pacchetto di distribuzione della funzione Lambda

Per creare una funzione Lambda, devi prima creare un pacchetto di distribuzione della funzione Lambda che contenga il codice della funzione e le dipendenze. Le funzioni Greengrass Lambda richiedono il [AWS IoT Greengrass Core SDK](#) per attività come la comunicazione con i messaggi MQTT nell'ambiente principale e l'accesso ai segreti locali. Questo tutorial crea una funzione Python, quindi utilizzi la versione Python dell'SDK nel pacchetto di distribuzione.

1. Dalla pagina dei download di [AWS IoT Greengrass Core SDK](#), scarica AWS IoT Greengrass Core SDK per Python sul tuo computer.
2. Decomprimere il pacchetto scaricato per ottenere l'SDK. Il kit SDK è la cartella greengrasssdk.
3. Salvare la seguente funzione del codice Python nel file locale `temp_monitor.py`.

```
import greengrasssdk
import json
import random

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



```
# publish to the Twilio Notifications connector through the twilio/txt topic
def function_handler(event, context):
    temp = event['temperature']

    # check the temperature
    # if greater than 30C, send a notification
    if temp > 30:
        data = build_request(event)
        client.publish(topic='twilio/txt', payload=json.dumps(data))
        print('published:' + str(data))

    print('temperature:' + str(temp))
    return

# build the Twilio request from the input data
def build_request(event):
    to_name = event['to_name']
    to_number = event['to_number']
    temp_report = 'temperature:' + str(event['temperature'])

    return {
        "request": {
            "recipient": {
                "name": to_name,
                "phone_number": to_number,
                "message": temp_report
            }
        },
        "id": "request_" + str(random.randint(1,101))
    }
```

4. Comprimerle le voci seguenti nel file `temp_monitor_python.zip`. Al momento della creazione del file ZIP, includere solo il codice e le dipendenze, non la cartella che li contiene.
- `temp_monitor.py`. La logica dell'app.
 - `greengrasssdk`. Libreria richiesta per le funzioni Python Greengrass Lambda che pubblicano messaggi MQTT.

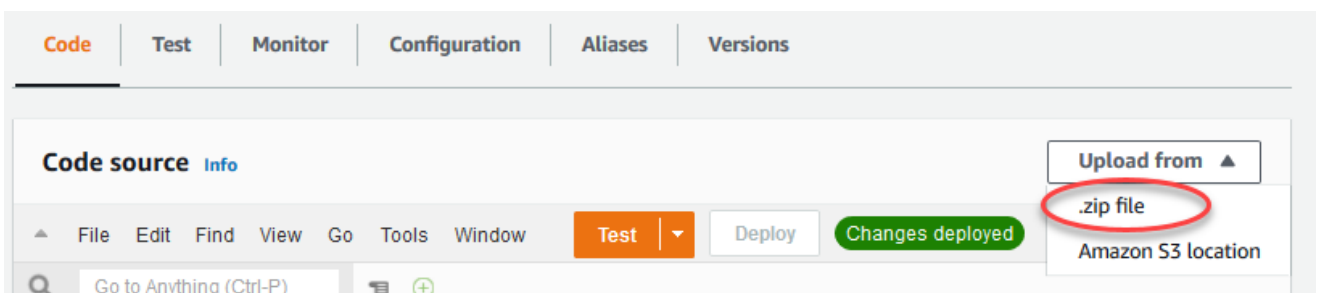
Questo è il tuo pacchetto di implementazione della funzione Lambda.

Ora, crea una funzione Lambda che utilizzi il pacchetto di distribuzione.

Fase 5: Creare una funzione Lambda nella console AWS Lambda

In questo passaggio, si utilizza la AWS Lambda console per creare una funzione Lambda e configurarla per utilizzare il pacchetto di distribuzione. In seguito, pubblicherai una versione della funzione e creerai un alias.

1. Innanzitutto, crea la funzione Lambda.
 - a. In AWS Management Console, scegli Servizi e apri la AWS Lambda console.
 - b. Scegli Crea funzione, quindi scegli Autore da zero.
 - c. Nella sezione Basic information (Informazioni di base), specifica i seguenti valori:
 - Nel campo Function name (Nome funzione), immettere **TempMonitor**.
 - In Runtime, scegliere Python 3.7.
 - Per le autorizzazioni, mantieni l'impostazione predefinita. Questo crea un ruolo di esecuzione che concede le autorizzazioni Lambda di base. Questo ruolo non viene utilizzato da AWS IoT Greengrass
 - d. Nella parte inferiore della pagina, scegli Crea funzione.
2. Quindi, registra il gestore e carica il pacchetto di distribuzione della funzione Lambda.
 - a. Nella scheda Codice, in Codice sorgente, scegli Carica da. Dal menu a discesa, scegli un file.zip.



- b. Scegli Carica, quindi scegli il pacchetto di temp_monitor_python.zip distribuzione. Quindi, scegliere Save (Salva).
- c. Nella scheda Codice della funzione, in Impostazioni di runtime, scegliete Modifica, quindi immettete i seguenti valori.
 - In Runtime, scegliere Python 3.7.

- In Handler (Gestore), immetti **temp_monitor.function_handler**

d. Seleziona Salva.

 Note

Il pulsante Test sulla AWS Lambda console non funziona con questa funzione. Il AWS IoT Greengrass Core SDK non contiene moduli necessari per eseguire le funzioni Greengrass Lambda in modo indipendente nella console. AWS Lambda Questi moduli (ad esempio,greengrass_common) vengono forniti alle funzioni dopo essere stati distribuiti nel core Greengrass.

3. Ora, pubblica la prima versione della tua funzione Lambda e crea un [alias per la](#) versione.

 Note

I gruppi Greengrass possono fare riferimento a una funzione Lambda tramite alias (consigliato) o per versione. L'utilizzo di un alias semplifica la gestione degli aggiornamenti del codice perché non è necessario modificare la tabella di sottoscrizione o la definizione del gruppo quando il codice della funzione viene aggiornato. Invece, è sufficiente indirizzare l'alias alla nuova versione della funzione.

- Nel menu Actions (Operazioni), seleziona Publish new version (Pubblica nuova versione).
- Per Version description (Descrizione versione), immettere **First version**, quindi scegliere Publish (Pubblica).
- Nella pagina di configurazione TempMonitor: 1, dal menu Azioni, scegli Crea alias.
- Nella pagina Create a new alias (Crea un nuovo alias), utilizza i seguenti valori:
 - Per Nome, immetti **GG_TempMonitor**.
 - In Version (Versione), selezionare 1.

 Note

AWS IoT Greengrass non supporta gli alias Lambda per le versioni \$LATEST.

e. Scegli Create (Crea) .

Ora sei pronto per aggiungere la funzione Lambda al tuo gruppo Greengrass.

Fase 6: Aggiungere una funzione Lambda al gruppo Greengrass

In questo passaggio, aggiungi la funzione Lambda al gruppo e quindi configuri il ciclo di vita e le variabili di ambiente. Per ulteriori informazioni, consulta [the section called “Controllo dell'esecuzione della funzione Greengrass Lambda”](#).

1. Nella pagina di configurazione del gruppo, scegli la scheda Funzioni Lambda.
2. In Funzioni My Lambda, scegli Aggiungi.
3. Nella pagina Aggiungi funzione Lambda, scegli TempMonitor la tua funzione Lambda.
4. Per la versione della funzione Lambda, scegli Alias: GG_. TempMonitor
5. Scegli Aggiungi funzione Lambda.

Fase 7: aggiunta di sottoscrizioni al gruppo Greengrass

In questo passaggio, aggiungi un abbonamento che consente alla funzione Lambda di inviare dati di input al connettore. Il connettore definisce gli argomenti MQTT a cui è sottoscritto. Pertanto, questa sottoscrizione utilizza uno degli argomenti. Si tratta dello argomento in cui la funzione di esempio effettua la pubblicazione.

Per questo tutorial, creerai anche abbonamenti che consentono alla funzione di ricevere letture simulate della temperatura AWS IoT e di AWS IoT ricevere informazioni sullo stato dal connettore.

1. Nella pagina di configurazione del gruppo, scegli la scheda Abbonamenti, quindi scegli Aggiungi abbonamento.
2. Nella pagina Crea un abbonamento, configura l'origine e la destinazione come segue:
 - a. Per Tipo di sorgente, scegli la funzione Lambda, quindi scegli. TempMonitor
 - b. Per Tipo di destinazione, scegli Connettore, quindi scegli Notifiche Twilio.
3. Per il filtro Argomento, scegli **twilio/txt**.
4. Scegli Crea sottoscrizione.
5. Ripeti i passaggi da 1 a 4 per creare un abbonamento che AWS IoT consenta di pubblicare messaggi sulla funzione.
 - a. Per Tipo di origine, scegli Servizio, quindi scegli IoT Cloud.

- b. Per Seleziona un obiettivo, scegli Funzione Lambda, quindi scegli TempMonitor
 - c. In Topic filter (Filtro argomento), immettere **temperature/input**.
6. Ripetere le fasi da 1 a 4 per creare una sottoscrizione che consenta al connettore di pubblicare messaggi in AWS IoT.
 - a. Per Tipo di sorgente, scegli Connettore, quindi scegli Notifiche Twilio.
 - b. Per il tipo di Target, scegli Service, quindi scegli IoT Cloud.
 - c. In Topic filter (Filtro argomento), viene inserito **twilio/message/status**. Si tratta dell'argomento predefinito in cui pubblica il connettore.

Fase 8: distribuzione del gruppo Greengrass

Distribuire il gruppo al nuovo dispositivo core.

1. Assicurati che il AWS IoT Greengrass core sia in funzione. Esegui i seguenti comandi nel terminale di Raspberry Pi in base alle esigenze.
 - a. Per controllare se il daemon è in esecuzione:

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

Se l'output contiene una voce root per `/greengrass/ggc/packages/ggc-version/bin/daemon`, allora il daemon è in esecuzione.

Note

La versione indicata nel percorso dipende dalla versione del software AWS IoT Greengrass Core installata sul dispositivo principale.

- b. Per avviare il demone:

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

2. Nella pagina di configurazione del gruppo, scegli Deploy.
3.
 - a. Nella scheda Funzioni Lambda, nella sezione Funzioni System Lambda, seleziona Rilevatore IP e scegli Modifica.

- b. Nella finestra di dialogo Modifica impostazioni del rilevatore IP, seleziona Rileva e sostituisci automaticamente gli endpoint del broker MQTT.
- c. Seleziona Salva.

Questo consente ai dispositivi di acquisire automaticamente informazioni di base sulla connettività, come, ad esempio indirizzo IP, DNS e numero della porta. Il rilevamento automatico è consigliato, ma supporta AWS IoT Greengrass anche gli endpoint specificati manualmente. Ti viene chiesto il metodo di individuazione solo la prima volta che il gruppo viene distribuito.

 Note

Se richiesto, concedi l'autorizzazione a creare il ruolo di [servizio Greengrass](#) e associarlo al Account AWS tuo ruolo attuale. Regione AWS Questo ruolo consente di accedere AWS IoT Greengrass alle tue risorse nei AWS servizi.

Nella pagina Deployments (Distribuzioni) vengono visualizzati il timestamp della distribuzione, l'ID versione e lo stato. Una volta completata, lo stato visualizzato per la distribuzione dovrebbe essere Completato.

Per la risoluzione dei problemi, consultare [Risoluzione dei problemi](#).

 Note

Un gruppo Greengrass può contenere una sola versione del connettore alla volta. Per informazioni sull'aggiornamento di una versione del connettore, consulta [the section called "Aggiornamento delle versioni dei connettori"](#).

Test della soluzione

1. Nella home page della AWS IoT console, scegli Test.
2. Per Sottoscrivi all'argomento, utilizza i seguenti valori, quindi scegli Sottoscrivi. Il connettore Twilio Notifications pubblica informazioni sullo stato di questo argomento.

Proprietà	Valore
Argomento sottoscrizione	twilio/message/status
Visualizzazione payload MQTT	Visualizza i payload come stringhe

3. Per Pubblica su argomento, utilizzate i seguenti valori, quindi scegliete Pubblica per richiamare la funzione.

Proprietà	Valore
Argomento	temperatura/input
Messaggio	<i>recipient-name</i> Sostituítelo con un nome e <i>recipient-phone-number</i> con il numero di telefono del destinatario del messaggio di testo. Esempio: +12345000000 <pre>{ "to_name": " <i>recipient-name</i> ", "to_number": " <i>recipient-phone-number</i> ", "temperature": 31 }</pre> Se utilizzi un account di prova, devi aggiungere numeri di telefono dei destinatari diversi da Twilio a un elenco di numeri di telefono verificati. Per ulteriori informazioni, consulta Verifica del tuo numero di telefono personale.

Se l'operazione viene completata, il destinatario riceve il messaggio di testo e la console mostra lo stato success dai [dati di output](#).

A questo punto, è necessario modificare temperature nel messaggio di input in **29** e pubblicare. Poiché è inferiore a 30, la TempMonitor funzione non attiva un messaggio Twilio.

Consulta anche

- [Integrazione con servizi e protocolli tramite i connettori](#)
- [the section called “AWS-connettori Greengrass forniti”](#)

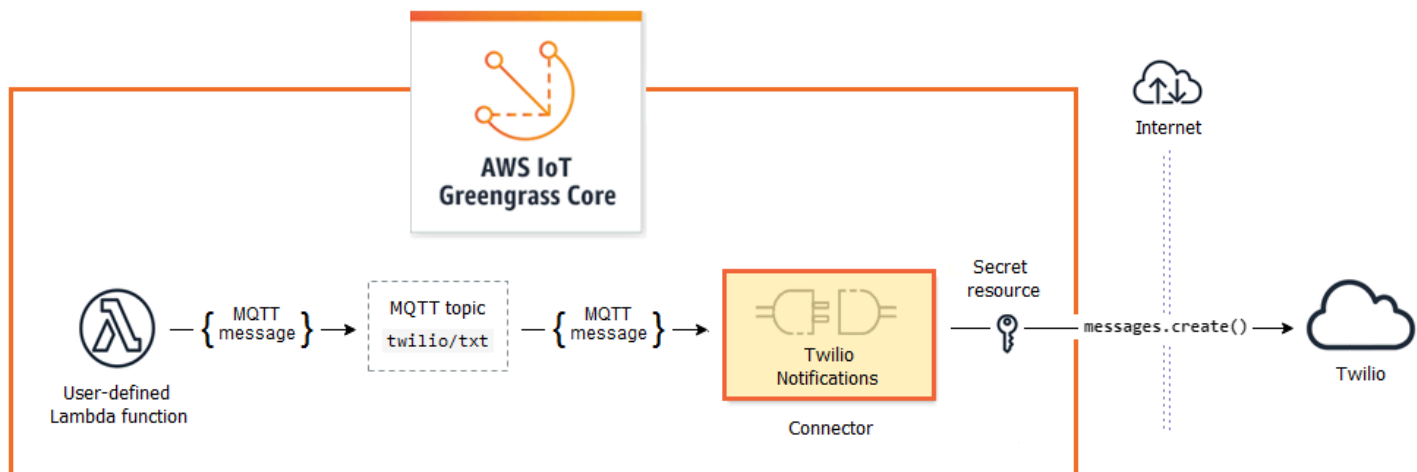
Nozioni di base sui connettori Greengrass (CLI)

Questa funzionalità è disponibile per AWS IoT Greengrass Core v1.7 e versioni successive.

Questo tutorial mostra come utilizzarlo per AWS CLI lavorare con i connettori.

Utilizza i connettori per accelerare il ciclo di vita dello sviluppo. I connettori sono moduli precostituiti e riutilizzabili che semplificano l'interazione con servizi, protocolli e risorse. Possono aiutarti a distribuire più rapidamente la logica di business ai dispositivi Greengrass. Per ulteriori informazioni, consulta [Integrazione con servizi e protocolli tramite i connettori](#).

In questo tutorial, configuri e distribuisce il connettore [Twilio Notifications](#). Il connettore riceve le informazioni sui messaggi Twilio come dati di input, quindi attiva un messaggio di testo Twilio. Il flusso di dati viene mostrato nel seguente schema.



Dopo aver configurato il connettore, crei una funzione Lambda e un abbonamento.

- La funzione valuta i dati simulati da un sensore di temperatura. Pubblica in base a condizioni le informazioni sul messaggio Twilio in un argomento MQTT. Questo è l'argomento a cui il connettore effettua la sottoscrizione.
- La sottoscrizione consente alla funzione di effettuare la pubblicazione nell'argomento e al connettore di ricevere i dati dall'argomento.

Il connettore Twilio Notifications richiede un token di autenticazione Twilio per interagire con l'API Twilio. Il token è un tipo di testo segreto creato AWS Secrets Manager e referenziato da una risorsa di gruppo. Ciò consente di AWS IoT Greengrass creare una copia locale del segreto sul core di Greengrass, dove viene crittografato e reso disponibile al connettore. Per ulteriori informazioni, consulta [Distribuzione dei segreti nel core](#).

Il tutorial include le seguenti fasi di alto livello:

1. [Crea un segreto di Secrets Manager](#)
2. [Creazione della versione e della definizione della risorsa](#)
3. [Creazione della versione e della definizione del connettore](#)
4. [Creare un pacchetto di distribuzione della funzione Lambda](#)
5. [Creazione di una funzione Lambda](#)
6. [Creazione della versione e della definizione della funzione](#)
7. [Creazione della versione e della definizione dell'abbonamento](#)
8. [Creazione di una versione del gruppo](#)
9. [Crea distribuzione](#)
10. [the section called "Test della soluzione"](#)

Il completamento di questo tutorial richiede circa 30 minuti.

Utilizzando l'API AWS IoT Greengrass

È utile comprendere i seguenti schemi quando lavori con i gruppi Greengrass e i componenti dei gruppi (ad esempio connettori, funzioni e risorse del gruppo).

- In alto nella gerarchia, un componente dispone di un oggetto definizione, ovvero un container di oggetti versione. Una versione è invece un container dei connettori, delle funzioni o di altri tipi di componenti.

- Quando effettui una distribuzione nel core Greengrass, distribuisce una specifica versione del gruppo. Una versione del gruppo può contenere una versione di ciascun tipo di componente. È necessario un core, ma gli altri sono inclusi in base alle necessità.
- Le versioni non possono essere modificate. Pertanto, se desideri apportare modifiche, dovrai crearne di nuove.

Tip

Se ricevi un errore quando esegui un AWS CLI comando, aggiungi il `--debug` parametro e quindi esegui nuovamente il comando per ottenere ulteriori informazioni sull'errore.

L'AWS IoT Greengrass API consente di creare più definizioni per un tipo di componente. Ad esempio, puoi creare un oggetto `FunctionDefinition` ogni volta che crei una `FunctionDefinitionVersion` oppure puoi aggiungere nuove versioni a una definizione esistente. Questa flessibilità ti consente di personalizzare il sistema di gestione delle versioni.

Prerequisiti

Per completare questo tutorial, è necessario quanto segue:

- Un gruppo Greengrass e un core Greengrass (v1.9.3 or later). Per informazioni su come creare un gruppo e un core Greengrass, consulta [Iniziare con AWS IoT Greengrass](#). Nel tutorial Nozioni di base sono descritte anche le fasi per l'installazione del software AWS IoT Greengrass Core.
- Python 3.7 installato sul dispositivo principale AWS IoT Greengrass .
- AWS IoT Greengrass deve essere configurato per supportare i segreti locali, come descritto in [Secrets Requirements](#).

Note

Questo requisito include l'autorizzazione all'accesso ai segreti di Secrets Manager. Se utilizzi il ruolo di servizio Greengrass predefinito, Greengrass è autorizzato a ottenere i valori dei segreti con nomi che iniziano con `greengrass-`.

- SID dell'account Twilio, token di autorizzazione e numero di telefono abilitato per Twilio. Dopo avere creato un progetto Twilio, questi valori sono disponibili nel pannello di controllo del progetto.

 Note

Puoi utilizzare un account di prova Twilio. Se utilizzi un account di prova, devi aggiungere i numeri di telefono dei destinatari diversi da Twilio a un elenco di numeri di telefono verificati. Per ulteriori informazioni, vedi [Come lavorare con il tuo account di prova Twilio gratuito](#).

- AWS CLI installato e configurato sul tuo computer. Per ulteriori informazioni, vedere [Installazione AWS Command Line Interface e configurazione di AWS CLI nella Guida per l'AWS Command Line Interface utente](#).

Gli esempi in questo tutorial si riferiscono a Linux e ad altri sistemi basati su Unix. Se utilizzi Windows, consulta [Specificazione dei valori dei parametri per AWS Command Line Interface conoscere le](#) differenze di sintassi.

Se il comando include una stringa JSON, il tutorial fornisce un esempio che ha JSON in un'unica riga. In alcuni sistemi, potrebbe essere più semplice modificare ed eseguire comandi utilizzando questo formato.

Fase 1: Creare un segreto di Secrets Manager

In questo passaggio, utilizzi l' AWS Secrets Manager API per creare un segreto per il tuo token di autenticazione Twilio.

1. È necessario creare innanzitutto il segreto.
 - Sostituiscilo *twilio-auth-token* con il tuo token di autenticazione Twilio.

```
aws secretsmanager create-secret --name greengrass-TwilioAuthToken --secret-string twilio-auth-token
```

Note

Per impostazione predefinita, il ruolo di servizio Greengrass consente di AWS IoT Greengrass ottenere il valore dei segreti con nomi che iniziano con greengrass -. Per ulteriori informazioni, consulta [Requisiti dei segreti](#).

2. Copiare l'ARN del segreto dall'output. Lo usi per creare la risorsa segreta e configurare il connettore Twilio Notifications.

Fase 2: creazione della versione e della definizione della risorsa

In questo passaggio, utilizzi l' AWS IoT Greengrass API per creare una risorsa segreta per il tuo segreto di Secrets Manager.

1. Creare una definizione di risorsa che includa una versione iniziale.
 - Sostituisci *secret-arn* con il ARN segreto che hai copiato nel passaggio precedente.

JSON Expanded

```
aws greengrass create-resource-definition --name MyGreengrassResources --
initial-version '{
  "Resources": [
    {
      "Id": "TwilioAuthToken",
      "Name": "MyTwilioAuthToken",
      "ResourceDataContainer": {
        "SecretsManagerSecretResourceData": {
          "ARN": "secret-arn"
        }
      }
    }
  ]
}'
```

JSON Single-line

```
aws greengrass create-resource-definition \  
--name MyGreengrassResources \  
--initial-version '{"Resources": [{"Id": "TwilioAuthToken",  
  "Name": "MyTwilioAuthToken", "ResourceDataContainer":  
    {"SecretsManagerSecretResourceData": {"ARN": "secret-arn"}}}]}'
```

2. Copiare la LatestVersionArn della definizione di risorsa dall'output. È possibile utilizzare questo valore per aggiungere la definizione di risorsa alla versione del gruppo distribuita nel core.

Fase 3: creazione della versione e della definizione del connettore

In questo passaggio, configuri i parametri per il connettore Twilio Notifications.

1. Creare una definizione del connettore con una versione iniziale.
 - *account-sid* Sostituiscilo con il SID del tuo account Twilio.
 - Sostituisci *secret-arn* con il segreto ARN del tuo Secrets Manager. Il connettore lo utilizzerà per ottenere il valore del segreto locale.
 - *phone-number* Sostituiscilo con il tuo numero di telefono abilitato a Twilio. Twilio lo utilizza per iniziare il messaggio di testo. Questo può essere sostituito nel payload del messaggio di input. Utilizza il seguente formato: +19999999999.

JSON Expanded

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --  
initial-version '{  
  "Connectors": [  
    {  
      "Id": "MyTwilioNotificationsConnector",  
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/  
TwilioNotifications/versions/4",  
      "Parameters": {  
        "TWILIO_ACCOUNT_SID": "account-sid",  
        "TwilioAuthTokenSecretArn": "secret-arn",  
        "TwilioAuthTokenSecretArn-ResourceId": "TwilioAuthToken",
```

```

        "DefaultFromPhoneNumber": "phone-number"
    }
}
]
}'

```

JSON Single-line

```

aws greengrass create-connector-definition \
--name MyGreengrassConnectors \
--initial-version '{"Connectors": [{"Id": "MyTwilioNotificationsConnector",
"ConnectorArn": "arn:aws:greengrass:region::/connectors/TwilioNotifications/
versions/4", "Parameters": {"TWILIO_ACCOUNT_SID": "account-sid",
"TwilioAuthTokenSecretArn": "secret-arn", "TwilioAuthTokenSecretArn-
ResourceId": "TwilioAuthToken", "DefaultFromPhoneNumber": "phone-number"}}]}'

```

Note

TwilioAuthToken è l'ID utilizzato nella fase precedente per creare la risorsa segreta.

2. Copiare il LatestVersionArn della definizione del connettore dall'output. È possibile utilizzare questo valore per aggiungere la definizione del connettore alla versione del gruppo distribuita nel core.

Fase 4: Creare un pacchetto di distribuzione della funzione Lambda

Per creare una funzione Lambda, devi prima creare un pacchetto di distribuzione della funzione Lambda che contenga il codice della funzione e le dipendenze. Le funzioni Greengrass Lambda richiedono il [AWS IoT Greengrass Core SDK](#) per attività come la comunicazione con i messaggi MQTT nell'ambiente principale e l'accesso ai segreti locali. Questo tutorial crea una funzione Python, quindi utilizzi la versione Python dell'SDK nel pacchetto di distribuzione.

1. Dalla pagina dei download di [AWS IoT Greengrass Core SDK](#), scarica AWS IoT Greengrass Core SDK per Python sul tuo computer.
2. Decomprimere il pacchetto scaricato per ottenere l'SDK. Il kit SDK è la cartella greengrasssdk.
3. Salvare la seguente funzione del codice Python nel file locale `temp_monitor.py`.

```
import greengrasssdk
```

```
import json
import random

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

# publish to the Twilio Notifications connector through the twilio/txt topic
def function_handler(event, context):
    temp = event['temperature']

    # check the temperature
    # if greater than 30C, send a notification
    if temp > 30:
        data = build_request(event)
        client.publish(topic='twilio/txt', payload=json.dumps(data))
        print('published:' + str(data))

    print('temperature:' + str(temp))
    return

# build the Twilio request from the input data
def build_request(event):
    to_name = event['to_name']
    to_number = event['to_number']
    temp_report = 'temperature:' + str(event['temperature'])

    return {
        "request": {
            "recipient": {
                "name": to_name,
                "phone_number": to_number,
                "message": temp_report
            }
        },
        "id": "request_" + str(random.randint(1,101))
    }
```

4. Comprimere le voci seguenti nel file `temp_monitor_python.zip`. Al momento della creazione del file ZIP, includere solo il codice e le dipendenze, non la cartella che li contiene.
- `temp_monitor.py`. La logica dell'app.
 - `greengrasssdk`. Libreria richiesta per le funzioni Python Greengrass Lambda che pubblicano messaggi MQTT.

Questo è il tuo pacchetto di implementazione della funzione Lambda.

Fase 5: Creare una funzione Lambda

Ora, crea una funzione Lambda che utilizzi il pacchetto di distribuzione.

1. Crea un ruolo IAM in modo da poter passare il ruolo ARN quando crei la funzione.

JSON Expanded

```
aws iam create-role --role-name Lambda_empty --assume-role-policy '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}'
```

JSON Single-line

```
aws iam create-role --role-name Lambda_empty --assume-role-policy '{"Version":
"2012-10-17", "Statement": [{"Effect": "Allow", "Principal": {"Service":
"lambda.amazonaws.com"},"Action": "sts:AssumeRole"}]}'
```

Note

AWS IoT Greengrass non utilizza questo ruolo perché le autorizzazioni per le funzioni Greengrass Lambda sono specificate nel ruolo del gruppo Greengrass. Per questo tutorial, viene creato un ruolo vuoto.

2. Copia il valore Arn dall'output.

3. Usa l' AWS Lambda API per creare la funzione. TempMonitor Il comando seguente presuppone che il file ZIP si trovi nella directory corrente.

- Sostituisci l'*role-arn* con l'Arn copiato.

```
aws lambda create-function \  
--function-name TempMonitor \  
--zip-file fileb://temp_monitor_python.zip \  
--role role-arn \  
--handler temp_monitor.function_handler \  
--runtime python3.7
```

4. Pubblicare una versione della funzione.

```
aws lambda publish-version --function-name TempMonitor --description 'First  
version'
```

5. Creare un alias della versione pubblicata.

I gruppi Greengrass possono fare riferimento a una funzione Lambda tramite alias (consigliato) o per versione. L'utilizzo di un alias semplifica la gestione degli aggiornamenti del codice perché non è necessario modificare la tabella di sottoscrizione o la definizione del gruppo quando il codice della funzione viene aggiornato. Invece, è sufficiente indirizzare l'alias alla nuova versione della funzione.

 Note

AWS IoT Greengrass non supporta gli alias Lambda per le versioni \$LATEST.

```
aws lambda create-alias --function-name TempMonitor --name GG_TempMonitor --  
function-version 1
```

6. Copia il valore AliasArn dall'output. Questo valore viene utilizzato quando si configura la funzione per AWS IoT Greengrass e quando si crea un abbonamento.

Ora sei pronto per configurare la funzione per AWS IoT Greengrass.

Fase 6: creazione della versione e della definizione della funzione

Per utilizzare una funzione Lambda su un AWS IoT Greengrass core, si crea una versione di definizione della funzione che fa riferimento alla funzione Lambda tramite alias e definisce la configurazione a livello di gruppo. Per ulteriori informazioni, consulta [the section called “Controllo dell'esecuzione della funzione Greengrass Lambda”](#).

1. Creare una definizione di funzione che includa una versione iniziale.

- *alias-arn* Sostituiscila con AliasArn quella che hai copiato quando hai creato l'alias.

JSON Expanded

```
aws greengrass create-function-definition --name MyGreengrassFunctions --
initial-version '{
  "Functions": [
    {
      "Id": "TempMonitorFunction",
      "FunctionArn": "alias-arn",
      "FunctionConfiguration": {
        "Executable": "temp_monitor.function_handler",
        "MemorySize": 16000,
        "Timeout": 5
      }
    }
  ]
}'
```

JSON Single-line

```
aws greengrass create-function-definition \
--name MyGreengrassFunctions \
--initial-version '{"Functions": [{"Id": "TempMonitorFunction",
"FunctionArn": "alias-arn", "FunctionConfiguration": {"Executable":
"temp_monitor.function_handler", "MemorySize": 16000,"Timeout": 5}}]}'
```

2. Copia il valore LatestVersionArn dall'output. È possibile utilizzare questo valore per aggiungere la definizione di funzione alla versione del gruppo distribuita nel core.

3. Copia il valore Id dall'output. È possibile utilizzare questo valore successivamente, al momento dell'aggiornamento della funzione.

Fase 7: creazione della versione e della definizione dell'abbonamento

In questo passaggio, aggiungi un abbonamento che consente alla funzione Lambda di inviare dati di input al connettore. Il connettore definisce gli argomenti MQTT a cui è sottoscritto. Pertanto, questa sottoscrizione utilizza uno degli argomenti. Si tratta dello argomento in cui la funzione di esempio effettua la pubblicazione.

Per questo tutorial, creerai anche abbonamenti che consentono alla funzione di ricevere letture simulate della temperatura AWS IoT e di AWS IoT ricevere informazioni sullo stato dal connettore.

1. Creare una definizione di abbonamento che includa una versione iniziale contenente gli abbonamenti.
 - *alias-arn* Sostituiscilo con AliasArn quello che hai copiato quando hai creato l'alias per la funzione. Utilizzare questo ARN per entrambi gli abbonamenti.

JSON Expanded

```
aws greengrass create-subscription-definition --initial-version '{
  "Subscriptions": [
    {
      "Id": "TriggerNotification",
      "Source": "alias-arn",
      "Subject": "twilio/txt",
      "Target": "arn:aws:greengrass:region::/connectors/
TwilioNotifications/versions/4"
    },
    {
      "Id": "TemperatureInput",
      "Source": "cloud",
      "Subject": "temperature/input",
      "Target": "alias-arn"
    },
    {
      "Id": "OutputStatus",
```

```

        "Source": "arn:aws:greengrass:region::/connectors/
TwilioNotifications/versions/4",
        "Subject": "twilio/message/status",
        "Target": "cloud"
    }
]
}'

```

JSON Single-line

```

aws greengrass create-subscription-definition \
--initial-version '{"Subscriptions": [{"Id": "TriggerNotification", "Source":
"alias-arn", "Subject": "twilio/txt", "Target": "arn:aws:greengrass:region::/
connectors/TwilioNotifications/versions/4"}, {"Id": "TemperatureInput",
"Source": "cloud", "Subject": "temperature/input", "Target": "alias-arn"},
{"Id": "OutputStatus", "Source": "arn:aws:greengrass:region::/connectors/
TwilioNotifications/versions/4", "Subject": "twilio/message/status", "Target":
"cloud"}]}'

```

2. Copia il valore LatestVersionArn dall'output. È possibile utilizzare questo valore per aggiungere la definizione dell'abbonamento alla versione del gruppo distribuita nel core.

Fase 8: creazione di una versione del gruppo

A questo punto è possibile creare una versione del gruppo che contenga tutte le voci da distribuire. A questo scopo, è necessario creare una versione di gruppo che faccia riferimento alla versione di destinazione di ciascun tipo di componente.

Ottenere innanzitutto l'ID del gruppo e l'ARN della versione della definizione del core. Questi valori sono necessari per creare la versione del gruppo.

1. Ottenere l'ID del gruppo e la versione gruppo più recente:
 - a. Ottieni il gruppo Greengrass IDs di destinazione e la versione del gruppo. Questa procedura presuppone che questa sia la versione più recente del gruppo e del gruppo. La seguente query restituisce il gruppo creato più di recente.

```

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

```

In alternativa, puoi eseguire query in base al nome. I nomi dei gruppi non devono essere univoci, pertanto potrebbero essere restituiti più gruppi.

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

 Note

Puoi trovare questi valori anche nella AWS IoT console. L'ID gruppo viene visualizzato nella pagina Settings (Impostazioni) del gruppo. IDs Le versioni del gruppo vengono visualizzate nella scheda Distribuzioni del gruppo.

- b. Copiare l'Id del gruppo di destinazione dall'output. Questo valore viene utilizzato per ottenere la versione della definizione del core e durante la distribuzione del gruppo.
 - c. Copiare la LatestVersion dall'output, che corrisponde all'ID dell'ultima versione aggiunta al gruppo. Questo valore viene utilizzato per ottenere la versione della definizione del core.
2. Per ottenere l'ARN della versione di definizione del core:
- a. Ottenere la versione del gruppo. In questa fase, si presume che la versione del gruppo più recente includa una versione della definizione del core.
 - *group-id* Sostituiscila con quella Id che hai copiato per il gruppo.
 - Sostituisci *group-version-id* con LatestVersion quello che hai copiato per il gruppo.

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

- b. Copia il valore CoreDefinitionVersionArn dall'output.
3. Creare una versione del gruppo.
- Sostituisci *group-id* con Id quello che hai copiato per il gruppo.
 - Sostituiscilo *core-definition-version-arn* con CoreDefinitionVersionArn quello che hai copiato per la versione di definizione principale.
 - Sostituisci *resource-definition-version-arn* con LatestVersionArn quello che hai copiato per la definizione della risorsa.

- *connector-definition-version-arn* Sostituiscilo con LatestVersionArn quello che hai copiato per la definizione del connettore.
- *function-definition-version-arn* Sostituiscilo con LatestVersionArn quello che hai copiato per la definizione della funzione.
- Sostituisci *subscription-definition-version-arn* con LatestVersionArn quello che hai copiato per la definizione della sottoscrizione.

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

4. Copia il valore di Version dall'output. Questo è l'ID della versione del gruppo. È possibile utilizzare questo valore per distribuire la versione del gruppo.

Fase 9: creazione di una distribuzione

Distribuire il gruppo al nuovo dispositivo core.

1. In un terminale del dispositivo principale, assicurati che il AWS IoT Greengrass demone sia in esecuzione.
 - a. Per controllare se il daemon è in esecuzione:

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

Se l'output contiene una voce root per /greengrass/ggc/packages/1.11.6/bin/daemon, allora il daemon è in esecuzione.

- b. Per avviare il demone:

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

2. Crea una implementazione .

- *group-id* Sostituiscilo con Id quello che hai copiato per il gruppo.
- Sostituisci *group-version-id* con Version quello che hai copiato per la nuova versione del gruppo.

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

3. Copia il valore DeploymentId dall'output.

4. Ottenere lo stato della distribuzione.

- Sostituiscilo *group-id* con Id quello che hai copiato per il gruppo.
- Sostituiscilo *deployment-id* con DeploymentId quello che hai copiato per la distribuzione.

```
aws greengrass get-deployment-status \  
--group-id group-id \  
--deployment-id deployment-id
```

Se lo stato è Success, la distribuzione è avvenuta con successo. Per la risoluzione dei problemi, consultare [Risoluzione dei problemi](#).

Test della soluzione

1. Nella home page della AWS IoT console, scegli Test.
2. Per Sottoscrivi all'argomento, utilizza i seguenti valori, quindi scegli Sottoscrivi. Il connettore Twilio Notifications pubblica informazioni sullo stato di questo argomento.

Proprietà	Valore
Argomento sottoscrizione	twilio/message/status
Visualizzazione payload MQTT	Visualizza i payload come stringhe

3. Per Pubblica su argomento, utilizzate i seguenti valori, quindi scegliete Pubblica per richiamare la funzione.

Proprietà	Valore
Argomento	temperatura/input
Messaggio	<i>recipient-name</i> Sostituítelo con un nome e <i>recipient-phone-number</i> con il numero di telefono del destinatario del messaggio di testo. Esempio: +12345000000 <pre>{ "to_name": " <i>recipient-name</i> ", "to_number": " <i>recipient-phone-number</i> ", "temperature": 31 }</pre> Se utilizzi un account di prova, devi aggiungere numeri di telefono dei destinatari diversi da Twilio a un elenco di numeri di telefono verificati. Per ulteriori informazioni, consulta Verifica del tuo numero di telefono personale.

Se l'operazione viene completata, il destinatario riceve il messaggio di testo e la console mostra lo stato success dai [dati di output](#).

A questo punto, è necessario modificare temperature nel messaggio di input in **29** e pubblicare. Poiché è inferiore a 30, la TempMonitor funzione non attiva un messaggio Twilio.

Consulta anche

- [Integrazione con servizi e protocolli tramite i connettori](#)
- [the section called "AWS-connettori Greengrass forniti"](#)

- [the section called “Nozioni di base sui connettori \(console\)”](#)
- [AWS Secrets Manager comandi nel AWS CLI Command Reference](#)
- [AWS Identity and Access Management comandi \(IAM\) nel AWS CLI Command Reference](#)
- [AWS Lambda comandi nel AWS CLI Command Reference](#)
- [AWS IoT Greengrass comandi nel AWS CLI Command Reference](#)

API Greengrass Discovery RESTful

Tutti i dispositivi client che comunicano con un AWS IoT Greengrass core devono far parte di un gruppo Greengrass. Ogni gruppo deve disporre di un core Greengrass. L'API Discovery consente ai dispositivi di recuperare le informazioni necessarie per connettersi a un core Greengrass che si trova nello stesso gruppo Greengrass del dispositivo client. Quando un dispositivo client è online per la prima volta, può connettersi al AWS IoT Greengrass servizio e utilizzare l'API Discovery per trovare:

- Il gruppo al quale appartiene. Un dispositivo client può far parte di un massimo di 10 gruppi.
- L'indirizzo IP e la porta per il core Greengrass nel gruppo.
- Il certificato emesso da una CA del gruppo, che può essere utilizzato per autenticare il dispositivo core Greengrass.

Note

I dispositivi client possono anche utilizzare il AWS IoT dispositivo SDKs per scoprire informazioni di connettività per un core Greengrass. Per ulteriori informazioni, consulta [AWS IoT SDK del dispositivo](#).

Per utilizzare questa API, invia le richieste HTTP all'endpoint dell'API Discovery. Per esempio:

```
https://greengrass-ats.iot.region.amazonaws.com:port/greengrass/discover/thing/thing-name
```

Per un elenco delle regioni e degli endpoint di Amazon Web Services supportati per l'API AWS IoT Greengrass Discovery, consulta [AWS IoT Greengrass endpoint e quote nel](#). Riferimenti generali di AWS Si tratta di un'API esclusivamente del piano dei dati. Gli endpoint per la gestione e le AWS IoT Core operazioni di gruppo sono diversi dagli endpoint dell'API Discovery.

Richiesta

La richiesta contiene le intestazioni HTTP standard e viene inviata all'endpoint Greengrass Discovery, come indicato nei seguenti esempi.

Il numero di porta varia a seconda che il core sia configurato per inviare il traffico HTTPS sulla porta 8443 o 443. Per ulteriori informazioni, consulta [the section called “Connessione alla porta 443 o tramite un proxy di rete”](#).

Porta 8443

```
HTTP GET https://greengrass-ats.iot.region.amazonaws.com:8443/greengrass/discover/thing/thing-name
```

Porta 443

```
HTTP GET https://greengrass-ats.iot.region.amazonaws.com:443/greengrass/discover/thing/thing-name
```

I client che si connettono alla porta 443 devono implementare l'estensione TLS [Application Layer Protocol Negotiation \(ALPN\)](#) e passare come. x-amzn-http-ca ProtocolName ProtocolNameList Per ulteriori informazioni, consulta [Protocols](#) nella Developer Guide.AWS IoT

Note

Questi esempi utilizzano Amazon Trust Services (ATS) con i certificati CA root ATS (scelta consigliata). Gli endpoint devono corrispondere al tipo di certificato CA root. Per ulteriori informazioni, consulta [the section called “Gli endpoint del servizio devono corrispondere al tipo di certificato”](#).

Risposta

In caso di esito positivo, la risposta include le intestazioni HTTP standard più il codice e il corpo seguenti:

```
HTTP 200  
BODY: response document
```

Per ulteriori informazioni, consulta [Documenti di risposta di individuazione di esempio](#).

Autorizzazione alla scoperta

Per recuperare le informazioni sulla connettività, è necessaria una policy che permetta all'intermediario di eseguire l'operazione `greengrass:Discover`. L'autenticazione reciproca TLS con certificato client è l'unica forma accettata di autenticazione. Di seguito è riportato un esempio di policy che permette a un intermediario di eseguire questa operazione:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "greengrass:Discover",
    "Resource": ["arn:aws:iot:us-west-2:123456789012:thing/MyThingName"]
  }]
}
```

Documenti di risposta di individuazione di esempio

Il seguente documento mostra la risposta per un dispositivo client che fa parte di un gruppo con un certificato CA Greengrass core, un endpoint e un gruppo:

```
{
  "GGGroups": [
    {
      "GGGroupId": "gg-group-01-id",
      "Cores": [
        {
          "thingArn": "core-01-thing-arn",
          "Connectivity": [
            {
              "id": "core-01-connection-id",
              "hostAddress": "core-01-address",
              "portNumber": core-01-port,
              "metadata": "core-01-description"
            }
          ]
        }
      ]
    }
  ],
  "CAs": [
    "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
  ]
}
```

```

    }
  ]
}

```

Il seguente documento mostra la risposta per un dispositivo client che è membro di due gruppi con un core Greengrass, più endpoint e più certificati CA di gruppo:

```

{
  "GGGroups": [
    {
      "GGGroupId": "gg-group-01-id",
      "Cores": [
        {
          "thingArn": "core-01-thing-arn",
          "Connectivity": [
            {
              "id": "core-01-connection-id",
              "hostAddress": "core-01-address",
              "portNumber": core-01-port,
              "metadata": "core-01-connection-1-description"
            },
            {
              "id": "core-01-connection-id-2",
              "hostAddress": "core-01-address-2",
              "portNumber": core-01-port-2,
              "metadata": "core-01-connection-2-description"
            }
          ]
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
      ]
    },
    {
      "GGGroupId": "gg-group-02-id",
      "Cores": [
        {
          "thingArn": "core-02-thing-arn",
          "Connectivity" : [
            {

```

```

        "id": "core-02-connection-id",
        "hostAddress": "core-02-address",
        "portNumber": core-02-port,
        "metadata": "core-02-connection-1-description"
    }
],
"CAs": [
    "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
    "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
    "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
]
}
}
}

```

Note

Un gruppo Greengrass deve definire esattamente un core Greengrass. Qualsiasi risposta dal AWS IoT Greengrass servizio che contiene un elenco di core Greengrass contiene solo un core Greengrass.

Se cURL è installato, è possibile testare la richiesta di discovery. Per esempio:

```

$ curl --cert 1a23bc4d56.cert.pem --key 1a23bc4d56.private.key https://greengrass-ats.iot.us-west-2.amazonaws.com:8443/greengrass/discover/thing/MyDevice
{"GGGroups":[{"GGGroupId":"1234a5b6-78cd-901e-2fgh-3i45j6k1789","Cores":[{"thingArn":"arn:aws:iot:us-west-2:123456789012:thing/MyFirstGroup_Core","Connectivity":[{"Id":"AUTOIP_192.168.1.4_1","HostAddress":"192.168.1.5","PortNumber":8883,"Metadata":""}]}],"CAs":["-----BEGIN CERTIFICATE-----\n cert-contents \n-----END CERTIFICATE-----\n"]}]}

```

Sicurezza in AWS IoT Greengrass

La sicurezza del cloud AWS è la massima priorità. In qualità di AWS cliente, puoi beneficiare di un data center e di un'architettura di rete progettati per soddisfare i requisiti delle organizzazioni più sensibili alla sicurezza.

La sicurezza è una responsabilità condivisa tra AWS e te. Il [modello di responsabilità condivisa](#) descrive questo aspetto come sicurezza del cloud e sicurezza nel cloud:

- **Sicurezza del cloud:** AWS è responsabile della protezione dell'infrastruttura che gestisce AWS i servizi in Cloud AWS. AWS fornisce inoltre servizi che è possibile utilizzare in modo sicuro. I revisori esterni testano e verificano regolarmente l'efficacia della nostra sicurezza nell'ambito dei [AWS Programmi di AWS conformità dei Programmi di conformità](#) dei di . Per ulteriori informazioni sui programmi di conformità applicabili AWS IoT Greengrass, consulta [AWS Servizi nell'ambito del programma di conformitàAWS](#) .
- **Sicurezza nel cloud:** la tua responsabilità è determinata dal servizio AWS che utilizzi. L'utente è anche responsabile di altri fattori, tra cui la riservatezza dei dati, i requisiti dell'azienda e le leggi e le normative applicabili.

Durante l'utilizzo AWS IoT Greengrass, l'utente è anche responsabile della protezione dei dispositivi, della connessione di rete locale e delle chiavi private.

Questa documentazione aiuta a capire come applicare il modello di responsabilità condivisa durante l'utilizzo AWS IoT Greengrass. I seguenti argomenti mostrano come configurare per AWS IoT Greengrass soddisfare gli obiettivi di sicurezza e conformità. Imparerai anche a utilizzare altri AWS servizi che ti aiutano a monitorare e proteggere AWS IoT Greengrass le tue risorse.

Argomenti

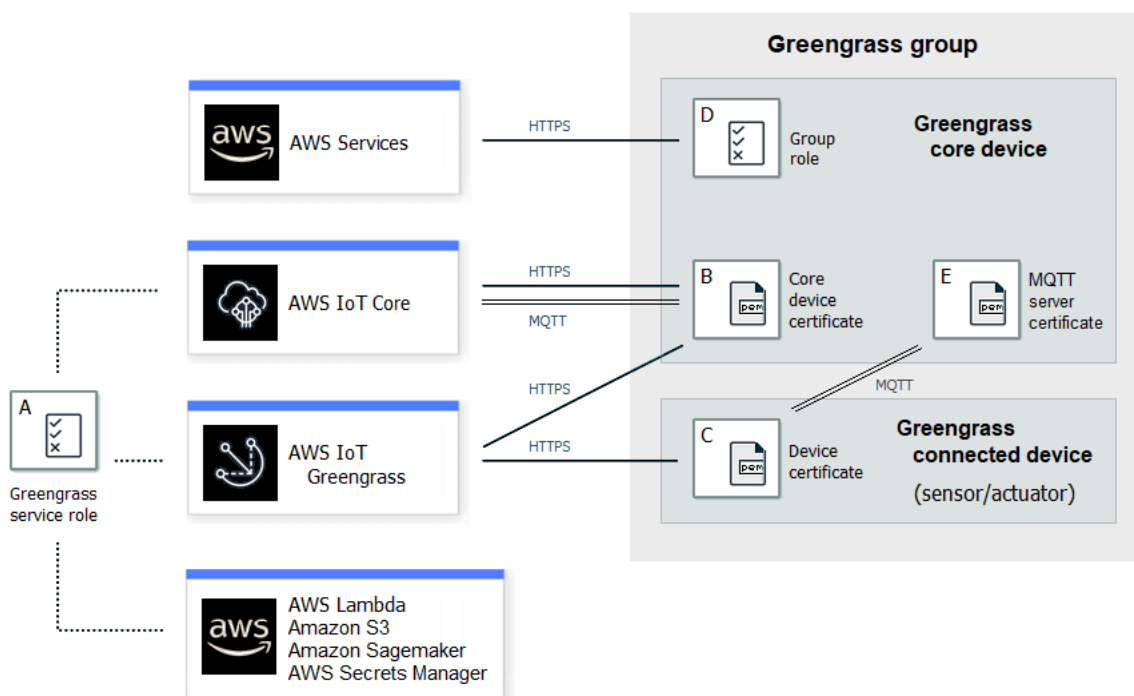
- [Panoramica della AWS IoT Greengrass sicurezza](#)
- [Protezione dei dati in AWS IoT Greengrass](#)
- [Autenticazione e autorizzazione del dispositivo per AWS IoT Greengrass](#)
- [Gestione delle identità e degli accessi per AWS IoT Greengrass](#)
- [Convalida della conformità per AWS IoT Greengrass](#)
- [Resilienza in AWS IoT Greengrass](#)
- [Sicurezza dell'infrastruttura in AWS IoT Greengrass](#)

- [Analisi della configurazione e delle vulnerabilità in AWS IoT Greengrass](#)
- [AWS IoT Greengrass e endpoint VPC di interfaccia \(AWS PrivateLink\)](#)
- [Best practice di sicurezza per AWS IoT Greengrass](#)

Panoramica della AWS IoT Greengrass sicurezza

AWS IoT Greengrass utilizza certificati, AWS IoT policy e ruoli IAM X.509 per proteggere le applicazioni eseguite sui dispositivi nell'ambiente Greengrass locale.

Il diagramma seguente mostra i componenti del modello di sicurezza: AWS IoT Greengrass



A - Ruolo del servizio Greengrass

Un ruolo IAM creato dal cliente che si assume AWS IoT Greengrass quando si accede alle AWS risorse da AWS IoT Core e altri servizi. AWS Lambda AWS Per ulteriori informazioni, consulta [the section called “Ruolo del servizio Greengrass”](#).

B - Certificato dispositivo core

Un certificato X.509 utilizzato per autenticare un core Greengrass con e. AWS IoT Core AWS IoT Greengrass Per ulteriori informazioni, consulta [the section called “Autenticazione e autorizzazione del dispositivo”](#).

C - Certificato dispositivo

Un certificato X.509 utilizzato per autenticare un dispositivo client, noto anche come dispositivo connesso, con e. AWS IoT Core AWS IoT Greengrass Per ulteriori informazioni, consulta [the section called “Autenticazione e autorizzazione del dispositivo”](#).

D - Ruolo gruppo

Un ruolo IAM creato dal cliente assunto da AWS IoT Greengrass quando si chiamano AWS servizi da un core Greengrass.

Questo ruolo viene utilizzato per specificare le autorizzazioni di accesso necessarie alle funzioni e ai connettori Lambda definiti dall'utente per accedere ai AWS servizi, come DynamoDB. Lo usi anche per consentire di AWS IoT Greengrass esportare i flussi dello stream manager nei servizi e scriverli AWS nei registri. CloudWatch Per ulteriori informazioni, consulta [the section called “Ruolo del gruppo Greengrass”](#).

Note

AWS IoT Greengrass non utilizza il ruolo di esecuzione Lambda specificato nella AWS Lambda versione cloud di una funzione Lambda.

Certificato server e-MQTT

Il certificato utilizzato per l'autenticazione reciproca Transport Layer Security (TLS) tra un dispositivo core Greengrass e i dispositivi client del gruppo Greengrass. Il certificato è firmato dal certificato CA di gruppo, che è memorizzato in. Cloud AWS

Flusso di lavoro di connessione del dispositivo

Questa sezione descrive come i dispositivi client si connettono al AWS IoT Greengrass servizio e ai dispositivi core Greengrass. I dispositivi client sono AWS IoT Core dispositivi registrati che fanno parte dello stesso gruppo Greengrass del dispositivo principale.

- Un dispositivo core Greengrass utilizza il certificato del dispositivo, la chiave privata e il certificato CA AWS IoT Core principale per connettersi al AWS IoT Greengrass servizio. Sul dispositivo principale, l'oggetto `crypto` nel [file di configurazione](#) specifica il percorso del file per questi elementi.

- Il dispositivo core Greengrass scarica le informazioni sull'appartenenza al gruppo dal servizio AWS IoT Greengrass .
- Quando viene effettuata una distribuzione sul dispositivo core di Greengrass, Device Certificate Manager (DCM) si occupa della gestione dei certificati del server locale per il dispositivo core Greengrass.
- Un dispositivo client si connette al AWS IoT Greengrass servizio utilizzando il certificato del dispositivo, la chiave privata e il certificato CA AWS IoT Core principale. Dopo aver effettuato la connessione, il dispositivo client utilizza il Greengrass Discovery Service per trovare l'indirizzo IP del suo dispositivo principale Greengrass. Il dispositivo client scarica anche il certificato CA di gruppo, che viene utilizzato per l'autenticazione reciproca TLS con il dispositivo principale Greengrass.
- Un dispositivo client tenta di connettersi al dispositivo principale Greengrass, passando il certificato del dispositivo e l'ID client. Se l'ID client corrisponde al nome dell'oggetto del dispositivo client e il certificato è valido (parte del gruppo Greengrass), viene stabilita la connessione. In caso contrario, la connessione viene terminata.

La AWS IoT politica per i dispositivi client deve concedere `greengrass:Discover` autorizzazione per consentire ai dispositivi client di scoprire le informazioni di connettività relative al core. Per ulteriori informazioni sull'istruzione di policy, consulta [the section called “Autorizzazione alla scoperta”](#).

Configurazione della sicurezza AWS IoT Greengrass

Per configurare la sicurezza dell'applicazione Greengrass:

1. Crea AWS IoT Core qualcosa per il tuo dispositivo principale Greengrass.
2. Generare una coppia di chiavi e un certificato di dispositivo per il dispositivo core Greengrass.
3. Creare e collegare una [policy AWS IoT](#) al certificato del dispositivo. Il certificato e la policy consentono al dispositivo principale di Greengrass di accedere ai servizi AWS IoT Core e AWS IoT Greengrass ai servizi. Per ulteriori informazioni, consulta [AWS IoT Politica minima per il dispositivo principale](#).

Note

L'uso di [thing policy variables](#) (`iot:Connection.Thing.*`) nella AWS IoT policy per un dispositivo principale non è supportato. Il core utilizza lo stesso certificato

del dispositivo per effettuare [più connessioni](#) AWS IoT Core , ma l'ID client in una connessione potrebbe non corrispondere esattamente al nome dell'oggetto principale.

4. Creare un [ruolo di servizio Greengrass](#). Questo ruolo IAM AWS IoT Greengrass autorizza l'accesso alle risorse di altri AWS servizi per tuo conto. Ciò consente di AWS IoT Greengrass eseguire attività essenziali, come il recupero delle AWS Lambda funzioni e la gestione delle ombre dei dispositivi.

Puoi utilizzare lo stesso ruolo di servizio su Regione AWS s, ma deve essere associato al tuo Account AWS in ogni Regione AWS luogo in cui lo utilizzi. AWS IoT Greengrass

5. (Facoltativo) Creare un [ruolo del gruppo Greengrass](#). Questo ruolo IAM concede l'autorizzazione alle funzioni e ai connettori Lambda in esecuzione su un core Greengrass per chiamare i servizi. AWS Ad esempio, il [connettore Kinesis Firehose](#) richiede l'autorizzazione a scrivere record in un flusso di distribuzione di Amazon Data Firehose.

È possibile associare un solo ruolo a un gruppo Greengrass.

6. Crea AWS IoT Core qualcosa per ogni dispositivo che si connette al tuo core Greengrass.

Note

Puoi anche usare AWS IoT Core oggetti e certificati esistenti.

7. Crea certificati, coppie di chiavi e AWS IoT policy per ogni dispositivo che si connette al tuo core Greengrass.

AWS IoT Greengrass principi di sicurezza fondamentali

Il core Greengrass utilizza i seguenti principi di sicurezza: AWS IoT client, server MQTT locale e gestore dei segreti locali. La configurazione per questi principal è memorizzata nell'oggetto crypto nel file di configurazione `config.json`. Per ulteriori informazioni, consulta [the section called “AWS IoT Greengrass file di configurazione principale”](#).

Questa configurazione include il percorso della chiave privata utilizzata dal componente principal per l'autenticazione e la crittografia. AWS IoT Greengrass supporta due modi di storage della chiave privata: basato su hardware o basato su file system (impostazione predefinita). Per ulteriori informazioni sulla memorizzazione delle chiavi in moduli di sicurezza hardware, consulta [the section called “Integrazione della sicurezza hardware”](#).

AWS IoT Cliente

Il AWS IoT client (client IoT) gestisce la comunicazione via Internet tra il core Greengrass e. AWS IoT Core AWS IoT Greengrass utilizza certificati X.509 con chiavi pubbliche e private per l'autenticazione reciproca quando stabilisce connessioni TLS per questa comunicazione. Per ulteriori informazioni consulta [Certificati X.509 e AWS IoT Core](#) nella Guida per sviluppatori AWS IoT Core .

Il client IoT supporta certificati e chiavi RSA ed EC. Il percorso del certificato e della chiave privata è specificato per il principal `IoTCertificate` in `config.json`.

Server MQTT

Il server MQTT locale gestisce la comunicazione sulla rete locale tra il core Greengrass e i dispositivi client del gruppo. AWS IoT Greengrass utilizza certificati X.509 con chiavi pubbliche e private per l'autenticazione reciproca quando stabilisce connessioni TLS per questa comunicazione.

Per impostazione predefinita, AWS IoT Greengrass genera automaticamente una chiave privata RSA. Per configurare il core per usare un'altra chiave privata, è necessario fornire il percorso della chiave per il principal `MQTTServerCertificate` in `config.json`. L'utente è responsabile della rotazione di una chiave fornita dal cliente.

Supporto per la chiave privata

	Chiave RSA	Chiave CE
Tipo di chiavi	Supportato	Supportato
Parametri chiave	Lunghezza minima 2048 bit	Curva NIST P-256 o NIST P-384
Formato disco	PKCS#1, PKCS#8	SECG1, PKCS #8
Versione GGC minima	• Usa chiave RSA predefinita: 1.0 • Specifica una chiave RSA: 1.7	• Specifica una chiave EC: 1.9

La configurazione della chiave privata determina processi correlati. Per l'elenco delle suite di cifratura supportate dal core Greengrass supporta come server, consulta [the section called “Supporto TLS per le suite di cifratura”](#).

Se nessuna chiave privata è specificata (impostazione predefinita)

- AWS IoT Greengrass ruota il tasto in base alle impostazioni di rotazione.
- Il core genera una chiave RSA che viene utilizzata per generare il certificato.
- Il certificato del server MQTT ha una chiave pubblica RSA e una firma RSA SHA-256.

Se viene specificata una chiave privata RSA (richiede GGC v1.7 o successivo)

- Sei responsabile della rotazione della chiave.
- Il core utilizza la chiave specificata per generare il certificato.
- La chiave RSA deve avere una lunghezza minima di 2048 bit.
- Il certificato del server MQTT ha una chiave pubblica RSA e una firma RSA SHA-256.

Se viene specificata una chiave privata EC (richiede GGC v1.9 o successivo)

- Sei responsabile della rotazione della chiave.
- Il core utilizza la chiave specificata per generare il certificato.
- La chiave privata EC deve utilizzare una curva NIST P-256 o NIST P-384.
- Il certificato del server MQTT ha una chiave pubblica EC e una firma RSA SHA-256.

Il certificato del server MQTT presentato dal core ha una firma RSA SHA-256, indipendentemente dal tipo di chiave. Per questo motivo, i client devono supportare la convalida del certificato RSA SHA-256 per stabilire una connessione sicura con il core.

Secrets Manager

Il gestore dei segreti locali gestisce in modo sicuro le copie locali dei segreti creati dall'utente. AWS Secrets Manager Utilizza una chiave privata per proteggere la chiave dati utilizzata per crittografare i segreti. Per ulteriori informazioni, consulta [Distribuzione dei segreti nel core](#).

Per impostazione predefinita, viene utilizzata la chiave privata del client IoT, ma è possibile specificare una chiave privata diversa per il principal SecretsManager in `config.json`. Solo il tipo di chiave RSA è supportato. Per ulteriori informazioni, consulta [the section called “Specificare la chiave privata per la crittografia dei segreti”](#).

Note

Attualmente, AWS IoT Greengrass supporta solo il meccanismo di riempimento [PKCS #1 v1.5](#) per la crittografia e la decrittografia dei segreti locali quando si utilizzano chiavi private basate su hardware. Se stai seguendo le istruzioni fornite dal fornitore per generare manualmente chiavi private basate su hardware, assicurati di scegliere PKCS #1 v1.5. AWS IoT Greengrass non supporta Optimal Asymmetric Encryption Padding (OAEP).

Supporto per la chiave privata

	Chiave RSA	Chiave CE
Tipo di chiavi	Supportato	Non supportato
Parametri chiave	Lunghezza minima 2048 bit	Non applicabile
Formato disco	PKCS#1, PKCS#8	Non applicabile
Versione GGC minima	1,7	Non applicabile

Sottoscrizioni gestite nel flusso di lavoro di messaggistica MQTT

AWS IoT Greengrass utilizza una tabella di sottoscrizione per definire come i messaggi MQTT possono essere scambiati tra dispositivi client, funzioni e connettori in un gruppo Greengrass e con AWS IoT Core o il servizio shadow locale. Ogni sottoscrizione specifica un'origine, una destinazione e un argomento (o oggetto) MQTT su cui inviare o ricevere i messaggi. AWS IoT Greengrass consente l'invio di messaggi da un'origine a una destinazione solo se è definita una sottoscrizione corrispondente.

Una sottoscrizione definisce il flusso dei messaggi solo in una direzione, dall'origine alla destinazione. Per supportare lo scambio di messaggi bidirezionale, è necessario creare due sottoscrizioni, una per ogni direzione.

Supporto TLS per le suite di cifratura

AWS IoT Greengrass utilizza il modello di sicurezza del AWS IoT Core trasporto per crittografare le comunicazioni con il cloud utilizzando suite di [crittografia TLS](#). Inoltre, AWS IoT Greengrass i dati

vengono crittografati quando sono inattivi (nel cloud). Per ulteriori informazioni sulla sicurezza del AWS IoT Core trasporto e sulle suite di crittografia supportate, consulta [Transport security](#) nella AWS IoT Core Developer Guide.

Suite di cifratura supportate per le comunicazioni sulla rete locale

Al contrario AWS IoT Core, il AWS IoT Greengrass core supporta le seguenti suite di crittografia TLS di rete locale per gli algoritmi di firma dei certificati. Tutte queste suite sono supportate quando le chiavi private sono memorizzate nel file system. Un sottoinsieme è supportato quando il core è configurato per l'utilizzo di moduli di sicurezza hardware (HSM). Per ulteriori informazioni, consulta [the section called “Principal di sicurezza”](#) e [the section called “Integrazione della sicurezza hardware”](#). La tabella include anche la versione minima del software Core richiesta per il supporto. AWS IoT Greengrass

	Crittografia	Supporto HSM	Versione GGC minima
TLSv12.	TLS_ECDHE _RSA_WITH _AES_128_CBC_SHA	Supportato	1
	TLS_ECDHE _RSA_WITH _AES_256_CBC_SHA	Supportato	1
	TLS_ECDHE _RSA_WITH _AES_256_GCM_ SHA384	Supportato	1
	TLS_RSA_W ITH_AES_1 28_CBC_SHA	Non supportato	1
	TLS_RSA_C ON_AES_128_GCM_ SHA256	Non supportato	1

	Crittografia	Supporto HSM	Versione GGC minima
TLSv11.	TLS_RSA_W ITH_AES_2 56_CBC_SHA	Non supportato	1
	TLS_RSA_C ON_AES_256_GCM_ SHA384	Non supportato	1
	TLS_ECDHE _ECDSA_CO N_AES_128_GCM_ SHA256	Supportato	1.9
	TLS_ECDHE _ECDSA_CO N_AES_256_GCM_ SHA384	Supportato	1.9
	TLS_ECDHE _RSA_WITH _AES_128_CBC_SHA	Supportato	1
	TLS_ECDHE _RSA_WITH _AES_256_CBC_SHA	Supportato	1
	TLS_RSA_W ITH_AES_1 28_CBC_SHA	Non supportato	1
	TLS_RSA_W ITH_AES_2 56_CBC_SHA	Non supportato	1

	Crittografia	Supporto HSM	Versione GGC minima
TLSv10.	TLS_ECDHE _RSA_WITH _AES_128_CBC_SHA	Supportato	1
	TLS_ECDHE _RSA_WITH _AES_256_CBC_SHA	Supportato	1
	TLS_RSA_W ITH_AES_1 28_CBC_SHA	Non supportato	1
	TLS_RSA_W ITH_AES_2 56_CBC_SHA	Non supportato	1

Protezione dei dati in AWS IoT Greengrass

Il [modello di responsabilità AWS condivisa](#) di si applica alla protezione dei dati in AWS IoT Greengrass. Come descritto in questo modello, AWS è responsabile della protezione dell'infrastruttura globale che gestisce tutti i Cloud AWS. L'utente è responsabile del controllo dei contenuti ospitati su questa infrastruttura. L'utente è inoltre responsabile della configurazione della protezione e delle attività di gestione per i Servizi AWS utilizzati. Per ulteriori informazioni sulla privacy dei dati, vedi le [Domande frequenti sulla privacy dei dati](#). Per informazioni sulla protezione dei dati in Europa, consulta il post del blog relativo al [Modello di responsabilità condivisa AWS e GDPR](#) nel Blog sulla sicurezza AWS .

Ai fini della protezione dei dati, consigliamo di proteggere Account AWS le credenziali e configurare i singoli utenti con AWS IAM Identity Center or AWS Identity and Access Management (IAM). In tal modo, a ogni utente verranno assegnate solo le autorizzazioni necessarie per svolgere i suoi compiti. Ti suggeriamo, inoltre, di proteggere i dati nei seguenti modi:

- Utilizza l'autenticazione a più fattori (MFA) con ogni account.
- Usa SSL/TLS per comunicare con le risorse. AWS È richiesto TLS 1.2 ed è consigliato TLS 1.3.

- Configura l'API e la registrazione delle attività degli utenti con. AWS CloudTrail Per informazioni sull'utilizzo dei CloudTrail percorsi per acquisire AWS le attività, consulta [Lavorare con i CloudTrail percorsi](#) nella Guida per l'AWS CloudTrail utente.
- Utilizza soluzioni di AWS crittografia, insieme a tutti i controlli di sicurezza predefiniti all'interno Servizi AWS.
- Utilizza i servizi di sicurezza gestiti avanzati, come Amazon Macie, che aiutano a individuare e proteggere i dati sensibili archiviati in Amazon S3.
- Se hai bisogno di moduli crittografici convalidati FIPS 140-3 per accedere AWS tramite un'interfaccia a riga di comando o un'API, usa un endpoint FIPS. Per ulteriori informazioni sugli endpoint FIPS disponibili, consulta il [Federal Information Processing Standard \(FIPS\) 140-3](#).

Ti consigliamo di non inserire mai informazioni riservate o sensibili, ad esempio gli indirizzi e-mail dei clienti, nei tag o nei campi di testo in formato libero, ad esempio nel campo Nome. Ciò include quando lavori AWS IoT Greengrass o Servizi AWS utilizzi la console, l'API o. AWS CLI AWS SDKs I dati inseriti nei tag o nei campi di testo in formato libero utilizzati per i nomi possono essere utilizzati per i la fatturazione o i log di diagnostica. Quando fornisci un URL a un server esterno, ti suggeriamo vivamente di non includere informazioni sulle credenziali nell'URL per convalidare la tua richiesta al server.

Per ulteriori informazioni sulla protezione delle informazioni sensibili in AWS IoT Greengrass, consulta [the section called “Non registrare informazioni riservate”](#).

Per ulteriori informazioni sulla protezione dei dati, consulta il post del blog [AWS Modello di responsabilità condivisa e GDPR](#) su AWS Security Blog.

Argomenti

- [Crittografia dei dati](#)
- [Integrazione della sicurezza hardware](#)

Crittografia dei dati

AWS IoT Greengrass utilizza la crittografia per proteggere i dati durante il transito (su Internet o rete locale) e quando sono inattivi (archiviati in Cloud AWS).

I dispositivi in un AWS IoT Greengrass ambiente spesso raccolgono dati che vengono inviati ai AWS servizi per un'ulteriore elaborazione. Per ulteriori informazioni sulla crittografia dei dati su altri AWS servizi, consulta la documentazione sulla sicurezza relativa a tale servizio.

Argomenti

- [Crittografia in transito](#)
- [Crittografia a riposo](#)
- [Gestione delle chiavi per il dispositivo Core Greengrass](#)

Crittografia in transito

AWS IoT Greengrass dispone di tre modalità di comunicazione in cui i dati sono in transito:

- [the section called “Dati in transito su Internet”](#). La comunicazione tra un core Greengrass e AWS IoT Greengrass Internet è crittografata.
- [the section called “Dati in transito sulla rete locale”](#). La comunicazione tra un core Greengrass e i dispositivi client su una rete locale è crittografata.
- [the section called “Dati sul dispositivo core”](#). La comunicazione tra i componenti del dispositivo centrale Greengrass non è crittografata.

Dati in transito su Internet

AWS IoT Greengrass utilizza Transport Layer Security (TLS) per crittografare tutte le comunicazioni su Internet. Tutti i dati inviati a vengono inviati tramite una connessione TLS utilizzando i protocolli MQTT o HTTPS, quindi sono sicuri per impostazione predefinita. Cloud AWS AWS IoT Greengrass utilizza il modello di sicurezza del AWS IoT trasporto. Per ulteriori informazioni, consulta l'argomento relativo alla [sicurezza del trasporto](#) nella Guida per gli sviluppatori AWS IoT Core .

Dati in transito sulla rete locale

AWS IoT Greengrass utilizza TLS per crittografare tutte le comunicazioni sulla rete locale tra il core di Greengrass e i dispositivi client. Per ulteriori informazioni, vedere [Supported Cipher Suite per la comunicazione di rete locale](#).

È responsabilità dell'utente proteggere la rete locale e le chiavi private.

Per i dispositivi core Greengrass, è responsabilità dell'utente:

- Tenere aggiornato il kernel con le patch di sicurezza più recenti.
- Mantenere le librerie di sistema aggiornate con le patch di sicurezza più recenti.
- Proteggere le chiavi private. Per ulteriori informazioni, consulta [the section called “Gestione delle chiavi”](#).

Per i dispositivi client, è tua responsabilità:

- Mantenere aggiornato lo stack TLS.
- Proteggere le chiavi private.

Dati sul dispositivo core

AWS IoT Greengrass non crittografa i dati scambiati localmente sul dispositivo principale Greengrass perché i dati non escono dal dispositivo. Ciò include la comunicazione tra funzioni Lambda definite dall'utente, connettori, AWS IoT Greengrass Core SDK e componenti di sistema, come lo stream manager.

Crittografia a riposo

AWS IoT Greengrass archivia i tuoi dati:

- [the section called “Dati inattivi in Cloud AWS”](#). Questi dati sono criptati.
- [the section called “Dati inattivi sul core Greengrass”](#). Questi dati non sono crittografati (tranne le copie locali dei tuoi segreti).

Dati inattivi in Cloud AWS

AWS IoT Greengrass crittografa i dati dei clienti archiviati in Cloud AWS. Questi dati sono protetti mediante AWS KMS chiavi gestite da AWS IoT Greengrass.

Dati inattivi sul core Greengrass

AWS IoT Greengrass si affida alle autorizzazioni di file Unix e alla crittografia dell'intero disco (se abilitata) per proteggere i dati archiviati integralmente. È tua responsabilità proteggere il file system e il dispositivo.

Tuttavia, crittografa AWS IoT Greengrass le copie locali dei tuoi segreti recuperati da AWS Secrets Manager. Per ulteriori informazioni, consulta [the section called “Crittografia dei segreti”](#).

Gestione delle chiavi per il dispositivo Core Greengrass

È responsabilità del cliente garantire l'archiviazione sicura delle chiavi crittografiche (pubbliche e private) sul dispositivo core Greengrass. AWS IoT Greengrass utilizza chiavi pubbliche e private per i seguenti scenari:

- La chiave client IoT viene utilizzata con il certificato IoT per autenticare l'handshake TLS (Transport Layer Security) quando un core Greengrass si connette a AWS IoT Core. Per ulteriori informazioni, consulta [the section called “Autenticazione e autorizzazione del dispositivo”](#).

Note

La chiave e il certificato sono anche indicati come chiave privata principale e il certificato del dispositivo core.

- La chiave del server MQTT viene utilizzata come certificato del server MQTT per autenticare le connessioni TLS tra i dispositivi core e client. Per ulteriori informazioni, consulta [the section called “Autenticazione e autorizzazione del dispositivo”](#).
- Il gestore dei segreti locali utilizza anche la chiave client IoT per proteggere la chiave dati utilizzata per crittografare i segreti locali, ma è possibile fornire la propria chiave privata. Per ulteriori informazioni, consulta [the section called “Crittografia dei segreti”](#).

Un core Greengrass supporta l'archiviazione di chiavi private utilizzando le autorizzazioni del file system, [i moduli di sicurezza hardware](#), o entrambi. Se si utilizzano chiavi private basate su file system, si è responsabili della loro archiviazione sicura sul dispositivo core.

Su un nucleo di Greengrass, la posizione delle chiavi private è specificata nella sezione `crypto` del file `config.json`. Se si configura il core per utilizzare una chiave fornita dal cliente per il certificato server MQTT, è responsabilità dell'utente ruotare la chiave. Per ulteriori informazioni, consulta [the section called “Principal di sicurezza”](#).

Per i dispositivi client, è tua responsabilità mantenere aggiornato lo stack TLS e proteggere le chiavi private. Le chiavi private vengono utilizzate con i certificati dei dispositivi per autenticare le connessioni TLS con il servizio. AWS IoT Greengrass

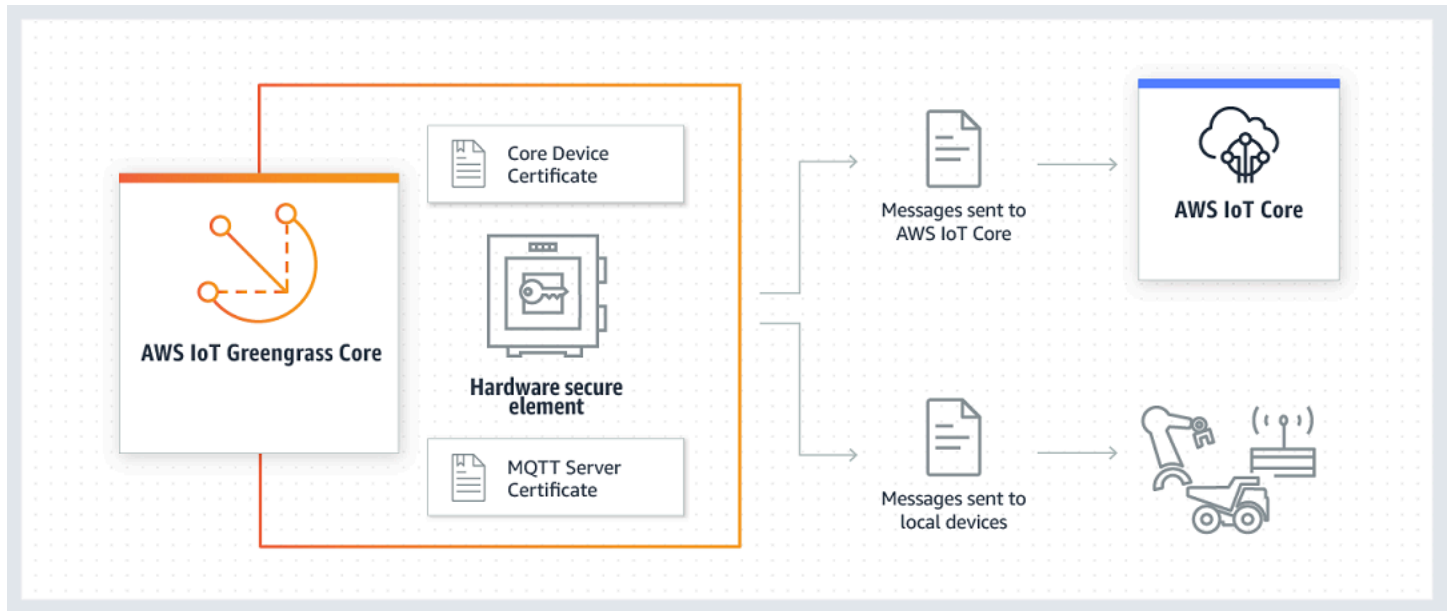
Integrazione della sicurezza hardware

Questa funzionalità è disponibile per AWS IoT Greengrass Core v1.7 e versioni successive.

AWS IoT Greengrass supporta l'uso di moduli di sicurezza hardware (HSM) tramite [l'interfaccia PKCS #11](#) per l'archiviazione e l'offload sicuri delle chiavi private. In questo modo, si impedisce che le chiavi vengano esposte o duplicate nel software. Le chiavi private possono essere archiviate in modo sicuro su moduli hardware HSMs, come Trusted Platform Modules (TPM) o altri elementi crittografici.

[Cerca i dispositivi idonei per questa funzionalità nel Catalogo dei AWS Partner dispositivi.](#)

Il diagramma seguente mostra l'architettura di sicurezza hardware per un AWS IoT Greengrass core.



In un'installazione standard, AWS IoT Greengrass utilizza due chiavi private. Una chiave viene utilizzata dal componente AWS IoT client (client IoT) durante l'handshake Transport Layer Security (TLS) quando si connette un core Greengrass. AWS IoT Core Questa chiave viene anche denominata chiave privata core. L'altra chiave viene utilizzata dal server MQTT locale, che consente ai dispositivi Greengrass di comunicare con il core Greengrass. Se desideri utilizzare la sicurezza hardware per entrambi i componenti, puoi utilizzare una chiave privata condivisa o chiavi private separate. Per ulteriori informazioni, consulta [the section called “ Prassi di effettuazione del provisioning”](#).

Note

In un'installazione standard, il Secrets Manager locale utilizza anche la chiave client IoT per il processo di crittografia, ma puoi utilizzare la tua chiave privata. Deve essere una chiave RSA con una lunghezza minima di 2048 bit. Per ulteriori informazioni, consulta [the section called “Specificare la chiave privata per la crittografia dei segreti”](#).

Requisiti

Prima di configurare la sicurezza hardware di un core Greengrass, devi disporre di quanto segue:

- Un modulo di sicurezza hardware (HSM) che supporta la configurazione della chiave privata target per il client IoT, il server MQTT locale e i componenti Secrets Manager locali. La configurazione

può includere una, due o tre chiavi private basate su hardware, a seconda che si configurino i componenti per condividere le chiavi. Per ulteriori informazioni sul supporto per la chiave privata, consulta [the section called “Principal di sicurezza”](#).

- Per le chiavi RSA: una dimensione di chiave RSA-2048 (o superiore) e uno schema di firma [PKCS#1 v1.5](#).
- Per le chiavi EC: una curva NIST P-256 o NIST P-384.

 Note

[Cerca i dispositivi idonei per questa funzionalità nel Catalogo dei AWS Partner dispositivi.](#)

- Una libreria del provider PKCS#11 che può essere caricata in fase di runtime (tramite libdl) e che offre funzioni [PKCS#11](#).
- Il modulo hardware deve essere risolvibile mediante l'etichetta dello slot, come definito nella specifica PKCS#11.
- La chiave privata deve essere generata e caricata nell'HSM con gli strumenti di provisioning del fornitore.
- La chiave privata deve essere risolvibile mediante l'etichetta dell'oggetto.
- Il certificato del dispositivo core. Si tratta di un certificato del client IoT corrispondente alla chiave privata.
- Se si utilizza l'agente di aggiornamento OTA Greengrass, è necessario installare la libreria wrapper [OpenSSL libp11 PKCS #11](#). Per ulteriori informazioni, consulta [the section called “Configurazione degli aggiornamenti OTA”](#).

Inoltre, assicurati che siano soddisfatte le seguenti condizioni:

- I certificati client IoT associati alla chiave privata vengono registrati AWS IoT e attivati. Puoi verificarlo nella AWS IoT console sotto Gestisci, espandere Tutti i dispositivi, scegliere Cose e scegliere la scheda Certificati per l'elemento principale.
- Il software AWS IoT Greengrass Core v1.7 o successivo viene installato sul dispositivo principale, come descritto nel [Modulo 2](#) del tutorial Getting Started. È richiesta la versione 1.9 o successiva per utilizzare una chiave EC per il server MQTT.
- I certificati sono collegati al core Greengrass. È possibile verificarlo dalla pagina Gestisci relativa all'elemento principale della AWS IoT console.

 Note

Attualmente, AWS IoT Greengrass non supporta il caricamento del certificato CA o del certificato client IoT direttamente dall'HSM. I certificati devono essere caricati come file di testo normale nel file system in un percorso che sia leggibile da Greengrass.

Configurazione di sicurezza hardware per un core AWS IoT Greengrass

La sicurezza hardware viene configurata nel file di configurazione di Greengrass. Si tratta del file [config.json](#), disponibile nella directory `/greengrass-root/config`.

 Note

Per visualizzare il processo di impostazione della configurazione HSM tramite un'implementazione software, consulta [the section called “Modulo 7: simulazione dell'integrazione di sicurezza hardware”](#).

 Important

La configurazione simulata nell'esempio non fornisce alcun vantaggio dal punto di vista della sicurezza. Lo scopo di questo esempio è quello di fornire nozioni di base sulla specifica PKCS#11 e di eseguire il test iniziale del software, se prevedi di utilizzare in futuro un HSM basato sull'hardware.

Per configurare la sicurezza hardware in AWS IoT Greengrass, è necessario modificare l'oggetto `crypto` in `config.json`.

Quando utilizzi la sicurezza hardware, l'oggetto `crypto` viene utilizzato per specificare i percorsi dei certificati, delle chiavi private e degli asset della libreria del provider PKCS#11 nel core, come mostrato nell'esempio seguente.

```
"crypto": {  
  "PKCS11" : {  
    "OpenSSLEngine" : "/path-to-p11-openssl-engine",  
    "P11Provider" : "/path-to-pkcs11-provider-so",  
    "slotLabel" : "crypto-token-name",  
  }  
}
```



```

"slotUserPin" : "crypto-token-user-pin"
},
"principals" : {
  "IoTCertificate" : {
    "privateKeyPath" : "pkcs11:object=core-private-key-label;type=private",
    "certificatePath" : "file:///path-to-core-device-certificate"
  },
  "MQTTServerCertificate" : {
    "privateKeyPath" : "pkcs11:object=server-private-key-label;type=private"
  },
  "SecretsManager" : {
    "privateKeyPath": "pkcs11:object=core-private-key-label;type=private"
  }
},
"caPath" : "file:///path-to-root-ca"

```

L'oggetto crypto include le seguenti proprietà:

Campo	Descrizione	Note
caPath	Il percorso assoluto verso la CA AWS IoT principale.	Deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code> .
 Note Assicurati che gli endpoint corrispon dano al tipo di certifica to.		
PKCS11		
OpenSSLEngine	Facoltativo. Il percorso assoluto del file .so del motore OpenSSL per abilitare il supporto PKCS#11 su OpenSSL.	Deve essere un percorso di un file nel file system. Questa proprietà è obbligatorio se si utilizza l'agente di aggiornamento OTA

Campo	Descrizione	Note
		Greengrass con sicurezza hardware. Per ulteriori informazioni, consulta the section called “Configurazione degli aggiornamenti OTA” .
P11Provider	Il percorso assoluto della libreria libdl-loadable dell'implementazione PKCS#11.	Deve essere un percorso di un file nel file system.
slotLabel	L'etichetta dello slot utilizzata per identificare il modulo hardware.	Deve essere conforme alle specifiche dell'etichetta PKCS#11.
slotUserPin	Il PIN utente utilizzato per autenticare il core Greengrass nel modulo.	Deve disporre delle autorizzazioni sufficienti a eseguire C_Sign con le chiavi private configurate.
principals		
IoTCertificate	Il certificato e la chiave privata che il core utilizza per effettuare le richieste a AWS IoT.	
IoTCertificate .privateKeyPath	Il percorso della chiave privata del core.	Per lo storage di file system, deve essere un URI di file nel formato: <i>file:///absolute/path/to/file</i> . Per lo storage HSM, deve essere un percorso PKCS#11 RFC 7512 che specifica l'etichetta dell'oggetto.

Campo	Descrizione	Note
IoTCertificate .certificatePath	Il percorso assoluto al certificato del dispositivo core.	Deve essere un URI di file nel formato: <i>file:///absolute/path/to/file</i> .
MQTTServerCertificate	Facoltativo. La chiave privata che il core utilizza insieme al certificato per fungere da gateway o server MQTT.	
MQTTServerCertificate .privateKeyPath	Il percorso della chiave privata del server MQTT locale.	Utilizza questo valore per specificare la tua chiave privata del server MQTT locale. Per lo storage di file system, deve essere un URI di file nel formato: <i>file:///absolute/path/to/file</i> . Per lo storage HSM, deve essere un percorso PKCS#11 RFC 7512 che specifica l'etichetta dell'oggetto. Se questa proprietà viene omessa, AWS IoT Greengrass ruota la chiave in base alle impostazioni di rotazione . Se viene specificata, il cliente sarà responsabile della rotazione della chiave.
SecretsManager	La chiave privata che protegge la chiave di dati utilizzata per la crittografia. Per ulteriori informazioni, consulta Distribuzione dei segreti nel core .	

Campo	Descrizione	Note
SecretsManager.privateKeyPath	Il percorso della chiave privata del Secrets Manager locale.	Solo una chiave RSA è supportata. Per lo storage di file system, deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code>. Per lo storage HSM, deve essere un percorso PKCS#11 RFC 7512 che specifica l'etichetta dell'oggetto. La chiave privata deve essere generata utilizzando il meccanismo di padding PKCS#1 v1.5.

Campo	Descrizione	Note
caPath	Il percorso assoluto verso la CA AWS IoT principale.	Deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code>.  Note Assicurati che gli endpoint corrispondano al tipo di certificato.

PKCS11

Campo	Descrizione	Note
OpenSSLEngine	Facoltativo. Il percorso assoluto del file .so del motore OpenSSL per abilitare il supporto PKCS#11 su OpenSSL.	Deve essere un percorso di un file nel file system. Questa proprietà è obbligatorio se si utilizza l'agente di aggiornamento OTA Greengrass con sicurezza hardware. Per ulteriori informazioni, consulta the section called “Configurazione degli aggiornamenti OTA” .
P11Provider	Il percorso assoluto della libreria libdl-loadable dell'implementazione PKCS#11.	Deve essere un percorso di un file nel file system.
slotLabel	L'etichetta dello slot utilizzata per identificare il modulo hardware.	Deve essere conforme alle specifiche dell'etichetta PKCS#11.
slotUserPin	Il PIN utente utilizzato per autenticare il core Greengrass nel modulo.	Deve disporre delle autorizzazioni sufficienti a eseguire C_Sign con le chiavi private configurate.
principals		
IoTCertificate	Il certificato e la chiave privata che il core utilizza per effettuare le richieste a AWS IoT.	

Campo	Descrizione	Note
IoTCertificate .privateKeyPath	Il percorso della chiave privata del core.	Per lo storage di file system, deve essere un URI di file nel formato: <i>file:///absolute/path/to/file</i> . Per lo storage HSM, deve essere un percorso PKCS#11 RFC 7512 che specifica l'etichetta dell'oggetto.
IoTCertificate .certificatePath	Il percorso assoluto al certificato del dispositivo core.	Deve essere un URI di file nel formato: <i>file:///absolute/path/to/file</i> .
MQTTServerCertificate	Facoltativo. La chiave privata che il core utilizza insieme al certificato per fungere da gateway o server MQTT.	

Campo	Descrizione	Note
MQTTServerCertificate.privateKeyPath	Il percorso della chiave privata del server MQTT locale.	Utilizza questo valore per specificare la tua chiave privata del server MQTT locale. Per lo storage di file system, deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code>. Per lo storage HSM, deve essere un percorso PKCS#11 RFC 7512 che specifica l'etichetta dell'oggetto. Se questa proprietà viene omessa, AWS IoT Greengrass ruota la chiave in base alle impostazioni di rotazione. Se viene specificata, il cliente sarà responsabile della rotazione della chiave.
SecretsManager	La chiave privata che protegge la chiave di dati utilizzata per la crittografia. Per ulteriori informazioni, consulta Distribuzione dei segreti nel core .	

Campo	Descrizione	Note
SecretsManager.privateKeyPath	Il percorso della chiave privata del Secrets Manager locale.	Solo una chiave RSA è supportata. Per lo storage di file system, deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code> . Per lo storage HSM, deve essere un percorso PKCS#11 RFC 7512 che specifica l'etichetta dell'oggetto. La chiave privata deve essere generata utilizzando il meccanismo di padding PKCS#1 v1.5.

Campo	Descrizione	Note
caPath	Il percorso assoluto verso la CA AWS IoT principale.	Deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code> .  Note Assicurati che gli endpoint corrispondano al tipo di certificato.

PKCS11

Campo	Descrizione	Note
OpenSSLEngine	Facoltativo. Il percorso assoluto del file .so del motore OpenSSL per abilitare il supporto PKCS#11 su OpenSSL.	Deve essere un percorso di un file nel file system. Questa proprietà è obbligatoria se si utilizza l'agente di aggiornamento OTA Greengrass con sicurezza hardware. Per ulteriori informazioni, consulta the section called “Configurazione degli aggiornamenti OTA” .
P11Provider	Il percorso assoluto della libreria libdl-loadable dell'implementazione PKCS#11.	Deve essere un percorso di un file nel file system.
slotLabel	L'etichetta dello slot utilizzata per identificare il modulo hardware.	Deve essere conforme alle specifiche dell'etichetta PKCS#11.
slotUserPin	Il PIN utente utilizzato per autenticare il core Greengrass nel modulo.	Deve disporre delle autorizzazioni sufficienti a eseguire C_Sign con le chiavi private configurate.
principals		
IoTCertificate	Il certificato e la chiave privata che il core utilizza per effettuare le richieste a AWS IoT.	

Campo	Descrizione	Note
IoTCertificate .privateKeyPath	Il percorso della chiave privata del core.	Per lo storage di file system, deve essere un URI di file nel formato: <i>file:///absolute/path/to/file</i> . Per lo storage HSM, deve essere un percorso PKCS#11 RFC 7512 che specifica l'etichetta dell'oggetto.
IoTCertificate .certificatePath	Il percorso assoluto al certificato del dispositivo core.	Deve essere un URI di file nel formato: <i>file:///absolute/path/to/file</i> .
MQTTServerCertificate	Facoltativo. La chiave privata che il core utilizza insieme al certificato per fungere da gateway o server MQTT.	

Campo	Descrizione	Note
MQTTServerCertificate.privateKeyPath	Il percorso della chiave privata del server MQTT locale.	Utilizza questo valore per specificare la tua chiave privata del server MQTT locale. Per lo storage di file system, deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code>. Per lo storage HSM, deve essere un percorso PKCS#11 RFC 7512 che specifica l'etichetta dell'oggetto. Se questa proprietà viene omessa, AWS IoT Greengrass ruota la chiave in base alle impostazioni di rotazione. Se viene specificata, il cliente sarà responsabile della rotazione della chiave.
SecretsManager	La chiave privata che protegge la chiave di dati utilizzata per la crittografia. Per ulteriori informazioni, consulta Distribuzione dei segreti nel core .	

Campo	Descrizione	Note
SecretsManager .privateKeyPath	Il percorso della chiave privata del Secrets Manager locale.	Solo una chiave RSA è supportata. Per lo storage di file system, deve essere un URI di file nel formato: <code>file:///absolute/path/to/file</code>. Per lo storage HSM, deve essere un percorso PKCS#11 RFC 7512 che specifica l'etichetta dell'oggetto. La chiave privata deve essere generata utilizzando il meccanismo di padding PKCS#1 v1.5.

Pratiche di provisioning per la sicurezza dell'hardware AWS IoT Greengrass

Di seguito sono elencate le prassi di provisioning correlate alla sicurezza e alle prestazioni.

Sicurezza

- Generare chiavi private direttamente nell'HSM tramite il generatore di numeri casuali hardware interno.

Note

Se configurate le chiavi private da utilizzare con questa funzionalità (seguendo le istruzioni fornite dal fornitore dell'hardware), tenete presente che AWS IoT Greengrass attualmente supporta solo il meccanismo di riempimento della versione PKCS1 1.5 per la crittografia e la decrittografia dei segreti locali. AWS IoT Greengrass non supporta Optimal Asymmetric Encryption Padding (OAEP).

- Configurare le chiavi private per impedire l'esportazione.

- Utilizza lo strumento di provisioning fornito dal fornitore dell'hardware per generare una richiesta di firma del certificato (CSR) utilizzando la chiave privata protetta dall'hardware, quindi utilizza la console per generare un certificato client. AWS IoT

 Note

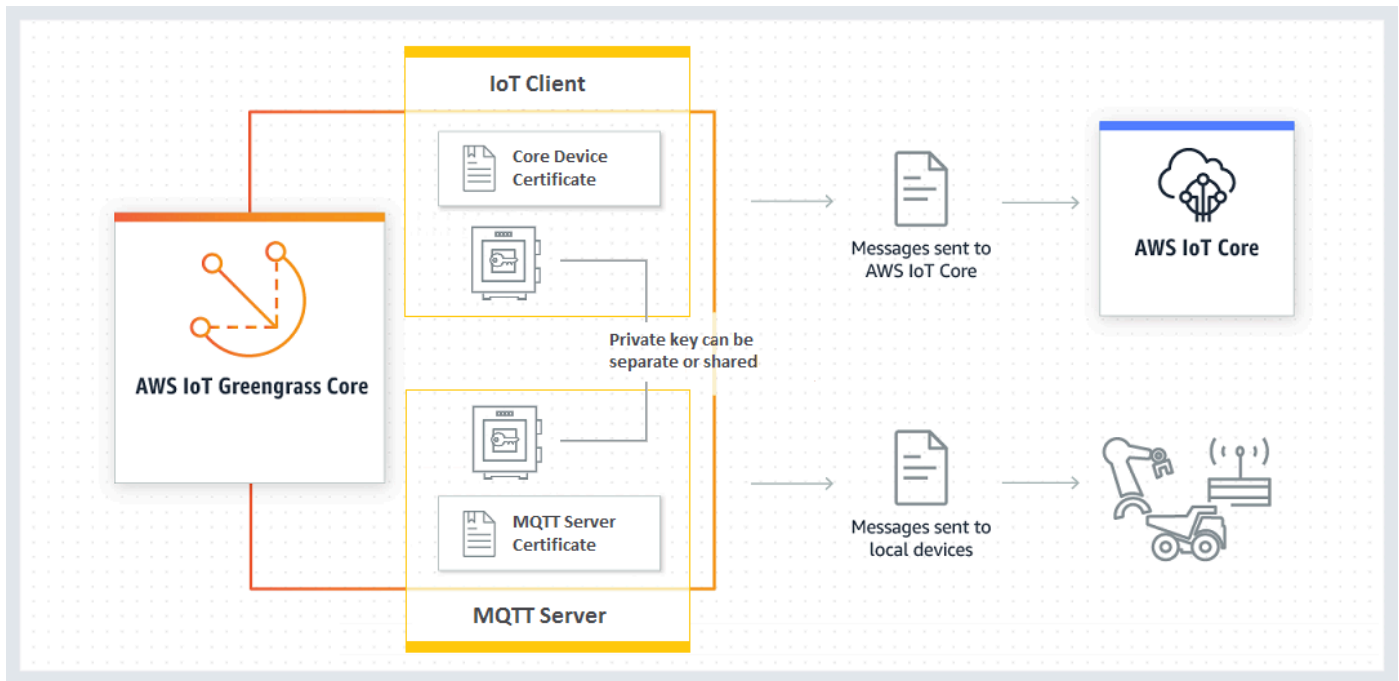
La rotazione delle chiavi non si applica se le chiavi private vengono generate in un HSM.

Prestazioni

Il diagramma seguente mostra il componente client IoT e il server MQTT locale sul AWS IoT Greengrass core. Se desideri utilizzare una configurazione HSM per entrambi i componenti, puoi utilizzare la stessa chiave privata o chiavi private separate. Se usi chiavi separate, devono essere memorizzate nello stesso slot.

 Note

AWS IoT Greengrass non impone alcun limite al numero di chiavi archiviate sull'HSM, quindi è possibile archiviare le chiavi private per il client IoT, il server MQTT e i componenti del gestore dei segreti. Tuttavia, alcuni fornitori HSM potrebbero imporre limiti al numero di chiavi che è possibile memorizzare in uno slot.



In generale, la chiave client IoT non viene utilizzata molto frequentemente perché il software AWS IoT Greengrass Core mantiene connessioni durature al cloud. Tuttavia, la chiave del server MQTT viene utilizzata ogni volta che un dispositivo Greengrass si connette al core. Queste interazioni incidono direttamente sulle prestazioni.

Quando la chiave del server MQTT è memorizzata nell'HSM, la velocità di connessione dei dispositivi dipende dal numero di operazioni di firma RSA al secondo che l'HSM è in grado di eseguire. Ad esempio, se l'HSM impiega 300 millisecondi per eseguire una firma RSASSA-PKCS1 -v1.5 su una chiave privata RSA-2048, solo tre dispositivi possono connettersi al core Greengrass al secondo. Dopo avere effettuato le connessioni, l'HSM non viene più utilizzato e vengono applicate le [quote per AWS IoT Greengrass](#) standard.

Per ridurre i colli di bottiglia delle prestazioni, puoi archiviare la chiave privata del server MQTT nel file system anziché nell'HSM. Con questa configurazione, il server MQTT si comporterà come se la sicurezza hardware non fosse abilitata.

AWS IoT Greengrass supporta più configurazioni di archiviazione delle chiavi per il client IoT e i componenti del server MQTT, in modo da poter ottimizzare i requisiti di sicurezza e prestazioni. La tabella seguente include le configurazioni di esempio.

Configurazione	Chiave IoT	Chiave MQTT	Prestazioni
Chiave condivisa HSM	HSM: chiave A	HSM: chiave A	Limitata dall'HSM o dalla CPU
Chiavi separate HSM	HSM: chiave A	HSM: chiave B	Limitata dall'HSM o dalla CPU
Solo HSM per IoT	HSM: chiave A	File system: chiave B	Limitata dalla CPU
Legacy	File system: chiave A	File system: chiave B	Limitata dalla CPU

Per configurare il core Greengrass per l'utilizzo di chiavi basate sul file system per il server MQTT, omettete la `principals.MQTTServerCertificate` sezione da `config.json` (o specificate un percorso basato su file per la chiave se non utilizzate la chiave predefinita generata da). AWS IoT Greengrass L'oggetto `crypto` risultante avrà il seguente aspetto:

```
"crypto": {
  "PKCS11": {
    "OpenSSLEngine": "...",
    "P11Provider": "...",
    "slotLabel": "...",
    "slotUserPin": "..."
  },
  "principals": {
    "IoTCertificate": {
      "privateKeyPath": "...",
      "certificatePath": "..."
    },
    "SecretsManager": {
      "privateKeyPath": "..."
    }
  },
  "caPath" : "..."
}
```

Suite di cifratura supportate per l'integrazione della sicurezza hardware

AWS IoT Greengrass supporta una serie di suite di crittografia quando il core è configurato per la sicurezza hardware. Si tratta di un sottoinsieme delle suite di cifratura supportate quando il core è configurato per l'uso della sicurezza basata su file. Per ulteriori informazioni, consulta [the section called “Supporto TLS per le suite di cifratura”](#).

Note

Quando ci si connette al core Greengrass dai dispositivi Greengrass sulla rete locale, assicurati di utilizzare una delle suite di cifratura supportate per effettuare la connessione TLS.

Configura il supporto per gli aggiornamenti over-the-air

Per abilitare gli aggiornamenti over-the-air (OTA) del software AWS IoT Greengrass Core quando si utilizza la sicurezza hardware, è necessario installare la libreria [wrapper OpenSC libp11 PKCS #11](#) e modificare il file di configurazione Greengrass. Per ulteriori informazioni sugli aggiornamenti OTA, consulta [Aggiornamenti OTA del software AWS IoT Greengrass Core](#).

1. Arrestare il daemon Greengrass.

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

Note

greengrass-root rappresenta il percorso in cui il software Core è installato sul dispositivo. AWS IoT Greengrass In genere, questa è la directory `/greengrass`.

2. Installare il motore OpenSSL. OpenSSL 1.0 o 1.1 sono supportati.

```
sudo apt-get install libengine-pkcs11-openssl
```

3. Individuare il percorso del motore OpenSSL (`libpkcs11.so`) nel sistema:
 - a. Ottenere l'elenco dei pacchetti installati per la libreria.


```
sudo dpkg -L libengine-pkcs11-openssl
```

Il file `libpkcs11.so` è disponibile nella directory `engines`.

- b. Copiare il percorso completo del file (ad esempio, `/usr/lib/ssl/engines/libpkcs11.so`).
4. Aprire il file di configurazione Greengrass. Si tratta del file [`config.json`](#) nella directory `/greengrass-root/config`.
5. Per la proprietà `OpenSSLEngine`, immettere il percorso del file `libpkcs11.so`.

```
{
  "crypto": {
    "caPath" : "file:///path-to-root-ca",
    "PKCS11" : {
      "OpenSSLEngine" : "/path-to-p11-openssl-engine",
      "P11Provider" : "/path-to-pkcs11-provider-so",
      "slotLabel" : "crypto-token-name",
      "slotUserPin" : "crypto-token-user-pin"
    },
    ...
  }
  ...
}
```

Note

Aggiungere la proprietà `OpenSSLEngine`, se non esiste nell'oggetto `PKCS11`.

6. Avviare il daemon Greengrass.

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

Retrocompatibilità con le versioni precedenti del software AWS IoT Greengrass principale

Il software AWS IoT Greengrass Core con supporto per la sicurezza hardware è completamente retrocompatibile con `config.json` i file generati per la versione 1.6 e versioni precedenti.

Se l'oggetto `crypto` non è presente nel file di configurazione `config.json`, AWS IoT Greengrass utilizza le proprietà e basate sul file `coreThing.certPath`, `coreThing.keyPath`, `coreThing.caPath`. Questa compatibilità con le versioni precedenti si applica agli aggiornamenti Greengrass OTA, che non sovrascrive una configurazione basata su file specificata in `config.json`.

Hardware senza supporto PKCS#11

La libreria PKCS#11 viene in genere offerta dal fornitore hardware o è open source. Ad esempio, con hardware conforme agli standard (ad esempio TPM1.2), potrebbe essere possibile utilizzare il software open source esistente. Tuttavia, se il tuo hardware non dispone di un'implementazione della libreria PKCS #11 corrispondente o se desideri scrivere un provider PKCS #11 personalizzato, contatta il tuo rappresentante Enterprise AWS Support per domande relative all'integrazione.

Consulta anche

- PKCS #11 Cryptographic Token Interface Usage Guide Version 2.40. Pubblicato da John Leiseboer e Robert Griffin. 16 Novembre 2014. Nota 02 del Comitato OASIS. <http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/cn02/pkcs11-ug-v2.40-cn02.html>. Ultima versione: <http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html>.
- [RFC 7512](#)
- [PKCS #1: RSA Encryption versione 1.5](#)

Autenticazione e autorizzazione del dispositivo per AWS IoT Greengrass

I dispositivi negli AWS IoT Greengrass ambienti utilizzano certificati X.509 per l'autenticazione e politiche di autorizzazione. AWS IoT I certificati e le politiche consentono ai dispositivi di connettersi in modo sicuro tra loro e. AWS IoT Core AWS IoT Greengrass

I certificati X.509 sono certificati digitali che usano lo standard di infrastruttura a chiave pubblica X.509 per associare una chiave pubblica all'identità contenuta in un certificato. I certificati X.509 vengono rilasciati da un'entità attendibile denominata autorità di certificazione (CA). L'autorità di certificazione mantiene uno o più certificati speciali chiamati certificati CA, che usa per rilasciare certificati X.509. Solo l'autorità di certificazione ha accesso a certificati CA.

AWS IoT le politiche definiscono l'insieme di operazioni consentite per AWS IoT i dispositivi. In particolare, consentono e negano l'accesso e le operazioni relative al AWS IoT Core piano AWS IoT Greengrass dati, come la pubblicazione di messaggi MQTT e il recupero delle ombre dei dispositivi.

Tutti i dispositivi richiedono una registrazione nel AWS IoT Core registro e un certificato X.509 attivato con una policy allegata. AWS IoT I dispositivi rientrano in due categorie:

- **ICore Greengrass.** I dispositivi core Greengrass utilizzano certificati e AWS IoT policy a cui connettersi. AWS IoT Core I certificati e le policy consentono inoltre di AWS IoT Greengrass distribuire informazioni di configurazione, funzioni Lambda, connettori e abbonamenti gestiti ai dispositivi principali.
- **Dispositivi client.** I dispositivi client (chiamati anche dispositivi connessi, dispositivi Greengrass o dispositivi) sono dispositivi che si connettono a un core Greengrass tramite MQTT. Utilizzano certificati e policy per connettersi AWS IoT Core e utilizzare il servizio. AWS IoT Greengrass Ciò consente ai dispositivi client di utilizzare il servizio AWS IoT Greengrass Discovery per trovare e connettersi a un dispositivo principale. Un dispositivo client utilizza lo stesso certificato per connettersi al gateway del AWS IoT Core dispositivo e al dispositivo principale. I dispositivi client utilizzano anche le informazioni di rilevamento per l'autenticazione reciproca con il dispositivo principale. Per ulteriori informazioni, consulta [the section called “Flusso di lavoro di connessione del dispositivo”](#) e [the section called “Gestione dell'autenticazione dei dispositivi con il core Greengrass”](#).

Certificati X.509

La comunicazione tra i dispositivi core e client e tra i dispositivi AWS IoT Core e/o AWS IoT Greengrass deve essere autenticata. Questa autenticazione si basa su certificati di dispositivo X.509 registrati e chiavi crittografiche.

In un AWS IoT Greengrass ambiente, i dispositivi utilizzano certificati con chiavi pubbliche e private per le seguenti connessioni Transport Layer Security (TLS):

- Il componente AWS IoT client del core Greengrass che si connette a AWS IoT Core e AWS IoT Greengrass tramite Internet.
- Dispositivi client che si connettono AWS IoT Greengrass per ottenere informazioni di discovery di base su Internet.
- Il componente server MQTT sul core Greengrass si connette ai dispositivi client del gruppo tramite la rete locale.

Il dispositivo AWS IoT Greengrass principale archivia i certificati in due posizioni:

- Certificato dispositivo core in `/greengrass-root/certs`. In genere, il certificato del dispositivo core è denominato `hash.cert.pem` (ad esempio `86c84488a5.cert.pem`). Questo certificato viene utilizzato dal AWS IoT client per l'autenticazione reciproca quando il core si connette ai AWS IoT Greengrass servizi AWS IoT Core and.
- Certificato server MQTT in `/greengrass-root/ggc/var/state/server`. Il certificato del server MQTT è denominato `server.crt`. Questo certificato viene utilizzato per l'autenticazione reciproca tra il server MQTT locale (nel core Greengrass) e i dispositivi Greengrass.

Note

`greengrass-root` rappresenta il percorso in cui il software AWS IoT Greengrass Core è installato sul dispositivo. In genere, questa è la directory `/greengrass`.

Per ulteriori informazioni, consulta [the section called “Principal di sicurezza”](#).

Certificati dell'autorità di certificazione (CA)

I dispositivi core e i dispositivi client scaricano un certificato CA principale utilizzato per l'autenticazione con AWS IoT Greengrass i servizi AWS IoT Core e. Ti consigliamo di utilizzare un certificato di CA root di Amazon Trust Services (ATS), ad esempio [Amazon Root CA 1](#). Per ulteriori informazioni, consulta [Certificati emessi da una CA per l'autenticazione del server](#) nella Guida per gli sviluppatori AWS IoT Core .

Note

Il tipo di certificato di root CA deve corrispondere all'endpoint. Utilizza un certificato CA radice ATS con un endpoint ATS (preferito) o un certificato CA VeriSign radice con un endpoint legacy. Solo alcune regioni di Amazon Web Services supportano gli endpoint legacy. Per ulteriori informazioni, consulta [the section called “Gli endpoint del servizio devono corrispondere al tipo di certificato”](#).

I dispositivi client scaricano anche il certificato CA del gruppo Greengrass. Questo viene utilizzato per convalidare il certificato server MQTT sul core Greengrass durante l'autenticazione reciproca. Per

ulteriori informazioni, consulta [the section called “Flusso di lavoro di connessione del dispositivo”](#). La scadenza predefinita del certificato server MQTT è sette giorni.

Rotazione dei certificati sul server MQTT locale

I dispositivi client utilizzano il certificato del server MQTT locale per l'autenticazione reciproca con il dispositivo principale Greengrass. Per impostazione predefinita, questo certificato scade entro sette giorni. Questo periodo limitato si basa sulle best practice in materia di sicurezza. Il certificato del server MQTT è firmato dal certificato CA del gruppo, archiviato nel cloud.

Affinché avvenga la rotazione dei certificati, il dispositivo principale Greengrass deve essere online e in grado di accedere direttamente al AWS IoT Greengrass servizio su base regolare. Quando il certificato scade, il dispositivo principale tenta di connettersi al AWS IoT Greengrass servizio per ottenere un nuovo certificato. Se viene stabilita la connessione, il dispositivo core scarica un nuovo certificato del server MQTT e riavvia il servizio MQTT locale. A questo punto, tutti i dispositivi client collegati al core vengono disconnessi. Se il dispositivo principale è offline al momento della scadenza, non riceve il certificato sostitutivo. Eventuali nuovi tentativi di connessione al dispositivo core vengono rifiutati. Le connessioni esistenti non sono interessate. I dispositivi client non possono connettersi al dispositivo principale finché non viene ripristinata la connessione al AWS IoT Greengrass servizio e non è possibile scaricare un nuovo certificato del server MQTT.

Puoi impostare il periodo di scadenza su qualsiasi valore compreso tra 7 e 30 giorni, in base alle esigenze. Una rotazione più frequente richiede una connessione cloud più frequente. Una rotazione meno frequente può causare problemi di sicurezza. Se desideri impostare la scadenza del certificato su un valore superiore a 30 giorni, contatta Supporto.

Nella AWS IoT console, puoi gestire il certificato nella pagina Impostazioni del gruppo. Nell' AWS IoT Greengrass API, puoi utilizzare l'[UpdateGroupCertificateConfiguration](#) azione.

Quando il certificato del server MQTT scade, ogni tentativo di convalidare il certificato ha esito negativo. I dispositivi client devono essere in grado di rilevare l'errore e interrompere la connessione.

AWS IoT politiche per le operazioni sul piano dati

Utilizza AWS IoT le politiche per autorizzare l'accesso al piano AWS IoT Greengrass dati AWS IoT Core and. Il piano AWS IoT Core dati è costituito da operazioni per dispositivi, utenti e applicazioni, come la connessione AWS IoT Core e la sottoscrizione agli argomenti. Il piano dei dati AWS IoT Greengrass è costituito da operazioni per i dispositivi Greengrass, come il recupero delle distribuzioni e l'aggiornamento delle informazioni sulla connettività.

Una AWS IoT policy è un documento JSON simile a una [policy IAM](#). Contiene una o più istruzioni delle policy che specificano le proprietà seguenti:

- **Effect.** La modalità di accesso, che può essere Allow o Deny.
- **Action.** L'elenco delle azioni consentite o negate dalla politica.
- **Resource.** L'elenco delle risorse su cui l'azione è consentita o negata.

AWS IoT i criteri supportano i caratteri jolly * come caratteri jolly e trattano i caratteri jolly MQTT (+and#) come stringhe letterali. Per ulteriori informazioni sui caratteri jolly, consultate [* Using jolly in resource](#) nella [Guida](#) per l'utente. ARNs AWS Identity and Access Management

Per ulteriori informazioni, consulta [Policy AWS IoT](#) e [Operazioni di policy AWS IoT](#) nella Guida per gli sviluppatori AWS IoT Core .

Note

AWS IoT Core consente di allegare AWS IoT policy a gruppi di oggetti per definire le autorizzazioni per gruppi di dispositivi. Le policy dei gruppi di oggetti non consentono l'accesso alle operazioni del piano AWS IoT Greengrass dati. Per consentire a un oggetto l'accesso a un'operazione del piano AWS IoT Greengrass dati, aggiungi l'autorizzazione a una AWS IoT policy che alleggi al certificato dell'oggetto.

AWS IoT Greengrass azioni politiche

Operazioni fondamentali di Greengrass

AWS IoT Greengrass definisce le seguenti azioni politiche che i dispositivi core Greengrass possono utilizzare nelle AWS IoT politiche:

`greengrass:AssumeRoleForGroup`

Autorizzazione per un dispositivo core Greengrass a recuperare le credenziali utilizzando la funzione Lambda del sistema Token Exchange Service (TES). Le autorizzazioni associate alle credenziali recuperate si basano sulla policy associata al ruolo del gruppo configurato.

Questa autorizzazione viene controllata quando un dispositivo core di Greengrass tenta di recuperare le credenziali (supponendo che le credenziali non siano memorizzate nella cache localmente).

`greengrass:CreateCertificate`

Autorizzazione per un dispositivo core Greengrass per creare il proprio certificato server.

Questa autorizzazione viene controllata quando un dispositivo core di Greengrass crea un certificato. I dispositivi core Greengrass tentano di creare un certificato server al primo avvio, quando le informazioni di connettività del core cambiano e nei periodi di rotazione designati.

`greengrass:GetConnectivityInfo`

Autorizzazione per un dispositivo core Greengrass per recuperare le proprie informazioni di connettività.

Questa autorizzazione viene verificata quando un dispositivo core Greengrass tenta di recuperare le informazioni di connettività da AWS IoT Core.

`greengrass:GetDeployment`

Autorizzazione per un dispositivo core Greengrass per recuperare le distribuzioni.

Questa autorizzazione viene controllata quando un dispositivo core di Greengrass tenta di recuperare le distribuzioni e gli stati di distribuzione dal cloud.

`greengrass:GetDeploymentArtifacts`

Autorizzazione per un dispositivo principale Greengrass a recuperare elementi di distribuzione come informazioni di gruppo o funzioni Lambda.

Questa autorizzazione viene controllata quando un dispositivo core Greengrass riceve una distribuzione e quindi tenta di recuperare gli artefatti della distribuzione.

`greengrass:UpdateConnectivityInfo`

Autorizzazione per un dispositivo core Greengrass per aggiornare le proprie informazioni di connettività con informazioni IP o hostname.

Questa autorizzazione viene controllata quando un dispositivo core Greengrass tenta di aggiornare le informazioni di connettività nel cloud.

`greengrass:UpdateCoreDeploymentStatus`

Autorizzazione per un dispositivo core Greengrass per aggiornare lo stato di una distribuzione.

Questa autorizzazione viene controllata quando un dispositivo core Greengrass riceve una distribuzione e tenta di aggiornare lo stato della distribuzione.

Operazioni dispositivo Greengrass

AWS IoT Greengrass definisce le seguenti azioni politiche che i dispositivi client possono utilizzare nelle politiche: AWS IoT

`greengrass:Discover`

Autorizzazione per un dispositivo client a utilizzare l'[API Discovery](#) per recuperare le informazioni di connettività principali del gruppo e l'autorità di certificazione del gruppo.

Questa autorizzazione viene verificata quando un dispositivo client chiama l'API Discovery con autenticazione reciproca TLS.

AWS IoT Politica minima per il dispositivo AWS IoT Greengrass principale

L'esempio di policy seguente include il set di operazioni minime necessarie per supportare le funzionalità di base di Greengrass per il dispositivo core.

- La policy elenca gli argomenti MQTT e i filtri di argomento su cui il dispositivo core può pubblicare messaggi, sottoscrivere e ricevere messaggi, inclusi gli argomenti utilizzati per lo stato shadow. Per supportare lo scambio di messaggi tra AWS IoT Core funzioni Lambda, connettori e dispositivi client nel gruppo Greengrass, specifica gli argomenti e i filtri degli argomenti che desideri consentire. Per ulteriori informazioni, consulta [Esempi di pubblicazione/sottoscrizione a policy](#) nella Guida per gli sviluppatori AWS IoT Core .
- La politica include una sezione che consente di AWS IoT Core ottenere, aggiornare ed eliminare l'ombra del dispositivo principale. Per consentire la sincronizzazione shadow per i dispositivi client del gruppo Greengrass, specifica l'Amazon Resource Names (ARNs) di destinazione nell'`Resource`elenco (ad esempio, `arn:aws:iot:region:account-id:thing/device-name`).
- L'uso di [thing policy variables](#) (`iot:Connection.Thing.*`) nella AWS IoT policy per un dispositivo principale non è supportato. Il core utilizza lo stesso certificato del dispositivo per effettuare [più connessioni](#) AWS IoT Core , ma l'ID client in una connessione potrebbe non corrispondere esattamente al nome dell'oggetto principale.
- Per l'autorizzazione `greengrass:UpdateCoreDeploymentStatus`, il segmento finale nell'ARN `Resource` è l'ARN codificato in formato URL del dispositivo core.


```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect"
      ],
      "Resource": [
        "arn:aws:iot:region:account-id:client/core-name-*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:region:account-id:topic/$aws/things/core-name-*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:region:account-id:topicfilter/$aws/things/core-name-*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:DeleteThingShadow"
      ],
      "Resource": [
        "arn:aws:iot:region:account-id:thing/core-name-*"
      ]
    },
    {

```

```

        "Effect": "Allow",
        "Action": [
            "greengrass:AssumeRoleForGroup",
            "greengrass>CreateCertificate"
        ],
        "Resource": [
            "*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "greengrass:GetDeployment"
        ],
        "Resource": [
            "arn:aws:greengrass:region:account-id:/greengrass/groups/group-id/
deployments/*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "greengrass:GetDeploymentArtifacts"
        ],
        "Resource": [
            "arn:aws:greengrass:region:account-id:/greengrass/groups/group-id/
deployments/*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "greengrass:UpdateCoreDeploymentStatus"
        ],
        "Resource": [
            "arn:aws:greengrass:region:account-id:/greengrass/groups/group-id/
deployments/*/cores/arn%3Aaws%3Aiot%3Aregion%3Aaccount-id%3Athing%2Fcore-name"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "greengrass:GetConnectivityInfo",
            "greengrass:UpdateConnectivityInfo"
        ]
    }

```

```

    ],
    "Resource": [
        "arn:aws:iot:region:account-id:thing/core-name-"
    ]
}
]
}

```

Note

AWS IoT le politiche per i dispositivi client richiedono in genere autorizzazioni simili per `iot:Connect`, `iot:Publish`, `iot:Receive`, e `iot:Subscribe` azioni.

Per consentire a un dispositivo client di rilevare automaticamente le informazioni di connettività per i core dei gruppi Greengrass a cui appartiene il dispositivo, AWS IoT la politica per un dispositivo client deve includere `greengrass:Discover` l'azione. Nella `Resource` sezione, specificare l'ARN del dispositivo client, non l'ARN del dispositivo principale Greengrass. Per esempio:

```

{
  "Effect": "Allow",
  "Action": [
    "greengrass:Discover"
  ],
  "Resource": [
    "arn:aws:iot:region:account-id:thing/device-name"
  ]
}

```

La AWS IoT politica per i dispositivi client in genere non richiede autorizzazioni o `iot:DeleteThingShadow` azioni `iot:GetThingShadow` o `iot:UpdateThingShadow`, poiché il core Greengrass gestisce le operazioni di sincronizzazione degli shadow per i dispositivi client. In questo caso, assicurati che la `Resource` sezione relativa alle azioni ombra nella AWS IoT policy di base includa i ARNs dispositivi client.

Nella AWS IoT console, puoi visualizzare e modificare la policy allegata al certificato principale.

1. Nel riquadro di navigazione, in **Gestisci**, espandi **Tutti i dispositivi**, quindi scegli **Cose**.

2. Scegli il tuo core.
3. Nella pagina di configurazione del core, scegli la scheda Certificati.
4. Nella scheda Certificati, scegli il tuo certificato.
5. Nella pagina di configurazione del certificato, scegli Policies (Policy) e quindi scegli la policy.

Se desideri modificare la politica, scegli Modifica versione attiva.

6. Rivedi la politica e aggiungi, rimuovi o modifica le autorizzazioni secondo necessità.
7. Per impostare una nuova versione della politica come versione attiva, in Stato della versione della politica, seleziona Imposta la versione modificata come versione attiva per questa politica.
8. Scegli Salva come nuova versione.

Gestione delle identità e degli accessi per AWS IoT Greengrass

AWS Identity and Access Management (IAM) è un software Servizio AWS che aiuta un amministratore a controllare in modo sicuro l'accesso alle AWS risorse. Gli amministratori IAM controllano chi può essere autenticato (effettuato l'accesso) e autorizzato (disporre delle autorizzazioni) a utilizzare le risorse. AWS IoT Greengrass IAM è uno Servizio AWS strumento che puoi utilizzare senza costi aggiuntivi.

Note

Questo argomento descrive i concetti e le funzionalità di IAM. Per informazioni sulle funzionalità IAM supportate da AWS IoT Greengrass, consulta [the section called “Come AWS IoT Greengrass funziona con IAM”](#).

Destinatari

Il modo in cui utilizzi AWS Identity and Access Management (IAM) varia a seconda del lavoro che AWS IoT Greengrass svolgi.

Utente del servizio: se utilizzi il AWS IoT Greengrass servizio per svolgere il tuo lavoro, l'amministratore ti fornisce le credenziali e le autorizzazioni necessarie. Man mano che utilizzi più AWS IoT Greengrass funzionalità per svolgere il tuo lavoro, potresti aver bisogno di autorizzazioni aggiuntive. La comprensione della gestione dell'accesso ti consente di richiedere le autorizzazioni

corrette all'amministratore. Se non riesci ad accedere a una funzionalità di AWS IoT Greengrass, consulta [Risoluzione dei problemi di identità e accesso per AWS IoT Greengrass](#).

Amministratore del servizio: se sei responsabile delle AWS IoT Greengrass risorse della tua azienda, probabilmente hai pieno accesso a AWS IoT Greengrass. È tuo compito determinare a quali AWS IoT Greengrass funzionalità e risorse devono accedere gli utenti del servizio. Devi inviare le richieste all'amministratore IAM per cambiare le autorizzazioni degli utenti del servizio. Esamina le informazioni contenute in questa pagina per comprendere i concetti di base relativi a IAM. Per saperne di più su come la tua azienda può utilizzare IAM con AWS IoT Greengrass, consulta [Come AWS IoT Greengrass funziona con IAM](#).

Amministratore IAM: un amministratore IAM potrebbe essere interessato a ottenere dei dettagli su come scrivere policy per gestire l'accesso a AWS IoT Greengrass. Per visualizzare esempi di policy AWS IoT Greengrass basate sull'identità che puoi utilizzare in IAM, consulta. [Esempi di policy basate su identità per AWS IoT Greengrass](#)

Autenticazione con identità

L'autenticazione è il modo in cui accedi AWS utilizzando le tue credenziali di identità. Devi essere autenticato (aver effettuato l' Utente root dell'account AWS accesso AWS) come utente IAM o assumendo un ruolo IAM.

Puoi accedere AWS come identità federata utilizzando le credenziali fornite tramite una fonte di identità. AWS IAM Identity Center Gli utenti (IAM Identity Center), l'autenticazione Single Sign-On della tua azienda e le tue credenziali di Google o Facebook sono esempi di identità federate. Se accedi come identità federata, l'amministratore ha configurato in precedenza la federazione delle identità utilizzando i ruoli IAM. Quando accedi AWS utilizzando la federazione, assumi indirettamente un ruolo.

A seconda del tipo di utente, puoi accedere al AWS Management Console o al portale di AWS accesso. Per ulteriori informazioni sull'accesso a AWS, vedi [Come accedere al tuo Account AWS nella Guida per l'Accedi ad AWS utente](#).

Se accedi a AWS livello di codice, AWS fornisce un kit di sviluppo software (SDK) e un'interfaccia a riga di comando (CLI) per firmare crittograficamente le tue richieste utilizzando le tue credenziali. Se non utilizzi AWS strumenti, devi firmare tu stesso le richieste. Per ulteriori informazioni sul metodo consigliato per la firma delle richieste, consulta [Signature Version 4 AWS per le richieste API](#) nella Guida per l'utente IAM.

A prescindere dal metodo di autenticazione utilizzato, potrebbe essere necessario specificare ulteriori informazioni sulla sicurezza. Ad esempio, ti AWS consiglia di utilizzare l'autenticazione a più fattori (MFA) per aumentare la sicurezza del tuo account. Per ulteriori informazioni, consulta [Autenticazione a più fattori](#) nella Guida per l'utente di AWS IAM Identity Center e [Utilizzo dell'autenticazione a più fattori \(MFA\)AWS in IAM](#) nella Guida per l'utente IAM.

Account AWS utente root

Quando si crea un account Account AWS, si inizia con un'identità di accesso che ha accesso completo a tutte Servizi AWS le risorse dell'account. Questa identità è denominata utente Account AWS root ed è accessibile effettuando l'accesso con l'indirizzo e-mail e la password utilizzati per creare l'account. Si consiglia vivamente di non utilizzare l'utente root per le attività quotidiane. Conserva le credenziali dell'utente root e utilizzale per eseguire le operazioni che solo l'utente root può eseguire. Per un elenco completo delle attività che richiedono l'accesso come utente root, consulta la sezione [Attività che richiedono le credenziali dell'utente root](#) nella Guida per l'utente IAM.

Utenti e gruppi IAM

Un [utente IAM](#) è un'identità interna Account AWS che dispone di autorizzazioni specifiche per una singola persona o applicazione. Ove possibile, consigliamo di fare affidamento a credenziali temporanee invece di creare utenti IAM con credenziali a lungo termine come le password e le chiavi di accesso. Tuttavia, se si hanno casi d'uso specifici che richiedono credenziali a lungo termine con utenti IAM, si consiglia di ruotare le chiavi di accesso. Per ulteriori informazioni, consulta la pagina [Rotazione periodica delle chiavi di accesso per casi d'uso che richiedono credenziali a lungo termine](#) nella Guida per l'utente IAM.

Un [gruppo IAM](#) è un'identità che specifica un insieme di utenti IAM. Non è possibile eseguire l'accesso come gruppo. È possibile utilizzare gruppi per specificare le autorizzazioni per più utenti alla volta. I gruppi semplificano la gestione delle autorizzazioni per set di utenti di grandi dimensioni. Ad esempio, potresti avere un gruppo denominato IAMAdminse concedere a quel gruppo le autorizzazioni per amministrare le risorse IAM.

Gli utenti sono diversi dai ruoli. Un utente è associato in modo univoco a una persona o un'applicazione, mentre un ruolo è destinato a essere assunto da chiunque ne abbia bisogno. Gli utenti dispongono di credenziali a lungo termine permanenti, mentre i ruoli forniscono credenziali temporanee. Per ulteriori informazioni, consulta [Casi d'uso per utenti IAM](#) nella Guida per l'utente IAM.

Ruoli IAM

Un [ruolo IAM](#) è un'identità interna all'utente Account AWS che dispone di autorizzazioni specifiche. È simile a un utente IAM, ma non è associato a una persona specifica. Per assumere temporaneamente un ruolo IAM in AWS Management Console, puoi [passare da un ruolo utente a un ruolo IAM \(console\)](#). Puoi assumere un ruolo chiamando un'operazione AWS CLI o AWS API o utilizzando un URL personalizzato. Per ulteriori informazioni sui metodi per l'utilizzo dei ruoli, consulta [Utilizzo di ruoli IAM](#) nella Guida per l'utente IAM.

I ruoli IAM con credenziali temporanee sono utili nelle seguenti situazioni:

- **Accesso utente federato:** per assegnare le autorizzazioni a una identità federata, è possibile creare un ruolo e definire le autorizzazioni per il ruolo. Quando un'identità federata viene autenticata, l'identità viene associata al ruolo e ottiene le autorizzazioni da esso definite. Per ulteriori informazioni sulla federazione dei ruoli, consulta [Create a role for a third-party identity provider \(federation\)](#) nella Guida per l'utente IAM. Se utilizzi IAM Identity Center, configura un set di autorizzazioni. IAM Identity Center mette in correlazione il set di autorizzazioni con un ruolo in IAM per controllare a cosa possono accedere le identità dopo l'autenticazione. Per informazioni sui set di autorizzazioni, consulta [Set di autorizzazioni](#) nella Guida per l'utente di AWS IAM Identity Center.
- **Autorizzazioni utente IAM temporanee:** un utente IAM o un ruolo può assumere un ruolo IAM per ottenere temporaneamente autorizzazioni diverse per un'attività specifica.
- **Accesso multi-account:** è possibile utilizzare un ruolo IAM per permettere a un utente (un principale affidabile) con un account diverso di accedere alle risorse nell'account. I ruoli sono lo strumento principale per concedere l'accesso multi-account. Tuttavia, con alcuni Servizi AWS, è possibile allegare una policy direttamente a una risorsa (anziché utilizzare un ruolo come proxy). Per informazioni sulle differenze tra ruoli e policy basate su risorse per l'accesso multi-account, consulta [Accesso a risorse multi-account in IAM](#) nella Guida per l'utente IAM.
- **Accesso a più servizi:** alcuni Servizi AWS utilizzano le funzionalità di altri Servizi AWS. Ad esempio, quando effettui una chiamata in un servizio, è normale che quel servizio esegua applicazioni in Amazon EC2 o archivi oggetti in Amazon S3. Un servizio può eseguire questa operazione utilizzando le autorizzazioni dell'entità chiamante, utilizzando un ruolo di servizio o utilizzando un ruolo collegato al servizio.
- **Sessioni di accesso inoltrato (FAS):** quando utilizzi un utente o un ruolo IAM per eseguire azioni AWS, sei considerato un principale. Quando si utilizzano alcuni servizi, è possibile eseguire un'operazione che attiva un'altra operazione in un servizio diverso. FAS utilizza le autorizzazioni del principale che chiama un Servizio AWS, combinate con la richiesta Servizio AWS per

effettuare richieste ai servizi downstream. Le richieste FAS vengono effettuate solo quando un servizio riceve una richiesta che richiede interazioni con altri Servizi AWS o risorse per essere completata. In questo caso è necessario disporre delle autorizzazioni per eseguire entrambe le operazioni. Per i dettagli delle policy relative alle richieste FAS, consulta [Forward access sessions](#).

- Ruolo di servizio: un ruolo di servizio è un [ruolo IAM](#) che un servizio assume per eseguire operazioni per tuo conto. Un amministratore IAM può creare, modificare ed eliminare un ruolo di servizio dall'interno di IAM. Per ulteriori informazioni, consulta la sezione [Create a role to delegate permissions to an Servizio AWS](#) nella Guida per l'utente IAM.
- Ruolo collegato al servizio: un ruolo collegato al servizio è un tipo di ruolo di servizio collegato a un Servizio AWS. Il servizio può assumere il ruolo per eseguire un'azione per tuo conto. I ruoli collegati al servizio vengono visualizzati nel tuo account Account AWS e sono di proprietà del servizio. Un amministratore IAM può visualizzare le autorizzazioni per i ruoli collegati ai servizi, ma non modificarle.
- Applicazioni in esecuzione su Amazon EC2: puoi utilizzare un ruolo IAM per gestire le credenziali temporanee per le applicazioni in esecuzione su un' EC2 istanza e che AWS CLI effettuano richieste AWS API. Questa soluzione è preferibile alla memorizzazione delle chiavi di accesso all'interno dell' EC2 istanza. Per assegnare un AWS ruolo a un' EC2 istanza e renderlo disponibile per tutte le sue applicazioni, create un profilo di istanza collegato all'istanza. Un profilo di istanza contiene il ruolo e consente ai programmi in esecuzione sull' EC2 istanza di ottenere credenziali temporanee. Per ulteriori informazioni, consulta [Utilizzare un ruolo IAM per concedere le autorizzazioni alle applicazioni in esecuzione su EC2 istanze Amazon](#) nella IAM User Guide.

Gestione dell'accesso con policy

Puoi controllare l'accesso AWS creando policy e collegandole a AWS identità o risorse. Una policy è un oggetto AWS che, se associato a un'identità o a una risorsa, ne definisce le autorizzazioni. AWS valuta queste politiche quando un principale (utente, utente root o sessione di ruolo) effettua una richiesta. Le autorizzazioni nelle policy determinano l'approvazione o il rifiuto della richiesta. La maggior parte delle politiche viene archiviata AWS come documenti JSON. Per ulteriori informazioni sulla struttura e sui contenuti dei documenti delle policy JSON, consulta [Panoramica delle policy JSON](#) nella Guida per l'utente IAM.

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale principale può eseguire operazioni su quali risorse e in quali condizioni.

Per impostazione predefinita, utenti e ruoli non dispongono di autorizzazioni. Per concedere agli utenti l'autorizzazione a eseguire operazioni sulle risorse di cui hanno bisogno, un amministratore IAM può creare policy IAM. L'amministratore può quindi aggiungere le policy IAM ai ruoli e gli utenti possono assumere i ruoli.

Le policy IAM definiscono le autorizzazioni relative a un'operazione, a prescindere dal metodo utilizzato per eseguirla. Ad esempio, supponiamo di disporre di una policy che consente l'operazione `iam:GetRole`. Un utente con tale policy può ottenere informazioni sul ruolo dall' AWS Management Console AWS CLI, dall'o dall' AWS API.

Policy basate sull'identità

Le policy basate su identità sono documenti di policy di autorizzazione JSON che è possibile allegare a un'identità (utente, gruppo di utenti o ruolo IAM). Tali policy definiscono le operazioni che utenti e ruoli possono eseguire, su quali risorse e in quali condizioni. Per informazioni su come creare una policy basata su identità, consulta [Definizione di autorizzazioni personalizzate IAM con policy gestite dal cliente](#) nella Guida per l'utente IAM.

Le policy basate su identità possono essere ulteriormente classificate come policy inline o policy gestite. Le policy inline sono integrate direttamente in un singolo utente, gruppo o ruolo. Le politiche gestite sono politiche autonome che puoi allegare a più utenti, gruppi e ruoli nel tuo Account AWS. Le politiche gestite includono politiche AWS gestite e politiche gestite dai clienti. Per informazioni su come scegliere tra una policy gestita o una policy inline, consulta [Scelta fra policy gestite e policy inline](#) nella Guida per l'utente IAM.

Policy basate sulle risorse

Le policy basate su risorse sono documenti di policy JSON che è possibile collegare a una risorsa. Esempi di policy basate sulle risorse sono le policy di attendibilità dei ruoli IAM e le policy dei bucket Amazon S3. Nei servizi che supportano policy basate sulle risorse, gli amministratori dei servizi possono utilizzarli per controllare l'accesso a una risorsa specifica. Quando è collegata a una risorsa, una policy definisce le operazioni che un principale può eseguire su tale risorsa e a quali condizioni. È necessario [specificare un principale](#) in una policy basata sulle risorse. I principali possono includere account, utenti, ruoli, utenti federati o. Servizi AWS

Le policy basate sulle risorse sono policy inline che si trovano in tale servizio. Non puoi utilizzare le policy AWS gestite di IAM in una policy basata sulle risorse.

Elenchi di controllo degli accessi (ACLs)

Le liste di controllo degli accessi (ACLs) controllano quali principali (membri dell'account, utenti o ruoli) dispongono delle autorizzazioni per accedere a una risorsa. ACLs sono simili alle politiche basate sulle risorse, sebbene non utilizzino il formato del documento di policy JSON.

Amazon S3 e Amazon VPC sono esempi di servizi che supportano. AWS WAF ACLs Per ulteriori informazioni ACLs, consulta la [panoramica della lista di controllo degli accessi \(ACL\)](#) nella Amazon Simple Storage Service Developer Guide.

Altri tipi di policy

AWS supporta tipi di policy aggiuntivi e meno comuni. Questi tipi di policy possono impostare il numero massimo di autorizzazioni concesse dai tipi di policy più comuni.

- **Limiti delle autorizzazioni:** un limite delle autorizzazioni è una funzionalità avanzata nella quale si imposta il numero massimo di autorizzazioni che una policy basata su identità può concedere a un'entità IAM (utente o ruolo IAM). È possibile impostare un limite delle autorizzazioni per un'entità. Le autorizzazioni risultanti sono l'intersezione delle policy basate su identità dell'entità e i relativi limiti delle autorizzazioni. Le policy basate su risorse che specificano l'utente o il ruolo nel campo `Principal` sono condizionate dal limite delle autorizzazioni. Un rifiuto esplicito in una qualsiasi di queste policy sostituisce l'autorizzazione. Per ulteriori informazioni sui limiti delle autorizzazioni, consulta [Limiti delle autorizzazioni per le entità IAM](#) nella Guida per l'utente IAM.
- **Politiche di controllo del servizio (SCPs):** SCPs sono politiche JSON che specificano le autorizzazioni massime per un'organizzazione o un'unità organizzativa (OU) in. AWS Organizations AWS Organizations è un servizio per il raggruppamento e la gestione centralizzata di più di proprietà dell' Account AWS azienda. Se abiliti tutte le funzionalità di un'organizzazione, puoi applicare le politiche di controllo del servizio (SCPs) a uno o tutti i tuoi account. L'SCP limita le autorizzazioni per le entità presenti negli account dei membri, inclusa ciascuna di esse. Utente root dell'account AWS Per ulteriori informazioni su Organizations and SCPs, consulta [le politiche di controllo dei servizi](#) nella Guida AWS Organizations per l'utente.
- **Politiche di controllo delle risorse (RCPs):** RCPs sono politiche JSON che puoi utilizzare per impostare le autorizzazioni massime disponibili per le risorse nei tuoi account senza aggiornare le politiche IAM allegate a ciascuna risorsa di tua proprietà. L'RCP limita le autorizzazioni per le risorse negli account dei membri e può influire sulle autorizzazioni effettive per le identità, incluse le Utente root dell'account AWS, indipendentemente dal fatto che appartengano o meno all'organizzazione. Per ulteriori informazioni su Organizations e RCPs, incluso un elenco di

Servizi AWS tale supporto RCPs, vedere [Resource control policies \(RCPs\)](#) nella Guida per l'AWS Organizations utente.

- Policy di sessione: le policy di sessione sono policy avanzate che vengono trasmesse come parametro quando si crea in modo programmatico una sessione temporanea per un ruolo o un utente federato. Le autorizzazioni della sessione risultante sono l'intersezione delle policy basate su identità del ruolo o dell'utente e le policy di sessione. Le autorizzazioni possono anche provenire da una policy basata su risorse. Un rifiuto esplicito in una qualsiasi di queste policy sostituisce l'autorizzazione. Per ulteriori informazioni, consulta [Policy di sessione](#) nella Guida per l'utente IAM.

Più tipi di policy

Quando più tipi di policy si applicano a una richiesta, le autorizzazioni risultanti sono più complicate da comprendere. Per scoprire come si AWS determina se consentire o meno una richiesta quando sono coinvolti più tipi di policy, consulta la [logica di valutazione delle policy](#) nella IAM User Guide.

Consulta anche

- [the section called “Come AWS IoT Greengrass funziona con IAM”](#)
- [the section called “Esempi di policy basate su identità”](#)
- [the section called “Risoluzione dei problemi di identità e accesso”](#)

Come AWS IoT Greengrass funziona con IAM

Prima di utilizzare IAM per gestire l'accesso AWS IoT Greengrass, è necessario comprendere le funzionalità IAM con cui è possibile utilizzare AWS IoT Greengrass.

Funzionalità IAM	Supportata da Greengrass?
Policy basate sull'identità con autorizzazioni a livello di risorse	Sì
Policy basate su risorse	No
Liste di controllo degli accessi (ACLs)	No
Autorizzazione basata su tag	No

Funzionalità IAM	Supportata da Greengrass?
Credenziali temporanee	Sì
Ruoli collegati al servizio	No
Ruoli di servizio	Sì

Per una panoramica di alto livello su come gli altri AWS servizi funzionano con IAM, consulta [AWS i servizi che funzionano con IAM nella IAM User Guide](#).

Politiche basate sull'identità per AWS IoT Greengrass

Con le policy basate sull'identità IAM, puoi specificare azioni e risorse consentite o negate e le condizioni in base alle quali le azioni sono consentite o negate. AWS IoT Greengrass supporta azioni, risorse e chiavi di condizione specifiche. Per conoscere tutti gli elementi utilizzati in una policy, consulta il [riferimento agli elementi della policy IAM JSON](#) nella IAM User Guide.

Operazioni

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale principale può eseguire operazioni su quali risorse, e in quali condizioni.

L'elemento `Action` di una policy JSON descrive le operazioni che è possibile utilizzare per consentire o negare l'accesso a un criterio. Le azioni politiche in genere hanno lo stesso nome dell'operazione AWS API associata. Ci sono alcune eccezioni, ad esempio le operazioni di sola autorizzazione che non hanno un'operazione API corrispondente. Esistono anche alcune operazioni che richiedono più operazioni in una policy. Queste operazioni aggiuntive sono denominate operazioni dipendenti.

Includi le operazioni in una policy per concedere le autorizzazioni a eseguire l'operazione associata.

Azioni politiche per AWS IoT Greengrass utilizzare il `greengrass:` prefisso prima dell'azione. Ad esempio, per consentire a qualcuno di utilizzare l'operazione `ListGroups` API per elencare i gruppi che lo compongono Account AWS, includi `greengrass:ListGroups` azione nella sua politica. Le istruzioni delle policy devono includere un elemento `Action` o `NotAction`. AWS IoT Greengrass definisce un proprio set di operazioni che descrivono le attività che è possibile eseguire con questo servizio.

Per specificare più azioni in una singola istruzione, elencale tra parentesi (`[]`) e separale con virgole, come segue:

```
"Action": [  
  "greengrass:action1",  
  "greengrass:action2",  
  "greengrass:action3"  
]
```

È possibile utilizzare i caratteri jolly (*) per specificare più operazioni. Ad esempio, per specificare tutte le azioni che iniziano con la parola `List`, includi la seguente azione:

```
"Action": "greengrass:List*"
```

Note

Si consiglia di evitare l'uso di caratteri jolly per specificare tutte le operazioni disponibili per un servizio. Come procedura consigliata, è necessario concedere privilegi minimi e autorizzazioni di ambito ristretto in una policy. Per ulteriori informazioni, consulta [the section called “Concedere autorizzazioni minime possibili”](#).

Per l'elenco completo delle AWS IoT Greengrass azioni, consulta [Actions Defined by AWS IoT Greengrass](#) nella IAM User Guide.

Risorse

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale principale può eseguire operazioni su quali risorse, e in quali condizioni.

L'elemento JSON `Resource` della policy specifica l'oggetto o gli oggetti ai quali si applica l'operazione. Le istruzioni devono includere un elemento `Resource` o un elemento `NotResource`. Come best practice, specifica una risorsa utilizzando il suo [nome della risorsa Amazon \(ARN\)](#). È possibile eseguire questa operazione per operazioni che supportano un tipo di risorsa specifico, note come autorizzazioni a livello di risorsa.

Per le operazioni che non supportano le autorizzazioni a livello di risorsa, ad esempio le operazioni di elenco, utilizza un carattere jolly (*) per indicare che l'istruzione si applica a tutte le risorse.

```
"Resource": "*" 
```

La tabella seguente contiene la AWS IoT Greengrass risorsa ARNs che può essere utilizzata nell'Resourceelemento di una dichiarazione politica. Per una mappatura delle autorizzazioni a livello di risorsa supportate per le AWS IoT Greengrass azioni, consulta [Actions Defined by AWS IoT Greengrass](#) nella IAM User Guide.

Risorsa	ARN
Group	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/groups/\${GroupId}
GroupVersion	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/groups/\${GroupId}/versions/\${VersionId}
CertificateAuthority	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/groups/\${GroupId}/certificateauthorities/\${CertificateAuthorityId}
Deployment	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/groups/\${GroupId}/deployments/\${DeploymentId}
BulkDeployment	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/bulk/deployments/\${BulkDeploymentId}
ConnectorDefinition	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/connectors/\${ConnectorDefinitionId}
ConnectorDefinitionVersion	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/connectors/\${ConnectorDefinitionId}/versions/\${VersionId}
CoreDefinition	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/cores/\${CoreDefinitionId}
CoreDefinitionVersion	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/cores/\${CoreDefinitionId}/versions/\${VersionId}

Risorsa	ARN
DeviceDefinition	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/devices/\${DeviceDefinitionId}
DeviceDefinitionVersion	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/devices/\${DeviceDefinitionId}/versions/\${VersionId}
FunctionDefinition	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/functions/\${FunctionDefinitionId}
FunctionDefinitionVersion	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/functions/\${FunctionDefinitionId}/versions/\${VersionId}
LoggerDefinition	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/loggers/\${LoggerDefinitionId}
LoggerDefinitionVersion	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/loggers/\${LoggerDefinitionId}/versions/\${VersionId}
ResourceDefinition	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/resources/\${ResourceDefinitionId}
ResourceDefinitionVersion	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/resources/\${ResourceDefinitionId}/versions/\${VersionId}
SubscriptionDefinition	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/subscriptions/\${SubscriptionDefinitionId}

Risorsa	ARN
SubscriptionDefinitionVersion	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/subscriptions/\${SubscriptionDefinitionId}/versions/\${VersionId}
ConnectivityInfo	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/things/\${ThingName}/connectivityInfo

L'Resourceelemento di esempio seguente specifica l'ARN di un gruppo nella regione Stati Uniti occidentali (Oregon) in: Account AWS 123456789012

```
"Resource": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/groups/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

Oppure, per specificare tutti i gruppi che appartengono Account AWS a un gruppo specifico Regione AWS, utilizzate il carattere jolly al posto dell'ID del gruppo:

```
"Resource": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/groups/*"
```

Alcune AWS IoT Greengrass azioni (ad esempio, alcune operazioni sugli elenchi) non possono essere eseguite su una risorsa specifica. In questi casi, è necessario utilizzare solo il carattere jolly.

```
"Resource": ""
```

Per specificare più risorse ARNs in un'istruzione, elencale tra parentesi ([]) e separale con virgole, come segue:

```
"Resource": [
  "resource-arn1",
  "resource-arn2",
  "resource-arn3"
]
```

Per ulteriori informazioni sui formati ARN, consulta [Amazon Resource Names \(ARNs\) e i namespace dei AWS servizi](#) nel. Riferimenti generali di Amazon Web Services

Chiavi di condizione

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale principale può eseguire operazioni su quali risorse, e in quali condizioni.

L'elemento `Condition`(o blocco `Condition`) consente di specificare le condizioni in cui un'istruzione è in vigore. L'elemento `Condition` è facoltativo. È possibile compilare espressioni condizionali che utilizzano [operatori di condizione](#), ad esempio uguale a o minore di, per soddisfare la condizione nella policy con i valori nella richiesta.

Se specifichi più elementi `Condition` in un'istruzione o più chiavi in un singolo elemento `Condition`, questi vengono valutati da AWS utilizzando un'operazione AND logica. Se si specificano più valori per una singola chiave di condizione, AWS valuta la condizione utilizzando un'operazione logica. OR Tutte le condizioni devono essere soddisfatte prima che le autorizzazioni dell'istruzione vengano concesse.

È possibile anche utilizzare variabili segnaposto quando specifichi le condizioni. Ad esempio, è possibile autorizzare un utente IAM ad accedere a una risorsa solo se è stata taggata con il relativo nome utente IAM. Per ulteriori informazioni, consulta [Elementi delle policy IAM: variabili e tag](#) nella Guida per l'utente di IAM.

AWS supporta chiavi di condizione globali e chiavi di condizione specifiche del servizio. Per visualizzare tutte le chiavi di condizione AWS globali, consulta le chiavi di [contesto delle condizioni AWS globali nella Guida](#) per l'utente IAM.

AWS IoT Greengrass supporta le seguenti chiavi di condizione globali.

Chiave	Descrizione
<code>aws:CurrentTime</code>	Filtra l'accesso controllando le condizioni di data/ora per la data e l'ora attuali.
<code>aws:EpochTime</code>	Filtra l'accesso controllando le condizioni di data/ora per la data e l'ora attuali e l'ora in Epoch o tempo Unix.
<code>aws:MultiFactorAuthAge</code>	Filtra l'accesso controllando quanto tempo fa (in secondi) le credenziali di sicurezza convalidate dalla Multi-Factor Authentication (MFA) che hanno effettuato la richiesta sono state create utilizzando l'MFA.

Chiave	Descrizione
<code>aws:MultiFactorAuthPresent</code>	Filtra l'accesso verificando se la Multi-Factor Authentication (MFA) è stata utilizzata per convalidare le credenziali di sicurezza provvisorie che hanno effettuato la richiesta attuale.
<code>aws:RequestTag/\${TagKey}</code>	Filtra le richieste di creazione in base alla serie di valori consentita per ciascuno dei tag obbligatori.
<code>aws:ResourceTag/\${TagKey}</code>	Filtra le operazioni in base al valore del tag associato alla risorsa.
<code>aws:SecureTransport</code>	Filtra l'accesso controllando se la richiesta è stata inviata tramite SSL.
<code>aws:TagKeys</code>	Filtra le richieste di creazione in base alla presenza di tag obbligatori nella richiesta.
<code>aws:UserAgent</code>	Filtra l'accesso dall'applicazione client del richiedente.

Per ulteriori informazioni, consulta [Chiavi di contesto delle condizioni globali AWS](#) nella Guida per l'utente IAM.

Esempi

Per visualizzare esempi di politiche AWS IoT Greengrass basate sull'identità, vedere [the section called “Esempi di policy basate su identità”](#)

Politiche basate sulle risorse per AWS IoT Greengrass

AWS IoT Greengrass [non supporta politiche basate sulle risorse](#).

Elenchi di controllo degli accessi (ACLs)

AWS IoT Greengrass non supporta [ACLs](#).

Autorizzazione basata su tag AWS IoT Greengrass

AWS IoT Greengrass non supporta l'autorizzazione basata su tag.

ruoli IAM per AWS IoT Greengrass

Un [ruolo IAM](#) è un'entità interna all'utente Account AWS che dispone di autorizzazioni specifiche.

Utilizzo di credenziali temporanee con AWS IoT Greengrass

Le credenziali temporanee vengono utilizzate per accedere con la federazione, assumere un ruolo IAM o assumere un ruolo tra account. È possibile ottenere credenziali di sicurezza temporanee chiamando operazioni AWS STS API come o. [AssumeRoleGetFederationToken](#)

Sul core di Greengrass, le credenziali temporanee per il [ruolo di gruppo](#) sono rese disponibili per le funzioni e i connettori Lambda definiti dall'utente. Se le funzioni Lambda utilizzano l' AWS SDK, non è necessario aggiungere logica per ottenere le credenziali perché l' AWS SDK lo fa per te.

Ruoli collegati ai servizi

AWS IoT Greengrass [non supporta i ruoli collegati ai servizi](#).

Ruoli dei servizi

Questa caratteristica consente a un servizio di assumere un [ruolo di servizio](#) per conto dell'utente. Questo ruolo consente al servizio di accedere alle risorse in altri servizi per completare un'azione per conto dell'utente. I ruoli dei servizi sono visualizzati nell'account IAM e sono di proprietà dell'account. Ciò significa che un amministratore IAM può modificare le autorizzazioni per questo ruolo. Tuttavia, questo potrebbe pregiudicare la funzionalità del servizio.

AWS IoT Greengrass utilizza un ruolo di servizio per accedere ad alcune delle tue AWS risorse per tuo conto. Per ulteriori informazioni, consulta [the section called “Ruolo del servizio Greengrass”](#).

Scelta di un ruolo IAM nella AWS IoT Greengrass console

Nella AWS IoT Greengrass console, potresti dover scegliere un ruolo di servizio Greengrass o un ruolo di gruppo Greengrass da un elenco di ruoli IAM nel tuo account.

- Il ruolo di servizio Greengrass consente di accedere AWS IoT Greengrass alle tue AWS risorse in altri servizi per tuo conto. In genere, non è necessario scegliere il ruolo del servizio perché la console può crearlo e configurarlo per l'utente. Per ulteriori informazioni, consulta [the section called “Ruolo del servizio Greengrass”](#).
- Il ruolo del gruppo Greengrass viene utilizzato per consentire alle funzioni e ai connettori Greengrass Lambda del gruppo di accedere alle risorse dell'utente. AWS Può anche concedere AWS IoT Greengrass le autorizzazioni per esportare flussi verso i servizi e scrivere AWS registri. CloudWatch Per ulteriori informazioni, consulta [the section called “Ruolo del gruppo Greengrass”](#).

Ruolo del servizio Greengrass

Il ruolo di servizio Greengrass è un ruolo di servizio AWS Identity and Access Management (IAM) che AWS IoT Greengrass autorizza l'accesso alle risorse AWS dei servizi per conto dell'utente. Ciò consente di AWS IoT Greengrass eseguire attività essenziali, come il recupero delle AWS Lambda funzioni e la gestione delle ombre. AWS IoT

Per consentire l'accesso AWS IoT Greengrass alle tue risorse, il ruolo di servizio Greengrass deve essere associato a te Account AWS e specificato AWS IoT Greengrass come entità affidabile. Il ruolo deve includere la politica [AWSGreengrassResourceAccessRolePolicy](#) gestita o una politica personalizzata che definisca autorizzazioni equivalenti per le AWS IoT Greengrass funzionalità utilizzate. Questa politica è gestita AWS e definisce l'insieme di autorizzazioni AWS IoT Greengrass utilizzate per accedere AWS alle risorse.

Puoi riutilizzare lo stesso ruolo del servizio Greengrass Regione AWS su tutti i server, ma devi associarlo al tuo account Regione AWS ovunque lo utilizzi. AWS IoT Greengrass L'implementazione di gruppo fallisce se il ruolo di servizio non esiste nell'attuale Account AWS e nella regione.

Le sezioni seguenti descrivono come creare e gestire il ruolo di servizio Greengrass in sala operatoria. AWS Management Console AWS CLI

- [Gestione del ruolo del servizio \(console\)](#)
- [Gestione del ruolo del servizio \(CLI\)](#)

Note

Oltre al ruolo di servizio che autorizza l'accesso a livello di servizio, è possibile assegnare un ruolo di gruppo a un gruppo. AWS IoT Greengrass Il ruolo di gruppo è un ruolo IAM separato che controlla il modo in cui le funzioni di Greengrass Lambda e i connettori del gruppo possono accedere ai servizi. AWS

Gestione del ruolo del servizio Greengrass (console)

La AWS IoT console semplifica la gestione del ruolo di servizio Greengrass. Ad esempio, quando si crea o si distribuisce un gruppo Greengrass, la console verifica se l' Account AWS utente è collegato a un ruolo di servizio Greengrass nel ruolo attualmente Regione AWS selezionato nella console.

In caso contrario, la console può creare e configurare un ruolo del servizio automaticamente. Per ulteriori informazioni, consulta [the section called “Creazione del ruolo del servizio Greengrass”](#).

È possibile utilizzare la AWS IoT console per le seguenti attività di gestione dei ruoli:

- [Individuazione del ruolo del servizio Greengrass](#)
- [Creazione del ruolo del servizio Greengrass](#)
- [Modifica del ruolo del servizio Greengrass](#)
- [Scollegare il ruolo del servizio Greengrass](#)

 Note

L'utente che ha effettuato l'accesso alla console deve disporre delle autorizzazioni per visualizzare, creare o modificare il ruolo del servizio.

Individuazione del ruolo del servizio Greengrass (console)

Utilizza i passaggi seguenti per trovare il ruolo di servizio AWS IoT Greengrass utilizzato nella versione corrente Regione AWS.

1. Dal pannello di navigazione della [AWS IoT console](#), scegli Impostazioni.
2. Scorrere fino alla sezione Greengrass service role (Ruolo del servizio Greengrass) per visualizzare il ruolo del servizio e le relative policy.

Se non vedi un ruolo del servizio, puoi consentire alla console di crearne o configurarne uno automaticamente. Per ulteriori informazioni, consulta [Creazione del ruolo del servizio Greengrass](#).

Creazione del ruolo del servizio Greengrass (console)

La console può creare e configurare automaticamente un ruolo di servizio Greengrass predefinito. Il ruolo ha le proprietà seguenti:

Proprietà	Valore
Nome	Greengrass_ServiceRole
Trusted entity (Entità attendibile)	AWS service: greengrass
Policy	AWSGreengrassResourceAccessRolePolicy

Note

Se la [configurazione del dispositivo Greengrass](#) crea il ruolo del servizio, il nome del ruolo è `GreengrassServiceRole_`*random-string*.

Quando crei o distribuisce un gruppo Greengrass dalla console, AWS IoT la console verifica se un ruolo del servizio Greengrass è associato al Account AWS tuo nel ruolo attualmente Regione AWS selezionato nella console. In caso contrario, la console richiede all'utente di consentire la lettura e la scrittura sui servizi AWS IoT Greengrass per conto dell'utente. AWS

Se concedi l'autorizzazione, la console verifica se `Greengrass_ServiceRole` esiste un ruolo denominato nel tuo. Account AWS

- Se il ruolo esiste, la console assegna il ruolo di servizio al tuo Account AWS ruolo attuale Regione AWS.
- Se il ruolo non esiste, la console crea un ruolo di servizio Greengrass predefinito e lo associa a quello corrente Account AWS . Regione AWS

Note

Se desideri creare un ruolo di servizio con politiche di ruolo personalizzate, utilizza la console IAM per creare o modificare il ruolo. Per ulteriori informazioni, consulta [Creazione di un ruolo per delegare le autorizzazioni a un AWS servizio](#) o [Modifica di un ruolo](#) nella Guida per l'utente IAM. Assicurati che il ruolo conceda autorizzazioni equivalenti alle policy `AWSGreengrassResourceAccessRolePolicy` gestite per le funzionalità e le risorse utilizzate. Ti consigliamo di includere anche le chiavi di contesto `aws:SourceArn` e della condizione `aws:SourceAccount` globale nella tua politica di fiducia per evitare il confuso

problema della sicurezza dei vicedirettori. Le chiavi di contesto delle condizioni limitano l'accesso per consentire solo le richieste che provengono dall'account specificato e dall'area di lavoro Greengrass. Per ulteriori informazioni sul problema del "confused deputy", consulta [Prevenzione del problema "confused deputy" tra servizi](#).

Se crei un ruolo di servizio, torna alla AWS IoT console e assegna il ruolo al gruppo. Puoi farlo in Greengrass service role (Ruolo di servizio Greengrass) nella pagina Settings (Impostazioni) del gruppo.

Modifica del ruolo del servizio Greengrass (console)

Usa la seguente procedura per scegliere un ruolo di servizio Greengrass diverso da assegnare al tuo Account AWS nel ruolo Regione AWS attualmente selezionato nella console.

1. Dal pannello di navigazione [AWS IoT della console](#), scegli Impostazioni.
2. In Ruolo di servizio Greengrass, scegli Cambia ruolo.

Si apre la finestra di dialogo Aggiorna ruolo di servizio Greengrass e mostra i ruoli IAM del tuo Account AWS che definisci AWS IoT Greengrass come entità attendibile.

3. Scegli il ruolo di servizio Greengrass da associare.
4. Scegli Allega ruolo.

Note

Per consentire alla console di creare automaticamente un ruolo di servizio Greengrass predefinito, scegliere Create role for me (Crea ruolo automaticamente) anziché scegliere un ruolo dall'elenco. Il link Crea ruolo per me non viene visualizzato se un ruolo denominato Greengrass_ServiceRole è nel tuo Account AWS.

Scollegare il ruolo del servizio Greengrass (console)

Usa la seguente procedura per scollegare il ruolo di servizio Greengrass dal Account AWS tuo ruolo attualmente selezionato nella console. Regione AWS Ciò revoca le autorizzazioni per l'accesso AWS IoT Greengrass ai servizi AWS nella versione corrente. Regione AWS

Important

Lo scollegamento del ruolo del servizio potrebbe interrompere le operazioni attive.

1. Dal pannello di navigazione della [AWS IoT console](#), scegli Impostazioni.
2. Nel ruolo di servizio Greengrass, scegli il ruolo Detach.
3. Nella finestra di dialogo di conferma, scegli Detach (Scollega).

Note

Se non hai più bisogno del ruolo, puoi eliminarlo nella console IAM. Per ulteriori informazioni, consulta la sezione [Eliminazione di ruoli o profili delle istanze](#) nella Guida per l'utente di IAM. Altri ruoli potrebbero consentire l'accesso AWS IoT Greengrass alle tue risorse. Per trovare tutti i ruoli che consentono di AWS IoT Greengrass assumere autorizzazioni per tuo conto, nella console IAM, nella pagina Ruoli, cerca i ruoli che includono AWS service: greengrass nella colonna Entità attendibili.

Gestione del ruolo del servizio Greengrass (CLI)

Nelle seguenti procedure, supponiamo che AWS CLI sia installato e configurato per utilizzare il tuo Account AWS ID. Per ulteriori informazioni, vedere [Installazione dell'interfaccia a riga di AWS comando](#) e [Configurazione di AWS CLI nella](#) Guida per l'AWS Command Line Interface utente.

È possibile utilizzare il AWS CLI per le seguenti attività di gestione dei ruoli:

- [Ottenimento del ruolo del servizio Greengrass](#)
- [Creazione del ruolo del servizio Greengrass](#)
- [Rimozione del ruolo del servizio Greengrass](#)

Ottenimento del ruolo del servizio Greengrass (CLI)

Utilizza la seguente procedura per scoprire se un ruolo di servizio Greengrass è associato al tuo Account AWS in un. Regione AWS

- Come ottenere il ruolo del servizio. *region* Sostituiscilo con il tuo Regione AWS (ad esempio, *us-west-2*).

```
aws Greengrass get-service-role-for-account --region region
```

Se un ruolo del servizio Greengrass è già associato all'account, vengono restituiti i seguenti metadati del ruolo.

```
{
  "AssociatedAt": "timestamp",
  "RoleArn": "arn:aws:iam::account-id:role/path/role-name"
}
```

Se non viene restituito alcun metadato del ruolo, devi creare il ruolo di servizio (se non esiste) e associarlo al tuo account in. Regione AWS

Creazione del ruolo del servizio Greengrass (CLI)

Utilizza i seguenti passaggi per creare un ruolo e associarlo al tuo Account AWS.

Per creare il ruolo di servizio utilizzando IAM

1. Crea il ruolo con una politica di fiducia che AWS IoT Greengrass consenta di assumere il ruolo. In questo esempio viene creato un ruolo denominato `Greengrass_ServiceRole`, ma è possibile utilizzare un nome diverso. Ti consigliamo di includere anche le chiavi del contesto `aws:SourceArn` e della condizione `aws:SourceAccount` globale nella tua politica di fiducia per evitare il confuso problema della sicurezza dei vicedirettori. Le chiavi di contesto delle condizioni limitano l'accesso per consentire solo le richieste che provengono dall'account specificato e dall'area di lavoro Greengrass. Per ulteriori informazioni sul problema del "confused deputy", consulta [Prevenzione del problema "confused deputy" tra servizi](#).

Linux, macOS, or Unix

```
aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-
document '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "greengrass.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "account-id"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
        }
      }
    }
  ]
}'
```

Windows command prompt

```
aws iam create-role --role-name Greengrass_ServiceRole --assume-role-
policy-document "{ \"Version\": \"2012-10-17\", \"Statement\": [{ \"Effect
\": \"Allow\", \"Principal\": { \"Service\": \"greengrass.amazonaws.com\" },
\"Action\": \"sts:AssumeRole\", \"Condition\": { \"ArnLike\": { \"aws:SourceArn
\": \"arn:aws:greengrass:region:account-id:*\" }, \"StringEquals\":
{ \"aws:SourceAccount\": \"account-id\" } } ] } }
```

2. Copiare il ruolo ARN dai metadati del ruolo nell'output. Utilizzare l'ARN per associare un ruolo all'account.
3. Collegare la policy AWSGreengrassResourceAccessRolePolicy al ruolo.

```
aws iam attach-role-policy --role-name Greengrass_ServiceRole --policy-arn
arn:aws:iam::aws:policy/service-role/AWSGreengrassResourceAccessRolePolicy
```

Per associare il ruolo di servizio al tuo Account AWS

- Associare il ruolo all'account. Sostituisci *role-arn* con il ruolo di servizio ARN e *region* con il tuo Regione AWS (ad esempio,us-west-2).

```
aws greengrass associate-service-role-to-account --role-arn role-arn --  
region region
```

Se l'operazione riesce, viene restituita la seguente risposta.

```
{  
  "AssociatedAt": "timestamp"  
}
```

Rimozione del ruolo del servizio Greengrass (CLI)

Utilizza i seguenti passaggi per dissociare il ruolo di servizio Greengrass dal tuo Account AWS

- Disassociare un ruolo del servizio dall'account. *region*Sostituiscilo con il tuo Regione AWS (ad esempio,us-west-2).

```
aws greengrass disassociate-service-role-from-account --region region
```

Se l'operazione riesce, viene restituita la seguente risposta.

```
{  
  "DisassociatedAt": "timestamp"  
}
```

Note

È necessario eliminare il ruolo di servizio se non lo si utilizza in nessuno Regione AWS. Per prima cosa utilizzare [delete-role-policy](#) per scollegare la policy gestita AWSGreengrassResourceAccessRolePolicy dal ruolo, quindi usare [delete-role](#) per eliminare il ruolo. Per ulteriori informazioni, consulta la sezione [Eliminazione di ruoli o profili delle istanze](#) nella Guida per l'utente di IAM.

Consulta anche

- [Creazione di un ruolo per delegare le autorizzazioni a un AWS servizio](#) nella IAM User Guide
- [Modifica di un ruolo nella Guida](#) per l'utente IAM
- [Eliminazione di ruoli o profili di istanza](#) nella IAM User Guide
- AWS IoT Greengrass comandi nel AWS CLI Command Reference
 - [associate-service-role-to-account](#)
 - [disassociate-service-role-from-conto](#)
 - [get-service-role-for-conto](#)
- Comandi IAM nel AWS CLI Command Reference
 - [attach-role-policy](#)
 - [create-role](#)
 - [delete-role](#)
 - [delete-role-policy](#)

Ruolo del gruppo Greengrass

Il ruolo del gruppo Greengrass è un ruolo IAM che autorizza il codice in esecuzione su un core Greengrass per accedere alle tue risorse. AWS Crei il ruolo e gestisci le autorizzazioni in AWS Identity and Access Management (IAM) e associ il ruolo al tuo gruppo Greengrass. Un gruppo Greengrass dispone di un ruolo del gruppo. Per aggiungere o modificare le autorizzazioni, puoi assegnare un ruolo diverso o modificare le politiche IAM associate al ruolo.

Il ruolo deve essere definito AWS IoT Greengrass come entità attendibile. A seconda del business case, il ruolo di gruppo potrebbe contenere policy IAM che definiscono:

- Autorizzazioni per le [funzioni Lambda](#) definite dall'utente per accedere ai servizi. AWS
- Autorizzazioni per i [connettori per accedere ai servizi](#). AWS
- Autorizzazioni per [lo stream manager](#) per esportare i flussi verso AWS IoT Analytics e Kinesis Data Streams.
- Autorizzazioni per consentire la [registrazione CloudWatch](#).

Le sezioni seguenti descrivono come collegare o scollegare un ruolo del gruppo Greengrass in AWS Management Console sala operatoria. AWS CLI

- [Gestione del ruolo del gruppo \(console\)](#)
- [Gestire il ruolo del gruppo \(CLI\)](#)

Note

Oltre al ruolo di gruppo che autorizza l'accesso dal core di Greengrass, puoi assegnare [un ruolo di servizio Greengrass](#) AWS IoT Greengrass che consente di accedere AWS alle risorse per tuo conto.

Gestione del ruolo del gruppo Greengrass (console)

È possibile utilizzare la AWS IoT console per le seguenti attività di gestione dei ruoli:

- [Individuazione del ruolo del gruppo Greengrass](#)
- [Aggiunta o modifica del ruolo del gruppo Greengrass](#)
- [Rimuovere il ruolo del gruppo Greengrass](#)

Note

L'utente che ha effettuato l'accesso alla console deve disporre delle autorizzazioni per gestire il ruolo.

Individuazione del ruolo del gruppo Greengrass (console)

Segui questi passaggi per trovare il ruolo assegnato a un gruppo Greengrass.

1. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).
2. Scegliere il gruppo target.
3. Nella pagina di configurazione del gruppo, scegli Visualizza impostazioni.

Se un ruolo è associato al gruppo, viene visualizzato in Ruolo di gruppo.

Aggiunta o modifica del ruolo del gruppo Greengrass (console)

Segui questi passaggi per scegliere un ruolo IAM dal tuo da aggiungere Account AWS a un gruppo Greengrass.

Un ruolo di gruppo ha i seguenti requisiti:

- AWS IoT Greengrass definita come entità affidabile.
- Le politiche di autorizzazione allegate al ruolo devono concedere alle AWS risorse le autorizzazioni richieste dalle funzioni e dai connettori Lambda del gruppo e dai componenti del sistema Greengrass.

Note

Ti consigliamo di includere anche le chiavi di contesto `aws:SourceArn` e della condizione `aws:SourceAccount` globale nella tua politica di fiducia per evitare il confuso problema della sicurezza secondaria. Le chiavi di contesto delle condizioni limitano l'accesso per consentire solo le richieste che provengono dall'account specificato e dall'area di lavoro Greengrass. Per ulteriori informazioni sul problema del "confused deputy", consulta [Prevenzione del problema "confused deputy" tra servizi](#).

Utilizza la console IAM per creare e configurare il ruolo e le relative autorizzazioni. Per i passaggi che creano un ruolo di esempio che consente l'accesso a una tabella Amazon DynamoDB, consulta [the section called "Configurazione del ruolo del gruppo"](#). Per i passaggi generali, consulta [Creating a role for an AWS service \(console\)](#) nella IAM User Guide.

Dopo aver configurato il ruolo, usa la AWS IoT console per aggiungere il ruolo al gruppo.

Note

Questa procedura è necessaria solo per scegliere un ruolo per il gruppo. Non è necessario dopo aver modificato le autorizzazioni del ruolo del gruppo attualmente selezionato.

1. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).
2. Scegliere il gruppo target.
3. Nella pagina di configurazione del gruppo, scegli Visualizza impostazioni.
4. In Ruolo di gruppo, scegli di aggiungere o modificare il ruolo:
 - Per aggiungere il ruolo, scegli Ruolo associato, quindi seleziona il tuo ruolo dall'elenco dei ruoli. Questi sono i ruoli Account AWS che definisci AWS IoT Greengrass come entità affidabile.
 - Per scegliere un ruolo diverso, scegli Modifica ruolo, quindi seleziona il tuo ruolo dall'elenco dei ruoli.
5. Seleziona Salva.

Rimozione del ruolo del gruppo Greengrass (console)

Attenersi alla seguente procedura per scollegare il ruolo da un gruppo Greengrass.

1. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).
2. Scegliere il gruppo target.
3. Nella pagina di configurazione del gruppo, scegli Visualizza impostazioni.
4. In Ruolo di gruppo, scegli Dissocia ruolo.
5. Nella finestra di dialogo di conferma, scegli Dissocia ruolo. Questo passaggio rimuove il ruolo dal gruppo ma non elimina il ruolo. Se desideri eliminare il ruolo, utilizza la console IAM.

Gestione del ruolo del gruppo Greengrass (CLI)

Puoi utilizzare il AWS CLI per le seguenti attività di gestione dei ruoli:

- [Ottenere il ruolo del gruppo Greengrass](#)
- [Creare il ruolo del gruppo Greengrass](#)
- [Rimuovere il ruolo del gruppo Greengrass](#)

Ottenere il ruolo del gruppo Greengrass (CLI)

Attenersi alla seguente procedura per scoprire se un gruppo Greengrass ha un ruolo associato.

1. Ottenere l'ID del gruppo di destinazione dall'elenco dei gruppi.

```
aws greengrass list-groups
```

Di seguito è riportata una risposta `list-groups` di esempio: Ogni gruppo nella risposta include una proprietà `Id` che contiene l'ID gruppo.

```
{
  "Groups": [
    {
      "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE/versions/4cbc3f07-fc5e-48c4-
a50e-7d356EXAMPLE",
      "Name": "MyFirstGroup",
      "LastUpdatedTimestamp": "2019-11-11T05:47:31.435Z",
      "LatestVersion": "4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
      "CreationTimestamp": "2019-11-11T05:47:31.435Z",
      "Id": "00dedaaa-ac16-484d-ad77-c3eedEXAMPLE",
      "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-
ac16-484d-ad77-c3eedEXAMPLE"
    },
    {
      "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE/versions/8fe9e8ec-64d1-4647-
b0b0-01dc8EXAMPLE",
      "Name": "GreenhouseSensors",
      "LastUpdatedTimestamp": "2020-01-07T19:58:36.774Z",
      "LatestVersion": "8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
      "CreationTimestamp": "2020-01-07T19:58:36.774Z",
      "Id": "036ceaf9-9319-4716-ba2a-237f9EXAMPLE",
      "Arn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE"
    },
    ...
  ]
}
```


Per ulteriori informazioni, inclusi esempi che utilizzano l'opzione `query` per filtrare i risultati, consulta [the section called “Ottenere l'ID del gruppo”](#).

2. Copiare l'Id del gruppo di destinazione dall'output.
3. Prendere il ruolo di gruppo. Sostituisci `group-id` con l'ID del gruppo target.

```
aws greengrass get-associated-role --group-id group-id
```

Se un ruolo è associato al gruppo Greengrass, vengono restituiti i seguenti metadati del ruolo.

```
{
  "AssociatedAt": "timestamp",
  "RoleArn": "arn:aws:iam::account-id:role/path/role-name"
}
```

Se il gruppo non ha un ruolo associato, viene restituito il seguente errore.

```
An error occurred (404) when calling the GetAssociatedRole operation: You need to
attach an IAM role to this deployment group.
```

Creare il ruolo del gruppo Greengrass (CLI)

Attenersi alla seguente procedura per creare un ruolo e associarlo a un gruppo Greengrass.

Per creare il ruolo di gruppo utilizzando IAM

1. Crea il ruolo con una politica di fiducia che AWS IoT Greengrass consenta di assumere il ruolo. In questo esempio viene creato un ruolo denominato `MyGreengrassGroupRole`, ma è possibile utilizzare un nome diverso. Ti consigliamo di includere anche le chiavi del contesto `aws:SourceArn` e della condizione `aws:SourceAccount` globale nella tua politica di fiducia per evitare il confuso problema della sicurezza dei vicedirettori. Le chiavi di contesto delle condizioni limitano l'accesso per consentire solo le richieste che provengono dall'account specificato e dall'area di lavoro Greengrass. Per ulteriori informazioni sul problema del "confused deputy", consulta [Prevenzione del problema "confused deputy" tra servizi](#).

Linux, macOS, or Unix

```
aws iam create-role --role-name MyGreengrassGroupRole --assume-role-policy-document '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "greengrass.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "account-id"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:greengrass:region:account-id:/greengrass/groups/group-id"
        }
      }
    }
  ]
}'
```

Windows command prompt

```
aws iam create-role --role-name MyGreengrassGroupRole --assume-role-policy-document "{\\"Version\\":\\"2012-10-17\\",\\"Statement\\":[{\\"Effect\\":\\"Allow\\",\\"Principal\\":{\\"Service\\":\\"greengrass.amazonaws.com\\"},\\"Action\\":\\"sts:AssumeRole\\",\\"Condition\\":{\\"ArnLike\\":{\\"aws:SourceArn\\":\\"arn:aws:greengrass:region:account-id:/greengrass/groups/group-id\\",\\"StringEquals\\":{\\"aws:SourceAccount\\":\\"account-id\\"}}}}]"}"
```

2. Copiare il ruolo ARN dai metadati del ruolo nell'output. Utilizzare l'ARN per associare un ruolo al gruppo.
3. Allegare le policy gestite o in linea al ruolo per supportare il proprio business case. Ad esempio, se una funzione Lambda definita dall'utente legge da Amazon S3, puoi allegare AmazonS3ReadOnlyAccess la policy gestita al ruolo.

```
aws iam attach-role-policy --role-name MyGreengrassGroupRole --policy-arn
arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess
```

In caso di esito positivo, non viene restituita alcuna risposta.

Per associare il ruolo al gruppo Greengrass

1. Ottenere l'ID del gruppo di destinazione dall'elenco dei gruppi.

```
aws greengrass list-groups
```

Di seguito è riportata una risposta `list-groups` di esempio: Ogni gruppo nella risposta include una proprietà `Id` che contiene l'ID gruppo.

```
{
  "Groups": [
    {
      "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE/versions/4cbc3f07-fc5e-48c4-
a50e-7d356EXAMPLE",
      "Name": "MyFirstGroup",
      "LastUpdatedTimestamp": "2019-11-11T05:47:31.435Z",
      "LatestVersion": "4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
      "CreationTimestamp": "2019-11-11T05:47:31.435Z",
      "Id": "00dedaaa-ac16-484d-ad77-c3eedEXAMPLE",
      "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-
ac16-484d-ad77-c3eedEXAMPLE"
    },
    {
      "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE/versions/8fe9e8ec-64d1-4647-
b0b0-01dc8EXAMPLE",
      "Name": "GreenhouseSensors",
      "LastUpdatedTimestamp": "2020-01-07T19:58:36.774Z",
      "LatestVersion": "8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
      "CreationTimestamp": "2020-01-07T19:58:36.774Z",
      "Id": "036ceaf9-9319-4716-ba2a-237f9EXAMPLE",

```

```

      "Arn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE"
    },
    ...
  ]
}

```

Per ulteriori informazioni, inclusi esempi che utilizzano l'opzione `query` per filtrare i risultati, consulta [the section called “Ottenere l'ID del gruppo”](#).

2. Copiare l'Id del gruppo di destinazione dall'output.
3. Associare il ruolo al cluster. Sostituisci *group-id* con l'ID del gruppo target e *role-arn* con l'ARN del ruolo del gruppo.

```
aws greengrass associate-role-to-group --group-id group-id --role-arn role-arn
```

Se l'operazione riesce, viene restituita la seguente risposta.

```

{
  "AssociatedAt": "timestamp"
}

```

Rimuovere il ruolo del gruppo Greengrass (CLI)

Attenersi alla seguente procedura per scollegare il ruolo del gruppo dal gruppo Greengrass.

1. Ottenere l'ID del gruppo di destinazione dall'elenco dei gruppi.

```
aws greengrass list-groups
```

Di seguito è riportata una risposta `list-groups` di esempio: Ogni gruppo nella risposta include una proprietà `Id` che contiene l'ID gruppo.

```

{
  "Groups": [
    {

```

```

        "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE/versions/4cbc3f07-fc5e-48c4-
a50e-7d356EXAMPLE",
        "Name": "MyFirstGroup",
        "LastUpdatedTimestamp": "2019-11-11T05:47:31.435Z",
        "LatestVersion": "4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
        "CreationTimestamp": "2019-11-11T05:47:31.435Z",
        "Id": "00dedaaa-ac16-484d-ad77-c3eedEXAMPLE",
        "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-
ac16-484d-ad77-c3eedEXAMPLE"
    },
    {
        "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE/versions/8fe9e8ec-64d1-4647-
b0b0-01dc8EXAMPLE",
        "Name": "GreenhouseSensors",
        "LastUpdatedTimestamp": "2020-01-07T19:58:36.774Z",
        "LatestVersion": "8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
        "CreationTimestamp": "2020-01-07T19:58:36.774Z",
        "Id": "036ceaf9-9319-4716-ba2a-237f9EXAMPLE",
        "Arn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE"
    },
    ...
]
}

```

Per ulteriori informazioni, inclusi esempi che utilizzano l'opzione `query` per filtrare i risultati, consulta [the section called “Ottenere l'ID del gruppo”](#).

2. Copiare l'Id del gruppo di destinazione dall'output.
3. Annullare associazione del ruolo dal gruppo. Sostituisci *group-id* con l'ID del gruppo target.

```
aws greengrass disassociate-role-from-group --group-id group-id
```

Se l'operazione riesce, viene restituita la seguente risposta.

```

{
  "DisassociatedAt": "timestamp"
}

```

 Note

È possibile eliminare il ruolo del gruppo se non lo si utilizza. Per prima cosa utilizzare [delete-role-policy](#) per scollegare la policy gestita dal ruolo, quindi usare [delete-role](#) per eliminare il ruolo. Per ulteriori informazioni, consulta la sezione [Eliminazione di ruoli o profili delle istanze](#) nella Guida per l'utente di IAM.

Consulta anche

- Argomenti correlati nella Guida per l'utente IAM
 - [Creazione di un ruolo per delegare le autorizzazioni a un servizio AWS](#)
 - [Modifica di un ruolo](#)
 - [Aggiungere e rimuovere i permessi di identità IAM](#)
 - [Eliminazione di ruoli o profili delle istanze](#)
- AWS IoT Greengrass comandi nel AWS CLI Command Reference
 - [list-groups](#)
 - [associate-role-to-group](#)
 - [disassociate-role-from-group](#)
 - [get-associated-role](#)
- Comandi IAM nel AWS CLI Command Reference
 - [attach-role-policy](#)
 - [create-role](#)
 - [delete-role](#)
 - [delete-role-policy](#)

Prevenzione del problema "confused deputy" tra servizi

Con "confused deputy" si intende un problema di sicurezza in cui un'entità che non dispone dell'autorizzazione per eseguire una certa operazione può costringere un'entità con più privilegi a eseguire tale operazione. Nel AWS, l'impersonificazione tra servizi può causare il problema del vice delegato confuso. La rappresentazione tra servizi può verificarsi quando un servizio (il servizio chiamante) effettua una chiamata a un altro servizio (il servizio chiamato). Il servizio chiamante può

essere manipolato per utilizzare le proprie autorizzazioni e agire sulle risorse di un altro cliente, a cui normalmente non avrebbe accesso. Per evitare ciò, AWS fornisce strumenti per poterti a proteggere i tuoi dati per tutti i servizi con entità di servizio a cui è stato concesso l'accesso alle risorse del tuo account.

Si consiglia di utilizzare [aws:SourceArn](#) le chiavi di contesto della condizione [aws:SourceAccount](#) globale nelle politiche delle risorse per limitare le autorizzazioni che AWS IoT Greengrass forniscono un altro servizio alla risorsa. Se si utilizzano entrambe le chiavi di contesto delle condizioni globali, il valore `aws:SourceAccount` e l'account nel valore `aws:SourceArn` devono utilizzare lo stesso ID account nella stessa istruzione di policy.

Il valore di `aws:SourceArn` deve essere la risorsa cliente Greengrass associata alla `sts:AssumeRole` richiesta.

Il modo più efficace per proteggersi dal problema "confused deputy" è quello di usare la chiave di contesto della condizione globale `aws:SourceArn` con l'ARN completo della risorsa. Se non si conosce l'ARN completo della risorsa o si scelgono più risorse, è necessario utilizzare la chiave di contesto della condizione globale `aws:SourceArn` con caratteri jolly (*) per le parti sconosciute dell'ARN. Ad esempio `arn:aws:greengrass:region:account-id:*`.

Per esempi di politiche che utilizzano le chiavi di contesto `aws:SourceArn` e `aws:SourceAccount` global condition, consultate i seguenti argomenti:

- [Creazione del ruolo del servizio Greengrass](#)
- [Creare il ruolo del gruppo Greengrass](#)
- [Crea e configura un ruolo di esecuzione IAM per distribuzioni di massa](#)

Esempi di policy basate su identità per AWS IoT Greengrass

Per impostazione predefinita, gli utenti e i ruoli IAM non dispongono dell'autorizzazione per creare o modificare risorse AWS IoT Greengrass. Inoltre, non possono eseguire attività utilizzando l'API AWS Management Console AWS CLI, o. AWS Un amministratore IAM deve creare policy IAM che concedono a utenti e ruoli l'autorizzazione per eseguire operazioni API specifiche sulle risorse specificate di cui hanno bisogno. L'amministratore deve quindi collegare queste policy a utenti o IAM che richiedono tali autorizzazioni.

Best practice per le policy

Le politiche basate sull'identità determinano se qualcuno può creare, accedere o eliminare AWS IoT Greengrass risorse nel tuo account. Queste operazioni possono comportare costi aggiuntivi per l' Account AWS. Quando crei o modifichi policy basate su identità, segui queste linee guida e raccomandazioni:

- Inizia con le policy AWS gestite e passa alle autorizzazioni con privilegi minimi: per iniziare a concedere autorizzazioni a utenti e carichi di lavoro, utilizza le politiche gestite che concedono le autorizzazioni per molti casi d'uso comuni. AWS Sono disponibili nel tuo Account AWS. Ti consigliamo di ridurre ulteriormente le autorizzazioni definendo politiche gestite dai AWS clienti specifiche per i tuoi casi d'uso. Per ulteriori informazioni, consulta [Policy gestite da AWS](#) o [Policy gestite da AWS per le funzioni dei processi](#) nella Guida per l'utente IAM.
- Applica le autorizzazioni con privilegio minimo: quando imposti le autorizzazioni con le policy IAM, concedi solo le autorizzazioni richieste per eseguire un'attività. È possibile farlo definendo le azioni che possono essere intraprese su risorse specifiche in condizioni specifiche, note anche come autorizzazioni con privilegi minimi. Per ulteriori informazioni sull'utilizzo di IAM per applicare le autorizzazioni, consulta [Policy e autorizzazioni in IAM](#) nella Guida per l'utente IAM.
- Condizioni d'uso nelle policy IAM per limitare ulteriormente l'accesso: per limitare l'accesso a operazioni e risorse è possibile aggiungere una condizione alle tue policy. Ad esempio, è possibile scrivere una condizione di policy per specificare che tutte le richieste devono essere inviate utilizzando SSL. Puoi anche utilizzare le condizioni per concedere l'accesso alle azioni del servizio se vengono utilizzate tramite uno specifico Servizio AWS, ad esempio AWS CloudFormation. Per ulteriori informazioni, consulta la sezione [Elementi delle policy JSON di IAM: condizione](#) nella Guida per l'utente IAM.
- Utilizzo di IAM Access Analyzer per convalidare le policy IAM e garantire autorizzazioni sicure e funzionali: IAM Access Analyzer convalida le policy nuove ed esistenti in modo che aderiscano alla sintassi della policy IAM (JSON) e alle best practice di IAM. IAM Access Analyzer offre oltre 100 controlli delle policy e consigli utili per creare policy sicure e funzionali. Per ulteriori informazioni, consulta [Convalida delle policy per il Sistema di analisi degli accessi IAM](#) nella Guida per l'utente IAM.
- Richiedi l'autenticazione a più fattori (MFA): se hai uno scenario che richiede utenti IAM o un utente root nel Account AWS tuo, attiva l'MFA per una maggiore sicurezza. Per richiedere la MFA quando vengono chiamate le operazioni API, aggiungi le condizioni MFA alle policy. Per ulteriori informazioni, consulta [Protezione dell'accesso API con MFA](#) nella Guida per l'utente IAM.

Per maggiori informazioni sulle best practice in IAM, consulta [Best practice di sicurezza in IAM](#) nella Guida per l'utente di IAM.

AWS politiche gestite per AWS IoT Greengrass

AWS IoT Greengrass mantiene le seguenti politiche AWS gestite che puoi utilizzare per concedere autorizzazioni a utenti e ruoli IAM.

Policy	Descrizione
AWSGreengrassFullAccess	Consente tutte le AWS IoT Greengrass azioni per tutte le tue AWS risorse. Questa politica è consigliata agli amministratori AWS IoT Greengrass del servizio o per scopi di test.
AWSGreengrassReadOnlyAccess	Get AWS IoT Greengrass Autorizzazioni List e azioni per tutte le tue AWS risorse.
AWSGreengrassResourceAccessRolePolicy	Consente l'accesso alle risorse da AWS servizi tra cui AWS Lambda AWS IoT Device Shadow. Questa è la policy predefinita utilizzata per il ruolo del servizio Greengrass . Questa policy è concepita per fornire una facilità generale di accesso. È possibile definire una policy personalizzata più restrittiva.
Greengrass OTAUpdate ArtifactAccess	Consente l'accesso in sola lettura agli elementi di aggiornamento over-the-air (OTA) per il software AWS IoT Greengrass Core in tutti i sistemi. Regione AWS

Esempi di policy

Nell'esempio riportato di seguito le policy definite dal cliente concedono autorizzazioni per scenari comuni.

Esempi

- [Consentire agli utenti di visualizzare le loro autorizzazioni](#)

Per informazioni su come creare una policy basata su identità IAM utilizzando questi documenti di policy JSON di esempio, consultare [Creazione di policy nella scheda JSON](#) nella Guida per l'utente IAM.

Consentire agli utenti di visualizzare le loro autorizzazioni

Questo esempio mostra in che modo è possibile creare una policy che consente agli utenti IAM di visualizzare le policy inline e gestite che sono collegate alla relativa identità utente. Questa politica include le autorizzazioni per completare questa azione sulla console o utilizzando l'API o a livello di codice. AWS CLI AWS

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupsWithUser",
        "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": "*"
    }
  ]
}
```

```
}
```

Risoluzione dei problemi di identità e accesso per AWS IoT Greengrass

Utilizza le seguenti informazioni per aiutarti a diagnosticare e risolvere i problemi più comuni che potresti riscontrare quando lavori con AWS IoT Greengrass IAM.

Problemi

- [Non sono autorizzato a eseguire un'azione in AWS IoT Greengrass](#)
- [Errore: Greengrass non è autorizzato ad assumere il ruolo di servizio associato a questo account, oppure l'errore: Non riuscito: il ruolo di servizio TES non è associato a questo account.](#)
- [<account-id><role-name><region>Errore: autorizzazione negata durante il tentativo di utilizzare il ruolo arn:aws:iam: :role/ per accedere a s3 url https://-greengrass-updates.s3.<region><architecture><distribution-version>.amazonaws.com/core/ /greengrass-core- .tar.gz.](#)
- [La shadow del dispositivo non si sincronizza con il cloud.](#)
- [Non sono autorizzato a eseguire iam: PassRole](#)
- [Sono un amministratore e voglio consentire ad altri di accedere AWS IoT Greengrass](#)
- [Voglio consentire a persone esterne a me di accedere Account AWS alle mie risorse AWS IoT Greengrass](#)

Per un aiuto generale nella risoluzione dei problemi, consulta [Risoluzione dei problemi](#).

Non sono autorizzato a eseguire un'azione in AWS IoT Greengrass

Se ricevi un errore che indica che non sei autorizzato a eseguire un'operazione, devi contattare il tuo amministratore per ricevere assistenza. L'amministratore è la persona che ti ha fornito il nome utente e la password.

Il seguente errore di esempio si verifica quando l'utente mateojackson IAM tenta di visualizzare i dettagli su una versione di definizione principale, ma non dispone greengrass:GetCoreDefinitionVersion delle autorizzazioni.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
greengrass:GetCoreDefinitionVersion on resource: resource: arn:aws:greengrass:us-
west-2:123456789012:/greengrass/definition/cores/78cd17f3-bc68-ee18-47bd-5bda5EXAMPLE/
versions/368e9ffa-4939-6c75-859c-0bd4cEXAMPLE
```

In questo caso, Mateo richiede al suo amministratore di aggiornare le policy per poter accedere alla risorsa `arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/cores/78cd17f3-bc68-ee18-47bd-5bda5EXAMPLE/versions/368e9ffa-4939-6c75-859c-0bd4cEXAMPLE` utilizzando l'azione `greengrass:GetCoreDefinitionVersion`.

Errore: Greengrass non è autorizzato ad assumere il ruolo di servizio associato a questo account, oppure l'errore: Non riuscito: il ruolo di servizio TES non è associato a questo account.

Soluzione: è possibile visualizzare questo errore quando la distribuzione ha esito negativo. Verifica che un ruolo di servizio Greengrass sia associato al tuo Account AWS ruolo attuale. Regione AWS Per ulteriori informazioni, consulta [the section called “Gestione del ruolo del servizio \(CLI\)”](#) o [the section called “Gestione del ruolo del servizio \(console\)”](#).

`<account-id><role-name><region>`Errore: autorizzazione negata durante il tentativo di utilizzare il ruolo `arn:aws:iam: :role/` per accedere a s3 url `https://-greengrass-updates.s3. <region><architecture><distribution-version>.amazonaws.com/core/ / greengrass-core- .tar.gz`.

Soluzione: potresti visualizzare questo errore quando un aggiornamento (OTA) fallisce. over-the-air Nella politica del ruolo del firmatario, aggiungi l'obiettivo Regione AWS comeResource. Questo ruolo di firmatario viene utilizzato per preassegnare l'URL S3 per l'aggiornamento del software. AWS IoT Greengrass Per ulteriori informazioni, consulta [Ruolo firmatario URL S3](#).

La shadow del dispositivo non si sincronizza con il cloud.

Soluzione: assicurati di disporre AWS IoT Greengrass delle autorizzazioni `iot:UpdateThingShadow` e delle `iot:GetThingShadow` azioni per il ruolo del [servizio Greengrass](#). Se il ruolo del servizio usa la policy gestita `AWSGreengrassResourceAccessRolePolicy`, queste autorizzazioni sono incluse per impostazione predefinita.

Per informazioni, consulta [Risoluzione dei problemi di timeout della sincronizzazione shadow](#).

Di seguito sono riportati i problemi generali relativi a IAM che potresti riscontrare quando lavori con. AWS IoT Greengrass

Non sono autorizzato a eseguire iam:PassRole

Se ricevi un errore che indica che non sei autorizzato a eseguire l'operazione `iam:PassRole`, le tue policy devono essere aggiornate per poter passare un ruolo a AWS IoT Greengrass.

Alcuni Servizi AWS consentono di passare un ruolo esistente a quel servizio invece di creare un nuovo ruolo di servizio o un ruolo collegato al servizio. Per eseguire questa operazione, è necessario disporre delle autorizzazioni per trasmettere il ruolo al servizio.

L'errore di esempio seguente si verifica quando un utente IAM denominato `marymajor` cerca di utilizzare la console per eseguire un'operazione in AWS IoT Greengrass. Tuttavia, l'operazione richiede che il servizio disponga delle autorizzazioni concesse da un ruolo di servizio. Mary non dispone delle autorizzazioni per passare il ruolo al servizio.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

In questo caso, le policy di Mary devono essere aggiornate per poter eseguire l'operazione `iam:PassRole`.

Se hai bisogno di aiuto, contatta il tuo AWS amministratore. L'amministratore è la persona che ti ha fornito le credenziali di accesso.

Sono un amministratore e voglio consentire ad altri di accedere AWS IoT Greengrass

Per consentire ad altri di accedere AWS IoT Greengrass, devi concedere l'autorizzazione alle persone o alle applicazioni che necessitano dell'accesso. Se si utilizza AWS IAM Identity Center per gestire persone e applicazioni, si assegnano set di autorizzazioni a utenti o gruppi per definirne il livello di accesso. I set di autorizzazioni creano e assegnano automaticamente le policy IAM ai ruoli IAM associati alla persona o all'applicazione. Per ulteriori informazioni, consulta [Set di autorizzazioni](#) nella Guida per l'AWS IAM Identity Center utente.

Se non utilizzi IAM Identity Center, devi creare entità IAM (utenti o ruoli) per le persone o le applicazioni che necessitano di accesso. Dovrai quindi collegare all'entità una policy che conceda le autorizzazioni corrette in AWS IoT Greengrass. Dopo aver concesso le autorizzazioni, fornisci le credenziali all'utente o allo sviluppatore dell'applicazione. Utilizzeranno tali credenziali per accedere. AWS Per ulteriori informazioni sulla creazione di utenti, gruppi, policy e autorizzazioni IAM, consulta [IAM Identities](#) and [Policies and permissions in IAM nella IAM](#) User Guide.

Voglio consentire a persone esterne a me di accedere Account AWS alle mie risorse AWS IoT Greengrass

Puoi creare un ruolo IAM che gli utenti di altri account o persone esterne all'organizzazione possano utilizzare per accedere alle tue AWS risorse. Puoi specificare chi è attendibile per l'assunzione del ruolo. Per ulteriori informazioni, consulta [Fornire l'accesso a un utente IAM in un altro Account AWS di tua](#) proprietà e [Fornire l'accesso agli account Amazon Web Services di proprietà di terze parti](#) nella IAM User Guide.

AWS IoT Greengrass non supporta l'accesso tra account in base a politiche basate sulle risorse o liste di controllo degli accessi (). ACLs

Convalida della conformità per AWS IoT Greengrass

Per sapere se un Servizio AWS programma rientra nell'ambito di specifici programmi di conformità, consulta Servizi AWS la sezione [Scope by Compliance Program Servizi AWS](#) e scegli il programma di conformità che ti interessa. Per informazioni generali, consulta Programmi di [AWS conformità Programmi](#) di di .

È possibile scaricare report di audit di terze parti utilizzando AWS Artifact. Per ulteriori informazioni, consulta [Scaricamento dei report in AWS Artifact](#) .

La vostra responsabilità di conformità durante l'utilizzo Servizi AWS è determinata dalla sensibilità dei dati, dagli obiettivi di conformità dell'azienda e dalle leggi e dai regolamenti applicabili. AWS fornisce le seguenti risorse per contribuire alla conformità:

- [Governance e conformità per la sicurezza](#): queste guide all'implementazione di soluzioni illustrano considerazioni relative all'architettura e i passaggi per implementare le funzionalità di sicurezza e conformità.
- [Riferimenti sui servizi conformi ai requisiti HIPAA](#): elenca i servizi HIPAA idonei. Non tutti Servizi AWS sono idonei alla normativa HIPAA.
- [AWS Risorse per](#) la per la conformità: questa raccolta di cartelle di lavoro e guide potrebbe essere valida per il tuo settore e la tua località.
- [AWS Guide alla conformità dei clienti](#): comprendi il modello di responsabilità condivisa attraverso la lente della conformità. Le guide riassumono le migliori pratiche per la protezione Servizi AWS e mappano le linee guida per i controlli di sicurezza su più framework (tra cui il National Institute of Standards and Technology (NIST), il Payment Card Industry Security Standards Council (PCI) e l'International Organization for Standardization (ISO)).

- [Valutazione delle risorse con regole](#) nella Guida per gli AWS Config sviluppatori: il AWS Config servizio valuta la conformità delle configurazioni delle risorse alle pratiche interne, alle linee guida e alle normative del settore.
- [AWS Security Hub](#)— Ciò Servizio AWS fornisce una visione completa dello stato di sicurezza interno. AWS La Centrale di sicurezza utilizza i controlli di sicurezza per valutare le risorse AWS e verificare la conformità agli standard e alle best practice del settore della sicurezza. Per un elenco dei servizi e dei controlli supportati, consulta la pagina [Documentazione di riferimento sui controlli della Centrale di sicurezza](#).
- [Amazon GuardDuty](#): Servizio AWS rileva potenziali minacce ai tuoi carichi di lavoro Account AWS, ai contenitori e ai dati monitorando l'ambiente alla ricerca di attività sospette e dannose. GuardDuty può aiutarti a soddisfare vari requisiti di conformità, come lo standard PCI DSS, soddisfacendo i requisiti di rilevamento delle intrusioni imposti da determinati framework di conformità.
- [AWS Audit Manager](#)— Ciò Servizio AWS consente di verificare continuamente l' AWS utilizzo per semplificare la gestione del rischio e la conformità alle normative e agli standard di settore.

Resilienza in AWS IoT Greengrass

L'infrastruttura AWS globale è costruita attorno alle regioni e alle zone di disponibilità di Amazon Web Services. Regione AWS Ciascuna fornisce più zone di disponibilità fisicamente separate e isolate, collegate con reti a bassa latenza, ad alto throughput e altamente ridondanti. Con le zone di disponibilità, puoi progettare e gestire applicazioni e database che eseguono automaticamente il failover tra zone di disponibilità senza interruzioni. Le zone di disponibilità sono più disponibili, tolleranti ai guasti e scalabili rispetto alle infrastrutture a data center singolo o multiplo tradizionali.

Per ulteriori informazioni sulle regioni e le zone di disponibilità di Amazon Web Services, consulta [AWS Global Infrastructure](#).

Oltre all'infrastruttura AWS globale, AWS IoT Greengrass offre diverse funzionalità per supportare le esigenze di resilienza e backup dei dati.

- Se il core perde la connettività Internet, i dispositivi client possono continuare a comunicare sulla rete locale.
- È possibile configurare il core per archiviare i messaggi non elaborati destinati alle Cloud AWS destinazioni in una cache di archiviazione locale anziché in memoria. La cache di archiviazione locale può persistere anche dopo i riavvii principali (ad esempio, dopo una distribuzione di gruppo o il riavvio di un dispositivo), quindi AWS IoT Greengrass può continuare a elaborare i messaggi

destinati a. AWS IoT Core Per ulteriori informazioni, consulta [the section called “Coda di messaggi MQTT”](#).

- È possibile configurare il core per stabilire una sessione persistente con il broker di messaggi. AWS IoT Core Ciò consente al core di ricevere messaggi inviati mentre il core è offline. Per ulteriori informazioni, consulta [the section called “Sessioni persistenti MQTT con AWS IoT Core”](#).
- È possibile configurare un gruppo Greengrass per scrivere i log sul file system locale e su Logs. CloudWatch Se il core perde la connettività, la registrazione locale può continuare, ma i log CloudWatch vengono inviati con un numero limitato di tentativi. Una volta esauriti i tentativi, l'evento viene rimosso. È necessario anche conoscere le [limitazioni di registrazione](#).
- Puoi creare funzioni Lambda che leggono i flussi di [stream manager](#) e inviano i dati a destinazioni di archiviazione locali.

Sicurezza dell'infrastruttura in AWS IoT Greengrass

In quanto servizio gestito, AWS IoT Greengrass è protetto dalla sicurezza di rete AWS globale. Per informazioni sui servizi AWS di sicurezza e su come AWS protegge l'infrastruttura, consulta [AWS Cloud Security](#). Per progettare il tuo AWS ambiente utilizzando le migliori pratiche per la sicurezza dell'infrastruttura, vedi [Infrastructure Protection](#) in Security Pillar AWS Well-Architected Framework.

Utilizzate chiamate API AWS pubblicate per accedere AWS IoT Greengrass attraverso la rete. I client devono supportare quanto segue:

- Transport Layer Security (TLS). È richiesto TLS 1.2 ed è consigliato TLS 1.3.
- Suite di cifratura con Perfect Forward Secrecy (PFS), ad esempio Ephemeral Diffie-Hellman (DHE) o Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). La maggior parte dei sistemi moderni, come Java 7 e versioni successive, supporta tali modalità.

Inoltre, le richieste devono essere firmate utilizzando un ID chiave di accesso e una chiave di accesso segreta associata a un principale IAM. O puoi utilizzare [AWS Security Token Service](#) (AWS STS) per generare credenziali di sicurezza temporanee per sottoscrivere le richieste.

In un AWS IoT Greengrass ambiente, i dispositivi utilizzano certificati X.509 e chiavi crittografiche per connettersi e autenticarsi a. Cloud AWS Per ulteriori informazioni, consulta [the section called “Autenticazione e autorizzazione del dispositivo”](#).

Analisi della configurazione e delle vulnerabilità in AWS IoT Greengrass

Gli ambienti IoT possono essere costituiti da un numero elevato di dispositivi con funzionalità diverse, usati per lunghi periodi di tempo e distribuiti in varie aree geografiche. Queste caratteristiche rendono la configurazione di un dispositivo complessa e soggetta a errori. E poiché i dispositivi presentano spesso vincoli di potenza di elaborazione, memoria e capacità di storage, ciò limita l'uso della crittografia e di altre forme di sicurezza nei dispositivi stessi. I dispositivi, inoltre, usano spesso software con vulnerabilità note. La combinazione di questi fattori rende i dispositivi IoT un facile bersaglio per gli hacker e ne rende difficile la protezione continuativa.

AWS IoT Device Defender affronta queste sfide fornendo strumenti per identificare i problemi di sicurezza e le deviazioni dalle migliori pratiche. È possibile utilizzarli AWS IoT Device Defender per analizzare, controllare e monitorare i dispositivi connessi per rilevare comportamenti anomali e mitigare i rischi per la sicurezza. AWS IoT Device Defender può controllare i dispositivi per garantire che rispettino le migliori pratiche di sicurezza e rilevare comportamenti anomali sui dispositivi. Offre la possibilità di applicare policy di sicurezza coerenti in tutti i dispositivi e di rispondere rapidamente quando i dispositivi vengono compromessi. In connessione con AWS IoT Core, AWS IoT Greengrass genera un [client prevedibile IDs](#) che è possibile utilizzare con le funzionalità. AWS IoT Device Defender Per ulteriori informazioni, consulta la sezione [AWS IoT Device Defender](#) nella Guida per gli sviluppatori di AWS IoT Core .

Negli AWS IoT Greengrass ambienti, è necessario tenere presente le seguenti considerazioni:

- È propria responsabilità proteggere i dispositivi fisici, il file system sui dispositivi e la rete locale.
- AWS IoT Greengrass [non impone l'isolamento di rete per le funzioni Lambda definite dall'utente, indipendentemente dal fatto che vengano eseguite o meno in un contenitore Greengrass](#). Pertanto, è possibile che le funzioni Lambda comunichino con qualsiasi altro processo in esecuzione nel sistema o all'esterno tramite la rete.

Se perdi il controllo di un dispositivo core Greengrass e desideri impedire ai dispositivi client di trasmettere dati al core, procedi come segue:

1. Rimuovere il core di Greengrass dal gruppo Greengrass.
2. Ruotare il certificato CA del gruppo. Nella AWS IoT console, puoi ruotare il certificato CA nella pagina Impostazioni del gruppo. Nell' AWS IoT Greengrass API, puoi utilizzare l'[CreateGroupCertificateAuthority](#) azione.

Si consiglia inoltre di utilizzare la crittografia completa del disco se il disco rigido del dispositivo core è vulnerabile al furto.

AWS IoT Greengrass e endpoint VPC di interfaccia (AWS PrivateLink)

Puoi stabilire una connessione privata tra il tuo VPC e il piano di AWS IoT Greengrass controllo creando un endpoint VPC di interfaccia. È possibile utilizzare questo endpoint per gestire gruppi, funzioni Lambda, distribuzioni e altre risorse del servizio. AWS IoT Greengrass Gli endpoint di interfaccia sono alimentati da [AWS PrivateLink](#), una tecnologia che consente di accedere AWS IoT Greengrass APIs in modo privato senza un gateway Internet, un dispositivo NAT, una connessione VPN o una connessione AWS Direct Connect. Le istanze del tuo VPC non necessitano di indirizzi IP pubblici con cui comunicare. AWS IoT Greengrass APIs Il traffico tra il tuo VPC e AWS IoT Greengrass non esce dalla rete Amazon.

Note

Al momento, non puoi configurare i dispositivi core Greengrass in modo che funzionino completamente all'interno del tuo VPC.

Ogni endpoint dell'interfaccia è rappresentato da una o più [interfacce di rete elastiche](#) nelle sottoreti.

Per ulteriori informazioni, consultare [Endpoint VPC di interfaccia \(AWS PrivateLink\)](#) nella Guida per l'utente di Amazon VPC.

Argomenti

- [Considerazioni sugli endpoint AWS IoT Greengrass VPC](#)
- [Crea un endpoint VPC di interfaccia per AWS IoT Greengrass le operazioni del piano di controllo](#)
- [Creazione di una policy per gli endpoint VPC per AWS IoT Greengrass](#)

Considerazioni sugli endpoint AWS IoT Greengrass VPC

Prima di configurare un endpoint VPC di interfaccia per AWS IoT Greengrass, consulta le [proprietà e le limitazioni dell'endpoint dell'interfaccia nella](#) Amazon VPC User Guide. Inoltre, tieni presente le seguenti considerazioni:

- AWS IoT Greengrass supporta l'esecuzione di chiamate a tutte le azioni dell'API del piano di controllo dal tuo VPC. Il piano di controllo include operazioni come [CreateDeployment](#) [StartBulkDeployment](#). Il piano di controllo non include operazioni come [GetDeployment](#) [Discover](#), che sono operazioni sul piano dati.
- Gli endpoint VPC per non AWS IoT Greengrass sono attualmente supportati nelle regioni della Cina AWS .

Crea un endpoint VPC di interfaccia per AWS IoT Greengrass le operazioni del piano di controllo

Puoi creare un endpoint VPC per il piano di AWS IoT Greengrass controllo utilizzando la console Amazon VPC o (). AWS Command Line Interface AWS CLI Per ulteriori informazioni, consulta [Creazione di un endpoint dell'interfaccia](#) nella Guida per l'utente di Amazon VPC.

Crea un endpoint VPC per AWS IoT Greengrass utilizzare il seguente nome di servizio:

- com.amazonaws. *region*. erba verde

Se abiliti il DNS privato per l'endpoint, puoi effettuare richieste API AWS IoT Greengrass utilizzando il nome DNS predefinito per la regione, ad esempio. `greengrass.us-east-1.amazonaws.com` DNS privato è abilitato per impostazione predefinita.

Per ulteriori informazioni, consulta [Accesso a un servizio tramite un endpoint dell'interfaccia](#) in Guida per l'utente di Amazon VPC.

Creazione di una policy per gli endpoint VPC per AWS IoT Greengrass

Puoi allegare una policy per gli endpoint all'endpoint VPC che controlla l'accesso AWS IoT Greengrass alle operazioni del piano di controllo. La policy specifica le informazioni riportate di seguito:

- Il principale che può eseguire operazioni.
- Le azioni che l'entità può eseguire.
- Le risorse su cui il preside può eseguire azioni.

Per ulteriori informazioni, consulta [Controllo degli accessi ai servizi con endpoint VPC](#) nella Guida per l'utente di Amazon VPC.

Example Esempio: policy degli endpoint VPC per le azioni AWS IoT Greengrass

Di seguito è riportato un esempio di policy sugli endpoint per AWS IoT Greengrass. Se associata a un endpoint, questa politica consente l'accesso alle AWS IoT Greengrass azioni elencate per tutti i principali su tutte le risorse.

```
{
  "Statement": [
    {
      "Principal": "*",
      "Effect": "Allow",
      "Action": [
        "greengrass:CreateDeployment",
        "greengrass:StartBulkDeployment"
      ],
      "Resource": "*"
    }
  ]
}
```

Best practice di sicurezza per AWS IoT Greengrass

Questo argomento contiene le migliori pratiche di sicurezza per AWS IoT Greengrass.

Concedere autorizzazioni minime possibili

Segui il principio del privilegio minimo utilizzando il set minimo di autorizzazioni nei ruoli IAM. Limita l'uso del * wildcard per le Resource proprietà Action and nelle tue policy IAM. Invece, dichiarare un insieme finito di operazioni e risorse quando possibile. Per ulteriori informazioni su privilegi minimi e altre best practice sulle policy, consulta [the section called "Best practice per le policy"](#).

La best practice con privilegi minimi si applica anche alle AWS IoT politiche allegate ai dispositivi core e client Greengrass.

Non codificare le credenziali nelle funzioni Lambda

Non codificare le credenziali nelle funzioni Lambda definite dall'utente. Per proteggere meglio le credenziali:

- Per interagire con AWS i servizi, definisci le autorizzazioni per azioni e risorse specifiche nel ruolo del [gruppo Greengrass](#).

- Utilizzare [i segreti locali](#) per archiviare le credenziali. Oppure, se la funzione utilizza l' AWS SDK, utilizza le credenziali della catena di provider di credenziali predefinita.

Non registrare informazioni riservate

È necessario impedire la registrazione delle credenziali e di altre informazioni di identificazione personale (PII). Ti consigliamo di implementare le seguenti misure di protezione anche se l'accesso ai log locali su un dispositivo principale richiede i privilegi di root e l'accesso ai log richiede le autorizzazioni IAM. CloudWatch

- Non utilizzare informazioni riservate nei percorsi argomento MQTT.
- Non utilizzare informazioni riservate nei nomi, nei tipi e negli attributi dei dispositivi (oggetti) nel registro. AWS IoT Core
- Non registrare informazioni sensibili nelle funzioni Lambda definite dall'utente.
- Non utilizzare informazioni sensibili nei nomi e nelle risorse IDs di Greengrass:
 - Connectors (Connettori)
 - Core
 - Dispositivi
 - Funzioni
 - Gruppi
 - Loggers
 - Risorse (locale, machine learning o segreto)
 - Sottoscrizioni

Creare sottoscrizioni mirate

Gli abbonamenti controllano il flusso di informazioni in un gruppo Greengrass definendo come i messaggi vengono scambiati tra servizi, dispositivi e funzioni Lambda. Per garantire che un'applicazione possa eseguire solo ciò che è previsto che faccia, le sottoscrizioni devono consentire agli editori di inviare messaggi solo ad argomenti specifici e limitare i sottoscrittori a ricevere messaggi solo da argomenti necessari per la loro funzionalità.

Tenere sincronizzato l'orologio del dispositivo

È importante avere un orario preciso sul dispositivo. I certificati X.509 hanno data e ora di scadenza. L'orologio sul dispositivo viene utilizzato per verificare che un certificato server sia ancora valido. Gli orologi dei dispositivi possono andare alla deriva nel tempo o le batterie possono scaricarsi.

Per ulteriori informazioni, consulta la best practice [Tenere sincronizzato l'orologio del dispositivo](#) nella Guida per sviluppatori AWS IoT Core .

Gestione dell'autenticazione dei dispositivi con il core Greengrass

I dispositivi client possono eseguire [FreerTOS](#) o utilizzare [Device SDK AWS IoT Greengrass o Discovery API per ottenere informazioni di rilevamento utilizzate per connettersi e autenticarsi con il core AWS IoT dello](#) stesso gruppo Greengrass. Le informazioni di individuazione includono:

- Informazioni sulla connettività per il core Greengrass che appartiene allo stesso gruppo Greengrass del dispositivo client. Queste informazioni includono l'indirizzo host e il numero di porta di ciascun endpoint per il dispositivo core.
- Il certificato CA del gruppo utilizzato per firmare il certificato del server MQTT locale. I dispositivi client utilizzano il certificato CA di gruppo per convalidare il certificato del server MQTT presentato dal core.

Di seguito sono riportate le migliori pratiche per i dispositivi client per gestire l'autenticazione reciproca con un core Greengrass. Queste procedure possono contribuire a ridurre i rischi in caso di compromissione del dispositivo core.

Convalidare il certificato del server MQTT locale per ogni connessione.

I dispositivi client devono convalidare il certificato del server MQTT presentato dal core ogni volta che stabiliscono una connessione con il core. Questa convalida riguarda il lato del dispositivo client dell'autenticazione reciproca tra un dispositivo principale e i dispositivi client. I dispositivi client devono essere in grado di rilevare un errore e interrompere la connessione.

Non eseguire le informazioni di individuazione hardcode.

I dispositivi client devono fare affidamento sulle operazioni di rilevamento per ottenere le informazioni di connettività di base e il certificato CA di gruppo, anche se il core utilizza un indirizzo IP statico. I dispositivi client non devono codificare queste informazioni di rilevamento.

Aggiornare periodicamente le informazioni di individuazione.

I dispositivi client devono eseguire periodicamente il rilevamento per aggiornare le informazioni di connettività di base e il certificato CA di gruppo. Consigliamo ai dispositivi client di aggiornare queste informazioni prima di stabilire una connessione con il core. Poiché intervalli più brevi tra le operazioni di rilevamento possono ridurre al minimo il potenziale tempo di esposizione, consigliamo di disconnettere e riconnettersi periodicamente i dispositivi client per attivare l'aggiornamento.

Se perdi il controllo di un dispositivo core Greengrass e desideri impedire ai dispositivi client di trasmettere dati al core, procedi come segue:

1. Rimuovere il core di Greengrass dal gruppo Greengrass.
2. Ruotare il certificato CA del gruppo. Nella AWS IoT console, puoi ruotare il certificato CA nella pagina Impostazioni del gruppo. Nell' AWS IoT Greengrass API, puoi utilizzare l'[CreateGroupCertificateAuthority](#) azione.

Si consiglia inoltre di utilizzare la crittografia completa del disco se il disco rigido del dispositivo core è vulnerabile al furto.

Per ulteriori informazioni, consulta [the section called “Autenticazione e autorizzazione del dispositivo”](#).

Consulta anche

- [Le migliori pratiche di sicurezza sono AWS IoT Core riportate nella](#) Guida per AWS IoT gli sviluppatori
- [Dieci regole d'oro di sicurezza per le soluzioni Industrial IoT](#) sull'Internet of Things sul blog AWS ufficiale

Registrazione e monitoraggio AWS IoT Greengrass

Il monitoraggio è un elemento importante per mantenere l'affidabilità, la disponibilità e le prestazioni delle AWS IoT Greengrass vostre AWS soluzioni. È necessario raccogliere i dati di monitoraggio da tutte le parti della AWS soluzione in modo da poter eseguire più facilmente il debug di un errore multipunto, se si verifica. Prima di iniziare il monitoraggio AWS IoT Greengrass, è necessario creare un piano di monitoraggio che includa le risposte alle seguenti domande:

- Quali sono gli obiettivi del monitoraggio?
- Quali risorse verranno monitorate?
- Con quale frequenza eseguirai il monitoraggio di queste risorse?
- Quali strumenti di monitoraggio verranno usati?
- Chi eseguirà i processi di monitoraggio?
- Chi deve ricevere una notifica quando si verifica un problema?

Strumenti di monitoraggio

AWS fornisce strumenti che è possibile utilizzare per il monitoraggio AWS IoT Greengrass. Alcuni di questi strumenti possono essere configurati per il monitoraggio automatico delle applicazioni. Alcuni degli strumenti richiedono l'intervento manuale. Si consiglia di automatizzare il più possibile i processi di monitoraggio.

È possibile utilizzare i seguenti strumenti di monitoraggio automatizzato per monitorare AWS IoT Greengrass e segnalare i problemi:

- Amazon CloudWatch Logs: monitora, archivia e accedi ai tuoi file di registro da AWS CloudTrail o altre fonti. Per ulteriori informazioni, consulta [Monitoring log files](#) nella Amazon CloudWatch User Guide.
- AWS CloudTrail Monitoraggio dei log: condividi i file di CloudTrail log tra account, monitora i file di log in tempo reale inviandoli a CloudWatch Logs, scrivi applicazioni di elaborazione dei log in Java e verifica che i file di log non siano cambiati dopo la consegna da parte di. CloudTrail Per ulteriori informazioni, consulta [Lavorare con i file di CloudTrail registro nella Guida](#) per l'AWS CloudTrail utente.
- Amazon EventBridge: utilizza le regole EventBridge degli eventi per ricevere notifiche sui cambiamenti di stato per le implementazioni del gruppo Greengrass o le chiamate API registrate.

CloudTrail Per ulteriori informazioni, consulta [the section called “Ottenere le notifiche di distribuzione”](#) o [Cos'è Amazon EventBridge?](#) nella Amazon EventBridge User Guide.

- Telemetria sanitaria del sistema Greengrass: iscriviti per ricevere i dati di telemetria inviati dal core Greengrass. Per ulteriori informazioni, consulta [the section called “Raccolta di dati di telemetria sanitaria del sistema”](#).
- Controllo sanitario locale: utilizza l'integrità APIs per ottenere un'istantanea dello stato dei processi locali sul dispositivo principale. AWS IoT Greengrass Per ulteriori informazioni, consulta [the section called “Chiamata dell'API di controllo sanitario locale”](#).

Consulta anche

- [the section called “Monitoraggio con i log AWS IoT Greengrass ”](#)
- [the section called “Registrazione delle chiamate AWS IoT Greengrass API con AWS CloudTrail”](#)
- [the section called “Ottenere le notifiche di distribuzione”](#)

Monitoraggio con AWS IoT Greengrass registri

AWS IoT Greengrass è costituito dal servizio cloud e dal software AWS IoT Greengrass Core. Il software AWS IoT Greengrass Core può scrivere log su Amazon CloudWatch e sul file system locale del dispositivo principale. Le funzioni e i connettori Lambda in esecuzione sul core possono anche scrivere log in CloudWatch Logs e nel file system locale. Puoi utilizzare i log di eventi per monitorare e risolvere i problemi. Tutte le voci di AWS IoT Greengrass registro includono un timestamp, un livello di registro e informazioni sull'evento. Le modifiche alle impostazioni di registrazione diventano effettive dopo la distribuzione del gruppo.

La registrazione è configurata a livello del gruppo. Per i passaggi che mostrano come configurare la registrazione per un gruppo Greengrass, consulta [the section called “Configurare la registrazione per AWS IoT Greengrass”](#).

Accesso ai log CloudWatch

Se configuri CloudWatch la registrazione, puoi visualizzare i log nella pagina Logs della console Amazon. CloudWatch I gruppi di log per i AWS IoT Greengrass log utilizzano le seguenti convenzioni di denominazione:

```
/aws/greengrass/GreengrassSystem/greengrass-system-component-name
```

```
/aws/greengrass/Lambda/aws-region/account-id/lambda-function-name
```

Ogni gruppo di log contiene flussi di log che utilizzano la seguente convenzione di denominazione:

```
date/account-id/greengrass-group-id/name-of-core-that-generated-log
```

Le seguenti considerazioni si applicano quando si utilizza Logs: CloudWatch

- I log vengono inviati a CloudWatch Logs con un numero limitato di tentativi in caso di mancanza di connettività Internet. Una volta esauriti i tentativi, l'evento viene rimosso.
- Si applicano limitazioni relative alla transazione, alla memoria e di altro tipo. Per ulteriori informazioni, consulta [the section called "Limitazioni di registrazione"](#).
- Il ruolo del tuo gruppo Greengrass deve consentire la scrittura AWS IoT Greengrass su Logs. CloudWatch Per concedere le autorizzazioni, [incorpora la seguente policy inline](#) nel ruolo del gruppo.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Resource": [
        "arn:aws:logs:*:*:*"
      ]
    }
  ]
}
```

 Note

Puoi concedere accesso più granulare alle risorse di log. Per ulteriori informazioni, consulta [Using Identity-Based Policy \(IAM policies\) for CloudWatch Logs nella Amazon User Guide](#).
CloudWatch

Il ruolo di gruppo è un ruolo IAM che crei e colleghi al tuo gruppo Greengrass. Puoi utilizzare la console o l' AWS IoT Greengrass API per gestire il ruolo di gruppo.

Utilizzo della console

1. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).
2. Scegliere il gruppo target.
3. Scegli Visualizza impostazioni. In Ruolo di gruppo, puoi visualizzare, associare o dissociare il ruolo di gruppo.

Per i passaggi che illustrano come collegare il ruolo del gruppo, consulta [ruolo del gruppo](#).

Utilizzo della CLI

- Per trovare il ruolo del gruppo, usa il [get-associated-role](#) comando.
- Per assegnare il ruolo del gruppo, usa il [associate-role-to-group](#) comando.
- Per rimuovere il ruolo del gruppo, usa il [disassociate-role-from-group](#) comando.

Per informazioni su come ottenere l'ID del gruppo da utilizzare con questi comandi, consulta [the section called “Ottenere l'ID del gruppo”](#).

Accesso ai log del file system

Se configuri la registrazione del file system, i file di log vengono archiviati in **greengrass-root/ggc/var/log** sul dispositivo core. Di seguito è riportata la struttura di directory generale:

`greengrass-root/ggc/var/log`

- `crash.log`
- `system`
 - log files for each Greengrass system component
- `user`
 - `region`
 - `account-id`
 - log files generated by each user-defined Lambda function
 - `aws`
 - log files generated by each connector

Note

Come impostazione predefinita, `greengrass-root` è la directory `/greengrass`. Se è configurata una [directory di scrittura](#), i log si trovano in tale directory.

Le seguenti considerazioni si applicano quando si utilizzano i log del file system:

- La lettura AWS IoT Greengrass dei log sul file system richiede i permessi di root.
- AWS IoT Greengrass supporta la rotazione basata sulle dimensioni e la pulizia automatica quando la quantità di dati di registro è prossima al limite configurato.
- Il file `crash.log` è disponibile solo nei log di file system. Questo registro non viene scritto in Logs. CloudWatch
- Si applicano limitazioni relative all'utilizzo del disco. Per ulteriori informazioni, consulta [the section called "Limitazioni di registrazione"](#).

Note

I log del software AWS IoT Greengrass Core v1.0 sono archiviati nella directory.
`greengrass-root/var/log`

Configurazione della registrazione predefinita

Se le impostazioni di registrazione non sono configurate in modo esplicito, AWS IoT Greengrass utilizza la seguente configurazione di registrazione predefinita dopo la prima distribuzione di gruppo.

AWS IoT Greengrass Componenti del sistema

- Tipo - FileSystem
- Componente - GreengrassSystem
- Livello - INFO
- Spazio - 128 KB

Funzioni Lambda definite dall'utente

- Tipo - FileSystem
- Componente - Lambda
- Livello - INFO
- Spazio - 128 KB

Note

Prima della prima distribuzione, solo i componenti del sistema scrivono i log nel file system perché non vengono distribuite funzioni Lambda definite dall'utente.

Configurare la registrazione per AWS IoT Greengrass

È possibile utilizzare la AWS IoT console o [AWS IoT Greengrass APIs](#) configurare la AWS IoT Greengrass registrazione.

Note

AWS IoT Greengrass Per consentire la scrittura di log su CloudWatch Logs, il ruolo del gruppo deve consentire le azioni Logs [richieste. CloudWatch](#)

Configurazione della registrazione (console)

Puoi configurare la registrazione nella pagina Impostazioni del gruppo.

1. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).

2. Scegli il gruppo in cui desideri configurare la registrazione.
3. Nella pagina di configurazione del gruppo, scegli la scheda Registri.
4. Scegli la posizione della registrazione, come segue:
 - Per configurare CloudWatch la registrazione, per la configurazione CloudWatch dei registri, scegli Modifica.
 - Per configurare la registrazione di file system, per Configurazione log locale, scegli Modifica.

Puoi configurare la registrazione per una posizione o entrambe le posizioni.

5. Nella modalità di configurazione dei registri, selezionare il livello di registro del sistema Greengrass o il livello di registro delle funzioni User Lambda. Puoi scegliere un componente o entrambi i componenti.
6. Scegli il livello minimo di eventi che desideri registrare. Gli eventi al di sotto di questa soglia vengono filtrati e non vengono archiviati.
7. Seleziona Salva. Le modifiche diventano effettive dopo la distribuzione del gruppo.

Configurazione della registrazione (API)

È possibile utilizzare AWS IoT Greengrass logger APIs per configurare la registrazione a livello di codice. Ad esempio, puoi utilizzare l'operazione [CreateLoggerDefinition](#) per creare una definizione di logger basata su un payload [LoggerDefinitionVersion](#), che utilizza la sintassi seguente:

```
{
  "Loggers": [
    {
      "Id": "string",
      "Type": "FileSystem|AWSCloudWatch",
      "Component": "GreengrassSystem|Lambda",
      "Level": "DEBUG|INFO|WARN|ERROR|FATAL",
      "Space": "integer"
    },
    {
      "Id": "string",
      ...
    }
  ]
}
```

```
}
```

`LoggerDefinitionVersion` è una serie di uno o più oggetti [Logger](#) che hanno le seguenti proprietà:

Id

Un identificatore per il logger.

Type

Il meccanismo di storage per gli eventi di log. Quando `AWSCloudWatch` viene utilizzato, gli eventi di registro vengono inviati ai registri. CloudWatch Quando viene utilizzato `FileSystem`, gli eventi di log vengono archiviati nel file system locale.

Valori validi: `AWSCloudWatch`, `FileSystem`

Component

L'origine dell'evento di log. Quando viene utilizzato `GreengrassSystem`, vengono registrati gli eventi provenienti dai componenti di sistema Greengrass. Quando viene utilizzato `Lambda`, vengono registrati gli eventi provenienti dalle funzioni Lambda definite dall'utente.

Valori validi: `GreengrassSystem`, `Lambda`

Level

La soglia del livello di log. Gli eventi di log al di sotto di questa soglia vengono filtrati e non vengono archiviati.

Valori validi: `DEBUG`, `INFO` (consigliato), `WARN`, `ERROR`, `FATAL`

Space

Il volume massimo di storage locale, in KB, da utilizzare per l'archiviazione dei log. Questo campo si applica solo se `Type` è impostato su `FileSystem`.

Esempio di configurazione

L'esempio `LoggerDefinitionVersion` seguente specifica una configurazione di registrazione che effettua le seguenti operazioni:

- Attiva la registrazione del file system `ERROR` e versioni successive per i componenti AWS IoT Greengrass del sistema.

- Attiva la registrazione del file system INFO (e versioni successive) per le funzioni Lambda definite dall'utente.
- Attiva CloudWatch INFO (e oltre) la registrazione per le funzioni Lambda definite dall'utente.

```
{
  "Name": "LoggingExample",
  "InitialVersion": {
    "Loggers": [
      {
        "Id": "1",
        "Component": "GreengrassSystem",
        "Level": "ERROR",
        "Space": 10240,
        "Type": "FileSystem"
      },
      {
        "Id": "2",
        "Component": "Lambda",
        "Level": "INFO",
        "Space": 10240,
        "Type": "FileSystem"
      },
      {
        "Id": "3",
        "Component": "Lambda",
        "Level": "INFO",
        "Type": "AWSCloudWatch"
      }
    ]
  }
}
```

Dopo aver creato una versione della definizione di logger, puoi utilizzare l'ARN della versione per creare una versione di gruppo prima di [distribuire il gruppo](#).

Limitazioni di registrazione

AWS IoT Greengrass presenta le seguenti limitazioni di registrazione.

Transazioni al secondo

Quando la registrazione a CloudWatch è abilitata, il componente di registrazione registra in batch gli eventi localmente prima di inviarli a CloudWatch, in modo da poter effettuare il log a una velocità superiore a cinque richieste al secondo per flusso di log.

Memoria

Se AWS IoT Greengrass è configurato per inviare log a CloudWatch e una funzione Lambda registra più di 5 MB/secondo per un periodo di tempo prolungato, la pipeline di elaborazione interna alla fine si riempie. Il caso peggiore teorico è di 6 MB per funzione Lambda.

Differenza dell'ora

Quando la registrazione a CloudWatch è abilitata, il componente di registrazione firma le richieste CloudWatch utilizzando il normale processo di firma Signature versione 4. Se l'ora di sistema sul dispositivo AWS IoT Greengrass principale non è sincronizzata per più di [15 minuti](#), le richieste vengono rifiutate.

Utilizzo del disco

Utilizza la seguente formula per calcolare il volume totale massimo di utilizzo del disco per la registrazione.

```
greengrass-system-component-space * 8      // 7 if automatic IP detection is disabled
+ 128KB                                       // the internal log for the local logging
component
+ lambda-space * lambda-count           // different versions of a Lambda function are
treated as one
```

Dove:

`greengrass-system-component-space`

La quantità massima di archiviazione locale per i registri dei componenti di AWS IoT Greengrass sistema.

`lambda-space`

La quantità massima di archiviazione locale per i log delle funzioni Lambda.

lambda-count

Il numero di funzioni Lambda distribuite.

Perdita di log

Se il dispositivo AWS IoT Greengrass principale è configurato per accedere solo a CloudWatch e non è disponibile alcuna connettività Internet, non è possibile recuperare i log attualmente presenti in memoria.

Quando le funzioni Lambda vengono terminate (ad esempio, durante la distribuzione), non vengono scritti registri di alcuni secondi. CloudWatch

CloudTrail registri

AWS IoT Greengrass viene eseguito con AWS CloudTrail, un servizio che fornisce un registro delle azioni intraprese da un utente, da un ruolo o da un AWS servizio in AWS IoT Greengrass. Per ulteriori informazioni, consulta [the section called “Registrazione delle chiamate AWS IoT Greengrass API con AWS CloudTrail”](#).

Registrazione delle chiamate AWS IoT Greengrass API con AWS CloudTrail

AWS IoT Greengrass è integrato con AWS CloudTrail, un servizio che fornisce una registrazione delle azioni intraprese da un utente, un ruolo o un AWS servizio in AWS IoT Greengrass. CloudTrail acquisisce tutte le chiamate API AWS IoT Greengrass come eventi. Le chiamate acquisite includono chiamate dalla AWS IoT Greengrass console e chiamate di codice alle operazioni AWS IoT Greengrass API. Se crei un trail, puoi abilitare la distribuzione continua di CloudTrail eventi a un bucket Amazon S3, inclusi gli eventi per. AWS IoT Greengrass Se non configuri un percorso, puoi comunque visualizzare gli eventi più recenti nella CloudTrail console nella cronologia degli eventi. Utilizzando le informazioni raccolte da CloudTrail, puoi determinare a quale richiesta è stata inviata AWS IoT Greengrass, l'indirizzo IP da cui è stata effettuata la richiesta, chi ha effettuato la richiesta, quando è stata effettuata e dettagli aggiuntivi.

Per ulteriori informazioni CloudTrail, consulta la [Guida AWS CloudTrail per l'utente](#).

AWS IoT Greengrass informazioni in CloudTrail

CloudTrail è abilitato sul tuo account al Account AWS momento della creazione dell'account. Quando si verifica un'attività in AWS IoT Greengrass, tale attività viene registrata in un CloudTrail evento insieme ad altri eventi AWS di servizio nella cronologia degli eventi. Puoi visualizzare, cercare e scaricare eventi recenti in Account AWS. Per ulteriori informazioni, consulta [Visualizzazione degli eventi con la cronologia degli CloudTrail eventi](#).

Per una registrazione continua degli eventi del tuo Account AWS, inclusi gli eventi di AWS IoT Greengrass, crea un percorso. Un trail consente di CloudTrail inviare file di log a un bucket Amazon S3. Per impostazione predefinita, quando crei un percorso nella console, il percorso si applica a tutti i Regione AWS file. Il trail registra gli eventi di tutte le regioni della AWS partizione e consegna i file di log al bucket Amazon S3 specificato. Inoltre, puoi configurare altri AWS servizi per analizzare ulteriormente e agire in base ai dati sugli eventi raccolti nei log. CloudTrail Per ulteriori informazioni, consulta gli argomenti seguenti:

- [Panoramica della creazione di un percorso](#)
- [CloudTrail servizi e integrazioni supportati](#)
- [Configurazione delle notifiche Amazon SNS per CloudTrail](#)
- [Ricezione di file di CloudTrail registro da più regioni](#) e [ricezione di file di CloudTrail registro da più account](#)

Tutte AWS IoT Greengrass le azioni vengono registrate CloudTrail e documentate nel riferimento [AWS IoT Greengrass API](#). Ad esempio, le chiamate alle `CreateFunctionDefinition` azioni `AssociateServiceRoleToAccount`, `GetGroupVersionGetConnectivityInfo`, e generano voci nei file di CloudTrail registro.

Ogni evento o voce di log contiene informazioni sull'utente che ha generato la richiesta. Le informazioni di identità consentono di determinare quanto segue:

- Se la richiesta è stata effettuata con credenziali utente root o AWS Identity and Access Management (IAM).
- Se la richiesta è stata effettuata con le credenziali di sicurezza temporanee per un ruolo o un utente federato.
- Se la richiesta è stata effettuata da un altro AWS servizio.

Per ulteriori informazioni, consulta [Elemento CloudTrail userIdentity](#).

Comprensione delle AWS IoT Greengrass voci dei file di registro

Un trail è una configurazione che consente la distribuzione di eventi come file di log in un bucket Amazon S3 specificato dall'utente. CloudTrail i file di registro contengono una o più voci di registro. Un evento rappresenta una singola richiesta proveniente da qualsiasi fonte e include informazioni sull'azione richiesta, la data e l'ora dell'azione, i parametri della richiesta e così via. CloudTrail i file di registro non sono una traccia ordinata dello stack delle chiamate API pubbliche, quindi non vengono visualizzati in un ordine specifico.

L'esempio seguente mostra una voce di CloudTrail registro che illustra l'AssociateServiceRoleToAccountazione.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::123456789012:user/Mary_Major",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "Mary_Major"
  },
  "eventTime": "2018-10-17T17:04:02Z",
  "eventSource": "greengrass.amazonaws.com",
  "eventName": "AssociateServiceRoleToAccount",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "203.0.113.12",
  "userAgent": "apimanager.amazonaws.com",
  "errorCode": "BadRequestException",
  "requestParameters": null,
  "responseElements": {
    "Message": "That role ARN is invalid."
  },
  "requestID": "a5990ec6-d22e-11e8-8ae5-c7d2eEXAMPLE",
  "eventID": "b9070ce2-0238-451a-a9db-2dbf1EXAMPLE",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

L'esempio seguente mostra una voce di CloudTrail registro che illustra l'GetGroupVersionazione.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::123456789012:user/Mary_Major",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "Mary_Major",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2018-10-17T18:14:57Z"
      }
    },
    "invokedBy": "apimanager.amazonaws.com"
  },
  "eventTime": "2018-10-17T18:15:11Z",
  "eventSource": "greengrass.amazonaws.com",
  "eventName": "GetGroupVersion",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "203.0.113.12",
  "userAgent": "apimanager.amazonaws.com",
  "requestParameters": {
    "GroupVersionId": "6c477753-dbf2-4cb8-acc3-5ba4eEXAMPLE",
    "GroupId": "90fcf6df-413c-4515-93a8-00056EXAMPLE"
  },
  "responseElements": null,
  "requestID": "95dcffce-d238-11e8-9240-a3993EXAMPLE",
  "eventID": "8a608034-82ed-431b-b5e0-87fbdEXAMPLE",
  "readOnly": true,
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

L'esempio seguente mostra una voce di CloudTrail registro che illustra l'GetConnectivityInfoazione.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
```

```

    "arn": "arn:aws:iam::123456789012:user/Mary_Major",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "Mary_Major"
  },
  "eventTime": "2018-10-17T17:02:12Z",
  "eventSource": "greengrass.amazonaws.com",
  "eventName": "GetConnectivityInfo",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "203.0.113.12",
  "userAgent": "apimanager.amazonaws.com",
  "requestParameters": {
    "ThingName": "us-east-1_CIS_1539795000000_"
  },
  "responseElements": null,
  "requestID": "63e3ebe3-d22e-11e8-9ddd-5baf3EXAMPLE",
  "eventID": "db2260d1-a8cc-4a65-b92a-13f65EXAMPLE",
  "readOnly": true,
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}

```

L'esempio seguente mostra una voce di CloudTrail registro che illustra l'CreateFunctionDefinitionazione.

```

{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::123456789012:user/Mary_Major",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "Mary_Major"
  },
  "eventTime": "2018-10-17T18:01:11Z",
  "eventSource": "greengrass.amazonaws.com",
  "eventName": "CreateFunctionDefinition",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "203.0.113.12",
  "userAgent": "apimanager.amazonaws.com",
  "requestParameters": {
    "InitialVersion": "****"
  }
}

```

```
    },
    "responseElements": {
      "CreationTimestamp": "2018-10-17T18:01:11.449Z",
      "LatestVersion": "dae06a61-c32c-41e9-b983-ee5cfEXAMPLE",
      "LatestVersionArn": "arn:aws:greengrass:us-east-1:123456789012:/greengrass/definition/functions/7a94847d-d4d2-406c-9796-a3529EXAMPLE/versions/dae06a61-c32c-41e9-b983-ee5cfEXAMPLE",
      "LastUpdatedTimestamp": "2018-10-17T18:01:11.449Z",
      "Id": "7a94847d-d4d2-406c-9796-a3529EXAMPLE",
      "Arn": "arn:aws:greengrass:us-east-1:123456789012:/greengrass/definition/functions/7a94847d-d4d2-406c-9796-a3529EXAMPLE"
    },
    "requestID": "a17d4b96-d236-11e8-a74e-3db27EXAMPLE",
    "eventID": "bdbf6677-a47a-4c78-b227-c5f64EXAMPLE",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
  }
}
```

Consulta anche

- [Che cos'è AWS CloudTrail?](#) nella Guida per l'AWS CloudTrail utente
- [Creazione di una EventBridge regola che si attiva su una chiamata AWS API utilizzando CloudTrail](#) la Amazon EventBridge User Guide
- [AWS IoT Greengrass Documentazione di riferimento dell'API](#)

Raccolta di dati di telemetria sanitaria del sistema dai dispositivi principali AWS IoT Greengrass

I dati di telemetria sullo stato del sistema sono dati diagnostici che possono aiutarti a monitorare le prestazioni delle operazioni critiche sui tuoi dispositivi principali Greengrass. L'agente di telemetria sul core di Greengrass raccoglie dati di telemetria locali e li pubblica su Amazon senza richiedere alcuna interazione con il cliente. EventBridge I dispositivi principali pubblicano i dati di telemetria con la massima diligenza possibile. EventBridge Ad esempio, i dispositivi principali potrebbero non riuscire a fornire dati di telemetria mentre sono offline.

 Note

Amazon EventBridge è un servizio di bus di eventi che puoi utilizzare per connettere le tue applicazioni con dati provenienti da una varietà di fonti, come i dispositivi principali Greengrass e le notifiche di [distribuzione](#). Per ulteriori informazioni, consulta [What is Amazon EventBridge?](#) nella Amazon EventBridge User Guide.

Puoi creare progetti e applicazioni per recuperare, analizzare, trasformare e generare report sui dati di telemetria dai tuoi dispositivi periferici. Gli esperti del settore, come gli ingegneri di processo, possono utilizzare queste applicazioni per ottenere informazioni sullo stato della flotta.

Per garantire il corretto funzionamento dei componenti Greengrass Edge, AWS IoT Greengrass utilizza i dati per scopi di sviluppo e miglioramento della qualità. Questa funzionalità aiuta anche a sviluppare nuove e migliorate funzionalità edge. AWS IoT Greengrass conserva i dati di telemetria solo per un massimo di sette giorni.

Questa funzionalità è disponibile nel software AWS IoT Greengrass Core v1.11.0 ed è abilitata per impostazione predefinita per tutti i core Greengrass, inclusi i core esistenti. L'utente inizia automaticamente a ricevere dati non appena si esegue l'aggiornamento al software Core v1.11.0 o versione successiva AWS IoT Greengrass .

Per informazioni su come accedere o gestire i dati di telemetria pubblicati, consulta. [the section called “Iscrizione per ricevere dati di telemetria”](#)

L'agente di telemetria raccoglie e pubblica le seguenti metriche di sistema.

Metriche di telemetria

Nome	Descrizione	Origine
SystemMemUsage	La quantità di memoria attualmente utilizzata da tutte le applicazioni sul dispositivo principale Greengrass, incluso il sistema operativo.	System (Sistema)
CpuUsage	La quantità di CPU attualmente utilizzata da tutte le applicazioni sul dispositivo	System (Sistema)

Nome	Descrizione	Origine
	principale Greengrass, incluso il sistema operativo.	
TotalNumberOfFDs	Il numero di descrittori di file memorizzati dal sistema operativo del dispositivo principale Greengrass. Un descrittore di file identifica in modo univoco un file aperto.	System (Sistema)
LambdaOutOfMemory	Il numero di esecuzioni che determinano l'esaurimento della memoria della funzione Lambda.	System (Sistema)
DroppedMessageCount	Il numero di messaggi eliminati a cui sono destinati. AWS IoT Core	GGCloudSpooler componente di sistema
LambdaTimeout	Il numero di timeout per l'esecuzione della funzione Lambda definita dall'utente.	Funzione Cloud AWS e sistema Lambda definiti dall'utente
LambdaUngracefully Killed	Il numero di esecuzioni che la funzione Lambda definita dall'utente non riesce a completare.	Funzione Cloud AWS e sistema Lambda definiti dall'utente
LambdaError	Il numero di esecuzioni che determinano la scrittura dei log degli errori da parte della funzione Lambda definita dall'utente.	Funzione Cloud AWS e sistema Lambda definiti dall'utente
BytesAppended	Il numero di byte di dati aggiunti allo stream manager.	GGStreamManager componente di sistema

Nome	Descrizione	Origine
BytesUploadedToIoTAnalytics	Il numero di byte di dati che Stream Manager esporta nei canali in AWS IoT Analytics cui vengono esportati.	GGStreamManager componente di sistema
BytesUploadedToKinesis	Il numero di byte di dati che Stream Manager esporta in flussi in Amazon Kinesis Data Streams.	GGStreamManager componente di sistema
BytesUploadedToIoTSiteWise	Il numero di byte di dati in cui lo stream manager esporta nelle AWS IoT SiteWise proprietà degli asset.	GGStreamManager componente di sistema
BytesUploadedToS3ExportTaskExecutor	Il numero di byte di dati che Stream Manager esporta in oggetti in Amazon S3.	GGStreamManager componente di sistema
BytesUploadedToHTTP	Il numero di byte di dati che lo stream manager esporta in HTTP.	GGStreamManager componente di sistema

Configurazione delle impostazioni di telemetria

La telemetria Greengrass utilizza le seguenti impostazioni:

- L'agente di telemetria aggrega i dati di telemetria ogni ora.
- L'agente di telemetria pubblica un messaggio di telemetria ogni 24 ore.

Note

Le impostazioni sono immutabili.

È possibile abilitare o disabilitare la funzione di telemetria per un dispositivo principale Greengrass. AWS IoT Greengrass utilizza le [ombre per gestire](#) la configurazione della telemetria. Le modifiche hanno effetto immediato quando il core è connesso a. AWS IoT Core

L'agente di telemetria pubblica i dati utilizzando il protocollo MQTT con un livello di qualità del servizio (QoS) pari a 0. Ciò significa che non conferma la consegna né riprova i tentativi di pubblicazione. I messaggi di telemetria condividono una connessione MQTT con altri messaggi per gli abbonamenti a cui sono destinati. AWS IoT Core

Oltre ai costi di collegamento dati, il trasferimento dei dati dal core all'altro è gratuito. AWS IoT Core Questo perché l'agente pubblica su un argomento AWS riservato. Tuttavia, a seconda del caso d'uso, potrebbero verificarsi dei costi quando si ricevono o si elaborano i dati.

Requisiti

Quando si configurano le impostazioni di telemetria, si applicano i seguenti requisiti:

- È necessario utilizzare il software AWS IoT Greengrass Core v1.11.0 o versione successiva.

Note

Se utilizzi una versione precedente e non desideri utilizzare la telemetria, non devi fare nulla.

- È necessario fornire le autorizzazioni IAM per aggiornare il core (thing) shadow e chiamare la configurazione APIs prima di aggiornare le impostazioni di telemetria.

Il seguente esempio di policy IAM consente di gestire la configurazione shadow e runtime di un core specifico:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowManageShadow",
      "Effect": "Allow",
      "Action": [
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:DeleteThingShadow",
        "iot:DescribeThing"
      ]
    }
  ]
}
```

```

    ],
    "Resource": [
        "arn:aws:iot:region:account-id:thing/core-name-"
    ]
},
{
    "Sid": "AllowManageRuntimeConfig",
    "Effect": "Allow",
    "Action": [
        "greengrass:GetCoreRuntimeConfiguration",
        "greengrass:UpdateCoreRuntimeConfiguration"
    ],
    "Resource": [
        "arn:aws:iot:region:account-id:thing/core-name"
    ]
}
]
}

```

È possibile concedere un accesso granulare o condizionale alle risorse, ad esempio utilizzando uno schema di denominazione con caratteri jolly*. Per ulteriori informazioni, consulta [Aggiungere e rimuovere le policy IAM nella IAM User Guide](#).

Configura le impostazioni di telemetria (console)

Di seguito viene illustrato come aggiornare le impostazioni di telemetria di un core Greengrass nella console. AWS IoT Greengrass

1. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).
2. In Gruppi Greengrass, scegli il tuo gruppo target.
3. Nella pagina di configurazione del gruppo, nella sezione Panoramica, scegli il tuo core Greengrass.
4. Nella pagina di configurazione del core, scegli la scheda Telemetria.
5. Nella sezione Telemetria dello stato del sistema, scegli Configura.
6. In Configura telemetria, seleziona Telemetria per abilitare o disabilitare lo stato della telemetria.

 Important

Per impostazione predefinita, la funzionalità di telemetria è abilitata per il software Core v1.11.0 o versione successiva. AWS IoT Greengrass

Le modifiche hanno effetto in fase di esecuzione. Non è necessario distribuire il gruppo.

Configurazione delle impostazioni di telemetria (CLI)

Nell' AWS IoT Greengrass API, l'`TelemetryConfiguration` oggetto rappresenta le impostazioni di telemetria di un core Greengrass. Questo oggetto fa parte dell'`RuntimeConfiguration` oggetto associato al core. Puoi utilizzare l' AWS IoT Greengrass API o l' AWS SDK per gestire la telemetria di Greengrass. AWS CLI Gli esempi in questa sezione utilizzano il. AWS CLI

Per controllare le impostazioni di telemetria

Il comando seguente ottiene le impostazioni di telemetria di un core Greengrass.

- Sostituisci *core-thing-name* con il nome del core di destinazione.

Per ottenere il nome della cosa, si usa il [get-core-definition-version](#) comando. Il comando restituisce l'ARN dell'oggetto che contiene il nome dell'oggetto.

```
aws greengrass get-thing-runtime-configuration --thing-name core-thing-name
```

Il comando restituisce un `GetCoreRuntimeConfigurationResponse` oggetto nella risposta JSON. Per esempio:

```
{
  "RuntimeConfiguration": {
    "TelemetryConfiguration": {
      "ConfigurationSyncStatus": "OutOfSync",
      "Telemetry": "On"
    }
  }
}
```

Per configurare le impostazioni di telemetria

Il comando seguente aggiorna le impostazioni di telemetria per un core Greengrass.

- Sostituisci *core-thing-name* con il nome del core di destinazione.

Per ottenere il nome della cosa, si usa il [get-core-definition-version](#) comando. Il comando restituisce l'ARN dell'oggetto che contiene il nome dell'oggetto.

JSON expanded

```
aws greengrass update-thing-runtime-configuration --thing-name core-thing-name --
telemetry-configuration '{
  "RuntimeConfiguration": {
    "TelemetryConfiguration": {
      "ConfigurationSyncStatus": "InSync",
      "Telemetry": "Off"
    }
  }
}
```

JSON single-line

```
aws greengrass update-thing-runtime-configuration --thing-name core-thing-name --
telemetry-configuration "{\"TelemetryConfiguration\":{\"ConfigurationSyncStatus
\":\"InSync\",\"Telemetry\":\"Off\"}}"
```

Le modifiche alle impostazioni di telemetria sono state applicate se lo è.

ConfigurationSyncStatus InSync Le modifiche hanno effetto in fase di esecuzione. Non è necessario distribuire il gruppo.

TelemetryConfiguration oggetto

L'TelemetryConfiguration oggetto ha le seguenti proprietà:

ConfigurationSyncStatus

Verifica se le impostazioni di telemetria sono sincronizzate. È possibile che non sia possibile apportare modifiche a questa proprietà.

Tipo: stringa

Valori validi: InSync o OutOfSync

Telemetry

Attiva o disattiva la telemetria. Il valore predefinito è On.

Tipo: stringa

Valori validi: On o Off

Iscrizione per ricevere dati di telemetria

Puoi creare regole in Amazon EventBridge che definiscono come elaborare i dati di telemetria pubblicati dal dispositivo principale Greengrass. Quando EventBridge riceve i dati, richiama le azioni mirate definite nelle regole. Ad esempio, è possibile creare regole relative agli eventi che inviano notifiche, archiviano informazioni sugli eventi, intraprendono azioni correttive o richiamano altri eventi.

Evento di telemetria

L'evento per una modifica dello stato di distribuzione, inclusi i dati di telemetria, utilizza il formato seguente:

```
{
  "version": "0",
  "id": "f70f943b-9ae2-e7a5-fec4-4c22178a3e6a",
  "detail-type": "Greengrass Telemetry Data",
  "source": "aws.greengrass",
  "account": "123456789012",
  "time": "2020-07-28T20:45:53Z",
  "region": "us-west-1",
  "resources": [],
  "detail": {
    "ThingName": "CoolThing",
    "Schema": "2020-06-30",
    "ADP": [
      {
        "TS": 123231546,
        "NS": "StreamManager",
        "M": [
          {
            "N": "BytesAppended|BytesUploadedToKinesis",
            "Sum": 11,
            "U": "Bytes"
          }
        ]
      }
    ]
  }
}
```

```
    ]
  },
  {
    "TS": 123231546,
    "NS": "StreamManager",
    "M": [
      {
        "N": "BytesAppended|BytesUploadedToS3ExportTaskExecutor",
        "Sum": 11,
        "U": "Bytes"
      }
    ]
  },
  {
    "TS": 123231546,
    "NS": "StreamManager",
    "M": [
      {
        "N": "BytesAppended|BytesUploadedToHTTP",
        "Sum": 11,
        "U": "Bytes"
      }
    ]
  },
  {
    "TS": 123231546,
    "NS": "StreamManager",
    "M": [
      {
        "N": "BytesAppended|BytesUploadedToIoTAnalytics",
        "Sum": 11,
        "U": "Bytes"
      }
    ]
  },
  {
    "TS": 123231546,
    "NS": "StreamManager",
    "M": [
      {
        "N": "BytesAppended|BytesUploadedToIoTSiteWise",
        "Sum": 11,
        "U": "Bytes"
      }
    ]
  }
}
```



```
    ]
  },
  {
    "TS": 123231546,
    "NS": "arn:aws:lambda:us-west-1:123456789012:function:my-function",
    "M": [
      {
        "N": "LambdaTimeout",
        "Sum": 15,
        "U": "Count"
      }
    ]
  },
  {
    "TS": 123231546,
    "NS": "CloudSpooler",
    "M": [
      {
        "N": "DroppedMessageCount",
        "Sum": 15,
        "U": "Count"
      }
    ]
  },
  {
    "TS": 1593727692,
    "NS": "SystemMetrics",
    "M": [
      {
        "N": "SystemMemUsage",
        "Sum": 11.23,
        "U": "Megabytes"
      },
      {
        "N": "CpuUsage",
        "Sum": 35.63,
        "U": "Percent"
      },
      {
        "N": "TotalNumberOfFDs",
        "Sum": 416,
        "U": "Count"
      }
    ]
  }
]
```

```
    },
    {
      "TS": 1593727692,
      "NS": "arn:aws:lambda:us-west-1:123456789012:function:my-function",
      "M": [
        {
          "N": "LambdaOutOfMemory",
          "Sum": 12,
          "U": "Count"
        },
        {
          "N": "LambdaUngracefullyKilled",
          "Sum": 100,
          "U": "Count"
        },
        {
          "N": "LambdaError",
          "Sum": 7,
          "U": "Count"
        }
      ]
    }
  ]
}
```

L'ADParray contiene un elenco di punti dati aggregati con le seguenti proprietà:

TS

Obbligatorio. Il timestamp di quando i dati sono stati aggregati.

NS

Obbligatorio. Lo spazio dei nomi del sistema.

M

Obbligatorio. L'elenco delle metriche. Una metrica contiene le seguenti proprietà:

N

Il nome della [metrica](#).

Sum

Il valore metrico aggregato. L'agente di telemetria aggiunge nuovi valori al totale precedente, quindi la somma è un valore sempre crescente. È possibile utilizzare il timestamp per trovare il valore di un'aggregazione specifica. Ad esempio, per trovare l'ultimo valore aggregato, sottrai il valore con data e ora precedente dall'ultimo valore con data e ora.

U

L'unità del valore metrico.

ThingName

Obbligatorio. Il nome del dispositivo oggetto scelto come target.

Prerequisiti per la creazione di regole EventBridge

Prima di creare una EventBridge regola per AWS IoT Greengrass, è necessario effettuare le seguenti operazioni:

- Acquisisci familiarità con eventi, regole e obiettivi in EventBridge
- Crea e configura gli [obiettivi](#) richiamati dalle tue regole. EventBridge Le regole possono richiamare molti tipi di destinazioni, ad esempio stream Amazon Kinesis, funzioni AWS Lambda , argomenti Amazon SNS e code Amazon SQS.

La tua EventBridge regola e gli obiettivi associati devono trovarsi nello stesso luogo in Regione AWS cui hai creato le tue risorse Greengrass. Per ulteriori informazioni, consulta [Service endpoints and quotas](#) in Riferimenti generali di AWS

Per ulteriori informazioni, consulta [What is Amazon EventBridge?](#) e [Guida introduttiva ad Amazon EventBridge](#) nella Amazon EventBridge User Guide.

Crea una regola di evento per ottenere dati di telemetria (console)

Utilizza i seguenti passaggi per AWS Management Console creare una EventBridge regola che riceva i dati di telemetria pubblicati dal core Greengrass. Ciò consente a server Web, indirizzi e-mail e altri sottoscrittori di argomenti di rispondere all'evento. Per ulteriori informazioni, consulta [Creazione di una EventBridge regola che si attiva su un evento da una AWS risorsa](#) nella Amazon EventBridge User Guide.

1. Apri la [EventBridgeconsole Amazon](#) e scegli Crea regola.
2. In Nome e descrizione, immettere un nome e una descrizione per la regola.
3. Scegli Event bus e abilita la regola sul bus di eventi selezionato.
4. Seleziona il tipo di regola e scegli Regola con uno schema di eventi.
5. Scegli Next (Successivo).
6. Per Origine dell'evento, scegli AWS eventi o eventi EventBridge partner.
7. Per Evento di esempio, scegli AWS eventi e seleziona Greengrass Telemetry Data.
8. In Event pattern, effettuate le seguenti selezioni:
 - a. Per Origine evento, scegli Servizi AWS .
 - b. Per l'AWS assistenza, scegli Greengrass.
 - c. Per Tipo di evento, scegli Greengrass Telemetry Data.
9. Scegli Next (Successivo).
10. Per Target 1, scegli il servizio.AWS
11. Per Seleziona una destinazione, scegli la coda SQS.
12. Per Queue, scegli la tua funzione.

Crea una regola di evento per ottenere dati di telemetria (CLI)

Utilizza i seguenti passaggi per AWS CLI creare una EventBridge regola che riceva i dati di telemetria pubblicati dal core Greengrass. Ciò consente a server Web, indirizzi e-mail e altri sottoscrittori di argomenti di rispondere all'evento.

1. Crea la regola.
 - *thing-name* Sostituiscilo con il nome dell'oggetto del core.

Per ottenere il nome della cosa, si usa il [get-core-definition-version](#) comando. Il comando restituisce l'ARN dell'oggetto che contiene il nome dell'oggetto.

```
aws events put-rule \  
  --name TestRule \  
  --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"ThingName\":  
    [\"thing-name\"]}}\"
```

Le proprietà omesse dal modello vengono ignorate.

2. Aggiungi l'argomento come destinazione della regola. L'esempio seguente utilizza Amazon SQS ma puoi configurare altri tipi di destinazione.
 - Sostituisci *queue-arn* con l'ARN della tua coda Amazon SQS.

```
aws events put-targets \  
  --rule TestRule \  
  --targets "Id"="1", "Arn"="queue-arn"
```

Note

Per consentire EventBridge ad Amazon di richiamare la tua coda di destinazione, devi aggiungere una politica basata sulle risorse all'argomento. Per ulteriori informazioni, consulta le [autorizzazioni di Amazon SQS](#) nella Amazon EventBridge User Guide.

Per ulteriori informazioni, consulta la sezione [Eventi e modelli di eventi EventBridge nella](#) Amazon EventBridge User Guide.

Risoluzione dei problemi di AWS IoT Greengrass telemetria

Utilizza le seguenti informazioni per risolvere i problemi relativi alla configurazione della telemetria. AWS IoT Greengrass

Errore: la risposta contiene "ConfigurationStatus«:" "dopo l'esecuzione del comando OutOfSync get-thing-runtime-configuration

Soluzioni:

- Il servizio AWS IoT Device Shadow richiede tempo per elaborare gli aggiornamenti della configurazione di runtime e per fornire gli aggiornamenti al dispositivo principale Greengrass. Potresti attendere e verificare se le impostazioni di telemetria sono sincronizzate in un secondo momento.
- Assicurati che il tuo dispositivo principale sia online.
- Abilita [Amazon CloudWatch Logs in AWS IoT Core](#) per monitorare lo shadow.

- Usa le [AWS IoT metriche](#) per monitorare le tue attività.

Chiamata dell'API di controllo sanitario locale

AWS IoT Greengrass contiene un'API HTTP locale che fornisce un'istantanea dello stato corrente dei processi di lavoro locali avviati da AWS IoT Greengrass. Questa istantanea include funzioni Lambda definite dall'utente e funzioni Lambda di sistema. Le funzioni System Lambda fanno parte del software AWS IoT Greengrass Core. Vengono eseguite come processi di lavoro locali sul dispositivo principale e gestiscono operazioni come il routing dei messaggi, la sincronizzazione degli shadow locali e il rilevamento automatico degli indirizzi IP.

L'API per il controllo dello stato di salute supporta le seguenti richieste:

- Invia una GET richiesta per [ottenere informazioni sanitarie per tutti i lavoratori](#).
- Invia una POST richiesta per [ottenere informazioni sanitarie per determinati lavoratori](#).

Le richieste vengono inviate localmente sul dispositivo e non richiedono una connessione Internet.

Ottieni informazioni sanitarie per tutti i lavoratori

Invia una GET richiesta per ottenere informazioni sanitarie su tutti i lavoratori che corrono.

- Sostituisci *port* con il numero di porta dell'IPC.

```
GET http://localhost:port/2016-11-01/health/workers
```

port

Il numero di porta dell'IPC.

Il valore può variare tra 1024 e 65535. Il valore predefinito è 8000.

Per modificare questo numero di porta, è possibile aggiornare la `ggDaemonPort` proprietà nel `config.json` file. Per ulteriori informazioni, consulta [AWS IoT Greengrass file di configurazione principale](#).

Richiesta di esempio

La seguente `curl` richiesta di esempio ottiene informazioni sanitarie per tutti i lavoratori.

```
curl http://localhost:8000/2016-11-01/health/workers
```

Risposta JSON

Questa richiesta restituisce una serie di oggetti [informativi sulla salute dei lavoratori](#).

Example response

La seguente risposta di esempio elenca gli oggetti informativi sullo stato di salute per tutti i processi di lavoro avviati da AWS IoT Greengrass.

```
[
  {
    "FuncArn": "arn:aws:lambda::function:GGShadowService:1",
    "WorkerId" : "65515053-2f70-43dc-7cc0-1712bEXAMPLE",
    "ProcessId": "1234",
    "WorkerState": "Waiting"
  },
  {
    "FuncArn": "arn:aws:lambda::function:GGSecretManager:1",
    "WorkerId": "a9916cc2-1b4d-4f0e-4b12-b1872EXAMPLE",
    "ProcessId": "9798",
    "WorkerState": "Waiting"
  },
  {
    "FuncArn": "arn:aws:lambda:us-west-2:123456789012:function:my-lambda-function:3",
    "WorkerId": "2e6f785e-66a5-42c9-67df-42073EXAMPLE",
    "ProcessId": "11837",
    "WorkerState": "Waiting"
  },
  ...
]
```

Ottieni informazioni sanitarie su determinati lavoratori

Invia una POST richiesta per ottenere informazioni sanitarie su determinati lavoratori. Sostituisci *port* con il numero di porta dell'IPC. L'impostazione predefinita è 8000.

```
POST http://localhost:port/2016-11-01/health/workers
```

Richiesta di esempio

La seguente `curl` richiesta di esempio ottiene informazioni sanitarie per determinati lavoratori.

```
curl --data "@body.json" http://localhost:8000/2016-11-01/health/workers
```

Ecco un esempio di corpo della `body.json` richiesta:

```
{
  "FuncArns": [
    "arn:aws:lambda::function:GGShadowService:1",
    "arn:aws:lambda:us-west-2:123456789012:function:my-lambda-function:3"
  ]
}
```

Il corpo della richiesta contiene un `FuncArns` array.

FuncArns

Un elenco di Amazon Resource Names (ARNs) per le funzioni Lambda che rappresentano i lavoratori target.

- Per le funzioni Lambda definite dall'utente, specifica l'ARN della versione attualmente distribuita. Se hai aggiunto funzioni Lambda al gruppo utilizzando un alias ARN, puoi utilizzare la richiesta GET per ottenere tutti i lavoratori e quindi scegliere quello per cui desideri eseguire ARNs la query.
- Per le funzioni Lambda di sistema, specificare l'ARN della funzione Lambda corrispondente. Per ulteriori informazioni, consulta [the section called “Funzioni Lambda del sistema”](#).

Tipo: Array of Strings

Lunghezza minima: 1

Lunghezza massima: il numero totale di lavoratori avviati dal dispositivo AWS IoT Greengrass principale.

Risposta JSON

Questa richiesta restituisce un `Workers` array e un `InvalidArns` array.

Workers

Un elenco di oggetti informativi sanitari per i lavoratori specificati.

Tipo: serie di [oggetti informativi sanitari](#)

InvalidArns

Un elenco di ARNs funzioni non valide, incluse le funzioni a ARNs cui non sono associati lavoratori.

Tipo: Array of Strings

Example response

La seguente risposta di esempio elenca [gli oggetti informativi sanitari](#) per i lavoratori specificati.

```
{
  "Workers": [
    {
      "FuncArn": "arn:aws:lambda::function:GGShadowService:1",
      "WorkerId" : "65515053-2f70-43dc-7cc0-1712bEXAMPLE",
      "ProcessId": "1234",
      "WorkerState": "Waiting"
    },
    {
      "FuncArn": "arn:aws:lambda:us-west-2:123456789012:function:my-lambda-
function:3",
      "WorkerId": "2e6f785e-66a5-42c9-67df-42073ESAMPLE",
      "ProcessId": "11837",
      "WorkerState": "Waiting"
    }
  ],
  "InvalidArns" : [
    "some-malformed-arn",
    "arn:aws:lambda:us-west-2:123456789012:function:some-unknown-function:1"
  ]
}
```

Questa richiesta restituisce i seguenti errori:

400 - Richiesta non valida

Il corpo della richiesta non è valido. Per risolvere il problema, utilizzate il seguente formato e inviate nuovamente la richiesta:

```
{"FuncArns":["function-1-arn","function-2-arn"]}
```

400 La richiesta supera il numero massimo di lavoratori

Il numero ARNs specificato nell'FuncArnsarray supera il numero di lavoratori.

Informazioni sulla salute dei lavoratori

Un oggetto informativo sanitario contiene le seguenti proprietà:

FuncArn

L'ARN della funzione Lambda del sistema che rappresenta il lavoratore.

Tipo: `string`

WorkerId

L'ID del lavoratore. Questa proprietà può essere utile per il debug. I `runtime.log` file e i registri delle funzioni Lambda contengono l'ID del lavoratore, quindi questa proprietà può essere particolarmente utile per eseguire il debug di una funzione Lambda su richiesta che genera più istanze.

Tipo: `string`

ProcessId

L'ID di processo (PID) del processo di lavoro.

Tipo: `int`

WorkerState

Lo stato del lavoratore.

Tipo: `string`

I seguenti sono i possibili stati del lavoratore:

Working

Elaborazione di un messaggio.

Waiting

In attesa di un messaggio. Si applica alle funzioni Lambda di lunga durata eseguite come demone o processo autonomo.

Starting

Spun up, inizio.

FailedInitialization

Inizializzazione non riuscita.

Terminated

Fermato dal demone Greengrass

NotStarted

Avvio non riuscito. Sto effettuando un altro tentativo di partenza.

Initialized

Inizializzato con successo.

Funzioni Lambda del sistema

È possibile richiedere informazioni sullo stato di salute per le seguenti funzioni Lambda del sistema:

GGCloudSpooler

Gestisce la coda per i messaggi MQTT che hanno AWS IoT Core come origine o destinazione.

ARN: `arn:aws:lambda:::function:GGCloudSpooler:1`

GGConnManager

Instrada i messaggi MQTT tra i dispositivi core e client di Greengrass.

ARN: `arn:aws:lambda:::function:GGConnManager`

GGDeviceCertificateManager

Ascolta l' AWS IoT ombra per rilevare le modifiche agli endpoint IP del core e genera il certificato lato server utilizzato da Manager per l'autenticazione reciproca. GGConn

ARN: `arn:aws:lambda:::function:GGDeviceCertificateManager`

GGIPDetector

Gestisce il rilevamento automatico degli indirizzi IP che consente ai dispositivi del gruppo Greengrass di scoprire il dispositivo principale Greengrass. Questo servizio non è applicabile quando gli indirizzi IP vengono forniti manualmente.

ARN: `arn:aws:lambda:::function:GGIPDetector:1`

GGSecretManager

Gestisce l'archiviazione sicura dei segreti locali e l'accesso tramite Lambda e connettori definiti dall'utente.

ARN: `arn:aws:lambda:::function:GGSecretManager:1`

GGShadowService

Gestisce le ombre locali per i dispositivi client.

ARN: `arn:aws:lambda:::function:GGShadowService`

GGShadowSyncManager

Sincronizza le ombre locali con quelle Cloud AWS per il dispositivo principale e i dispositivi client, se la `syncShadow` proprietà del dispositivo è impostata su `true`

ARN: `arn:aws:lambda:::function:GGShadowSyncManager`

GGStreamManager

Elabora i flussi di dati localmente ed esegue esportazioni automatiche verso Cloud AWS

ARN: `arn:aws:lambda:::function:GGStreamManager:1`

GGTES

Il servizio di scambio di token locale che recupera le credenziali IAM definite nel ruolo del gruppo Greengrass utilizzato dal codice locale per accedere ai servizi AWS

ARN: `arn:aws:lambda:::function:GGTES`

Taggare le tue risorse AWS IoT Greengrass

I tag possono aiutarti a organizzare e gestire i tuoi AWS IoT Greengrass gruppi. Puoi usare i tag per assegnare metadati ai gruppi, le distribuzioni di massa, i core, i dispositivi e altre risorse che vengono aggiunti ai gruppi. I tag possono essere utilizzati anche nelle policy IAM per definire l'accesso condizionale alle risorse Greengrass.

Note

Attualmente, i tag di risorsa Greengrass non sono supportati per i gruppi di AWS IoT fatturazione o i report sull'allocazione dei costi.

Nozioni di base sui tag

I tag consentono di classificare le AWS IoT Greengrass risorse, ad esempio per scopo, proprietario e ambiente. Se disponi di tante risorse dello stesso tipo, puoi rapidamente individuare una risorsa specifica in base ai tag assegnati a essa. Un tag è formato da una chiave e da un valore opzionale, entrambi definiti da te. Ti consigliamo di creare un set di chiavi di tag per ciascun tipo di risorsa. Con un set di chiavi di tag coerente, la gestione delle risorse risulta semplificata. Ad esempio, è possibile definire un set di tag per i tuoi gruppi che consente di monitorare la sede di produzione dei tuoi dispositivi core. Per ulteriori informazioni, consulta [Strategie di tagging di AWS](#).

Supporto per l'etichettatura nella console AWS IoT

Puoi creare, visualizzare e gestire i tag per le tue Group risorse Greengrass nella AWS IoT console. Prima di creare i tag, tenere presente le limitazioni relative al tagging. Per ulteriori informazioni, consulta la sezione relativa alle [convenzioni di denominazione e utilizzo dei tag](#) nella Riferimenti generali di Amazon Web Services.

Per assegnare tag al momento della creazione di un gruppo

Puoi assegnare tag a un gruppo durante la creazione del gruppo. Scegli Aggiungi nuovo tag nella sezione Tag per mostrare i campi di immissione dei tag.

Per visualizzare e gestire i tag dalla pagina di configurazione del gruppo

Puoi visualizzare e gestire i tag dalla pagina di configurazione del gruppo scegliendo Visualizza impostazioni. Nella sezione Tag del gruppo, scegli Gestisci tag per aggiungere, modificare o rimuovere i tag di gruppo.

Supporto per l'etichettatura nell'API AWS IoT Greengrass

Puoi utilizzare l' AWS IoT Greengrass API per creare, elencare e gestire i tag per le AWS IoT Greengrass risorse che supportano il tagging. Prima di creare i tag, tenere presente le limitazioni relative al tagging. Per ulteriori informazioni, consulta la sezione relativa alle [convenzioni di denominazione e utilizzo dei tag](#) nella Riferimenti generali di Amazon Web Services.

- Per aggiungere i tag durante la creazione delle risorse, definirli nella proprietà tags della risorsa.
- Per aggiungere i tag dopo aver creato una risorsa o per aggiornare i valori dei tag, utilizzare l'operazione TagResource.
- Per rimuovere i tag da una risorsa, utilizzare l'operazione UntagResource.
- Per recuperare i tag associati a una risorsa, utilizzare l'operazione ListTagsForResource o ottenere la risorsa e ispezionare la proprietà tags.

La tabella seguente elenca le risorse a cui è possibile aggiungere tag nell' AWS IoT Greengrass API e le relative Create Get azioni.

Risorsa	Crea	Get
Group	CreateGroup	GetGroup
ConnectorDefinition	CreateConnectorDefinition	GetConnectorDefinition
CoreDefinition	CreateCoreDefinition	GetCoreDefinition
DeviceDefinition	CreateDeviceDefinition	GetDeviceDefinition
FunctionDefinition	CreateFunctionDefinition	GetFunctionDefinition

Risorsa	Crea	Get
LoggerDefinition	CreateLoggerDefinition	GetLoggerDefinition
ResourceDefinition	CreateResourceDefinition	GetResourceDefinition
SubscriptionDefinition	CreateSubscriptionDefinition	GetSubscriptionDefinition
BulkDeployment	StartBulkDeployment	GetBulkDeploymentsStatus

Utilizza le operazioni seguenti per elencare e gestire i tag per le risorse che supportano il tagging:

- [TagResource](#). Aggiunge tag a una risorsa. Utilizzato anche per modificare il valore della coppia chiave-valore del tag.
- [ListTagsForResource](#). Elenca i tag di una risorsa.
- [UntagResource](#). Rimuove i tag da una risorsa.

Puoi aggiungere o modificare i tag da una risorsa in qualsiasi momento. Per modificare il valore di una chiave del tag, aggiungere un tag alla risorsa che definisca la stessa chiave e il nuovo valore. Il nuovo valore sovrascrive il valore precedente. Puoi impostare il valore su una stringa vuota, ma non su null.

Se elimini una risorsa, verranno eliminati anche tutti i tag associati alla risorsa.

Note

Non confondete i tag delle risorse con gli attributi che potete assegnare agli AWS IoT oggetti. Sebbene i core di Greengrass siano AWS IoT cose, i tag di risorsa descritti in questo argomento sono allegati a un `elementoCoreDefinition`, non a un elemento fondamentale.

Utilizzo dei tag con policy IAM

Nelle policy IAM, puoi utilizzare i tag delle risorse per controllare l'accesso e le autorizzazioni degli utenti. Ad esempio, le policy possono consentire agli utenti di creare solo le risorse con un determinato tag. Le policy possono anche limitare gli utenti nella creazione o nella modifica di risorse con determinati tag. Puoi applicare tag alle risorse durante la creazione (operazione detta tag alla creazione), in modo da non dover eseguire script di tagging personalizzati in un secondo momento. Quando vengono lanciati nuovi ambienti con tag, le autorizzazioni IAM corrispondenti vengono applicate automaticamente.

I seguenti valori e chiavi di contesto della condizione possono essere utilizzati nell'elemento Condition (detto anche blocco Condition) della policy.

`greengrass:ResourceTag/tag-key: tag-value`

Concedi o nega agli utenti operazioni su risorse con specifici tag.

`aws:RequestTag/tag-key: tag-value`

Richiedi che un tag specifico venga utilizzato (o non utilizzato) durante la creazione di richieste API per creare o modificare tag su una risorsa compatibile con l'assegnazione di tag.

`aws:TagKeys: [tag-key, ...]`

Richiedi che un set di chiavi di tag specifico venga utilizzato (o non utilizzato) durante la creazione di una richiesta API per creare o modificare una risorsa compatibile con l'assegnazione di tag.

Le chiavi e i valori del contesto delle condizioni possono essere utilizzati solo su AWS IoT Greengrass azioni che agiscono su una risorsa taggabile. Queste operazioni considerano la risorsa un parametro obbligatorio. Ad esempio, puoi impostare l'accesso condizionale su `GetGroupVersion`. Non è possibile impostare l'accesso condizionale su `AssociateServiceRoleToAccount` senza che nella richiesta si faccia riferimento a una risorsa compatibile con l'assegnazione di tag (ad esempio un gruppo, una definizione core o una definizione dispositivo).

Per ulteriori informazioni, consulta [Controlling access using tags](#) e [riferimento alla policy IAM JSON](#) nella IAM User Guide. Il riferimento alle policy JSON include sintassi dettagliate, descrizioni ed esempi degli elementi, delle variabili e della logica di valutazione delle policy JSON in IAM.

Policy IAM di esempio

La seguente policy di esempio applica le autorizzazioni basate su tag che vincolano un utente beta alle sole operazioni sulle risorse beta.

- La prima istruzione consente a un utente IAM di agire solo su risorse che hanno il tag env=beta.
- La seconda istruzione impedisce a un utente IAM di rimuovere il tag env=beta dalle risorse. Ciò offre all'utente la garanzia di non rimuoverne l'accesso.

Note

Se si utilizzano i tag per controllare l'accesso alle risorse, è necessario anche gestire le autorizzazioni che consentono agli utenti di aggiungere o rimuovere i tag dalle stesse risorse. In caso contrario, in alcuni casi, un utente può eludere le restrizioni e ottenere l'accesso a una risorsa modificandone i tag.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "greengrass:*",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "greengrass:ResourceTag/env": "beta"
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": "greengrass:UntagResource",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/env": "beta"
        }
      }
    }
  ]
}
```

```
}
```

Per consentire agli utenti di applicare tag al momento della creazione, è necessario fornire loro le autorizzazioni appropriate. La policy di esempio seguente include la condizione `"aws:RequestTag/env": "beta"` sulle operazioni `greengrass:TagResource` e `greengrass:CreateGroup`, che consentono agli utenti di creare un gruppo solo aggiungono a questo il tag `env=beta`. In questo modo, gli utenti sono costretti ad applicare tag ai nuovi gruppi.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "greengrass:TagResource",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/env": "beta"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": "greengrass:CreateGroup",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/env": "beta"
        }
      }
    }
  ]
}
```

Lo snippet seguente mostra come è possibile specificare più valori di tag per una chiave di tag racchiudendoli in un elenco:

```
"StringEquals" : {
  "greengrass:ResourceTag/env" : ["dev", "test"]
}
```

Consulta anche

- [Etichettare AWS](#) le risorse in Riferimenti generali di Amazon Web Services

AWS CloudFormation supporto per AWS IoT Greengrass

AWS CloudFormation è un servizio che può aiutarti a creare, gestire e replicare AWS le tue risorse. È possibile utilizzare AWS CloudFormation i modelli per definire AWS IoT Greengrass i gruppi e i dispositivi client, gli abbonamenti e gli altri componenti che si desidera distribuire. Per vedere un esempio, consulta [the section called “Modello di esempio”](#).

Le risorse e le infrastrutture che è possibile generare da un modello si definiscono stack. Puoi definire tutte le risorse in un modello o consultarle da altri stack. Per ulteriori informazioni su AWS CloudFormation modelli e funzionalità, consulta [Cos'è? AWS CloudFormation](#) nella Guida AWS CloudFormation per l'utente.

Creazione di risorse

AWS CloudFormation i modelli sono documenti JSON o YAML che descrivono le proprietà e le relazioni delle risorse. AWS Sono supportate le seguenti AWS IoT Greengrass risorse:

- Gruppi
- Core
- Dispositivi client (dispositivi)
- Funzioni Lambda
- Connectors (Connettori)
- Risorse (locale, machine learning e segreto)
- Sottoscrizioni
- Logger (configurazioni di registrazione)

Nei AWS CloudFormation modelli, la struttura e la sintassi delle risorse Greengrass si basano sull' AWS IoT Greengrass API. Ad esempio, il [modello di esempio](#) associa un livello superiore DeviceDefinition a un DeviceDefinitionVersion che contiene un singolo dispositivo client. Per ulteriori informazioni, consulta [the section called “Panoramica del modello di oggetti del gruppo”](#).

Il [riferimento ai tipi di AWS IoT Greengrass risorse](#) nella Guida per AWS CloudFormation l'utente descrive le risorse Greengrass con cui è possibile gestire. AWS CloudFormation Quando utilizzi AWS CloudFormation modelli per creare risorse Greengrass, ti consigliamo di gestirle solo da. AWS CloudFormation Ad esempio, dovresti aggiornare il modello se desideri aggiungere, modificare o rimuovere un dispositivo (anziché utilizzare l' AWS IoT Greengrass API o la AWS IoT console).

In questo modo potrai utilizzare il rollback e altre caratteristiche di gestione delle modifiche AWS CloudFormation . Per ulteriori informazioni sull'utilizzo per AWS CloudFormation creare e gestire risorse e stack, consulta [Working with stacks nella Guida](#) per l'AWS CloudFormation utente.

Per una procedura dettagliata che mostra come creare e distribuire AWS IoT Greengrass risorse in un AWS CloudFormation modello, consulta [Automatizzare la AWS IoT Greengrass configurazione con AWS CloudFormation](#) su The Internet of Things sul blog ufficiale. AWS

Distribuzione delle risorse

Dopo aver creato uno AWS CloudFormation stack contenente la versione di gruppo, puoi utilizzare la console AWS CLI o AWS IoT per distribuirlo.

Note

Per distribuire un gruppo, devi avere un ruolo di servizio Greengrass associato al tuo Account AWS. Il ruolo di servizio consente di accedere AWS IoT Greengrass alle risorse dell'utente AWS Lambda e ad altri AWS servizi. Questo ruolo dovrebbe esistere se hai già schierato un gruppo Greengrass nella versione attuale. Regione AWS Per ulteriori informazioni, consulta [the section called “Ruolo del servizio Greengrass”](#).

Per distribuire il gruppo (AWS CLI)

- Esegui il comando [create-deployment](#).

```
aws greengrass create-deployment --group-id GroupId --group-version-id GroupVersionId --deployment-type NewDeployment
```

Note

L'CommandToDeployGroupistruzione nel [modello di esempio](#) mostra come generare il comando con il gruppo e la versione del gruppo IDs quando si crea uno stack.

Per distribuire il gruppo (console)

1. Nel riquadro di navigazione della AWS IoT console, in Gestione, espandi i dispositivi Greengrass, quindi scegli Gruppi (V1).

2. Scegliere il gruppo.
3. Nella pagina di configurazione del gruppo, scegli Distribuisci.

Modello di esempio

Il seguente modello di esempio crea un gruppo Greengrass che contiene un core, un dispositivo client, una funzione, un logger, un abbonamento e due risorse. A tale scopo, il modello segue il modello a oggetti dell' AWS IoT Greengrass API. Ad esempio, i dispositivi client che si desidera aggiungere al gruppo sono contenuti in una `DeviceDefinitionVersion` risorsa associata a una `DeviceDefinition` risorsa. Per aggiungere i dispositivi al gruppo, la versione del gruppo fa riferimento all'ARN di `DeviceDefinitionVersion`.

Il modello include parametri che consentono di specificare il certificato ARNs per il core e il dispositivo e la versione ARN della funzione Lambda di origine (che è una risorsa). AWS Lambda Utilizza le Ref funzioni GetAtt intrinseche di riferimento IDs e altri attributi necessari per creare le risorse Greengrass. ARNs

Il modello definisce anche due AWS IoT dispositivi (oggetti), che rappresentano il dispositivo principale e il dispositivo client che vengono aggiunti al gruppo Greengrass.

Dopo aver creato lo stack con le risorse Greengrass, puoi utilizzare AWS CLI la console o la console [per distribuire AWS IoT](#) il gruppo.

Note

L'istruzione `CommandToDeployGroup` nell'esempio mostra come inviare un comando `create-deployment` completo dell'interfaccia a riga di comando che puoi utilizzare per distribuire il tuo gruppo.

JSON

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "AWS IoT Greengrass example template that creates a group version with a core, device, function, logger, subscription, and resources.",
  "Parameters": {
    "CoreCertificateArn": {
      "Type": "String"
```

```

    },
    "DeviceCertificateArn": {
      "Type": "String"
    },
    "LambdaVersionArn": {
      "Type": "String"
    }
  },
  "Resources": {
    "TestCore1": {
      "Type": "AWS::IoT::Thing",
      "Properties": {
        "ThingName": "TestCore1"
      }
    },
    "TestCoreDefinition": {
      "Type": "AWS::Greengrass::CoreDefinition",
      "Properties": {
        "Name": "DemoTestCoreDefinition"
      }
    },
    "TestCoreDefinitionVersion": {
      "Type": "AWS::Greengrass::CoreDefinitionVersion",
      "Properties": {
        "CoreDefinitionId": {
          "Ref": "TestCoreDefinition"
        },
        "Cores": [
          {
            "Id": "TestCore1",
            "CertificateArn": {
              "Ref": "CoreCertificateArn"
            },
            "SyncShadow": "false",
            "ThingArn": {
              "Fn::Join": [
                ":",
                [
                  "arn:aws:iot",
                  {
                    "Ref": "AWS::Region"
                  },
                  {
                    "Ref": "AWS::AccountId"

```

```

        },
        "thing/TestCore1"
    ]
  ]
}
}
],
},
"TestClientDevice1": {
  "Type": "AWS::IoT::Thing",
  "Properties": {
    "ThingName": "TestClientDevice1"
  }
},
"TestDeviceDefinition": {
  "Type": "AWS::Greengrass::DeviceDefinition",
  "Properties": {
    "Name": "DemoTestDeviceDefinition"
  }
},
"TestDeviceDefinitionVersion": {
  "Type": "AWS::Greengrass::DeviceDefinitionVersion",
  "Properties": {
    "DeviceDefinitionId": {
      "Fn::GetAtt": [
        "TestDeviceDefinition",
        "Id"
      ]
    },
    "Devices": [
      {
        "Id": "TestClientDevice1",
        "CertificateArn": {
          "Ref": "DeviceCertificateArn"
        },
        "SyncShadow": "true",
        "ThingArn": {
          "Fn::Join": [
            ":",
            [
              "arn:aws:iot",
              {
                "Ref": "AWS::Region"
              }
            ]
          ]
        }
      }
    ]
  }
}

```



```

        },
        {
            "Ref": "AWS::AccountId"
        },
        "thing/TestClientDevice1"
    ]
]
}
}
],
},
"TestFunctionDefinition": {
    "Type": "AWS::Greengrass::FunctionDefinition",
    "Properties": {
        "Name": "DemoTestFunctionDefinition"
    }
},
"TestFunctionDefinitionVersion": {
    "Type": "AWS::Greengrass::FunctionDefinitionVersion",
    "Properties": {
        "FunctionDefinitionId": {
            "Fn::GetAtt": [
                "TestFunctionDefinition",
                "Id"
            ]
        },
        "DefaultConfig": {
            "Execution": {
                "IsolationMode": "GreengrassContainer"
            }
        },
        "Functions": [
            {
                "Id": "TestLambda1",
                "FunctionArn": {
                    "Ref": "LambdaVersionArn"
                },
                "FunctionConfiguration": {
                    "Pinned": "true",
                    "Executable": "run.exe",
                    "ExecArgs": "argument1",
                    "MemorySize": "512",
                    "Timeout": "2000",

```

```

        "EncodingType": "binary",
        "Environment": {
            "Variables": {
                "variable1": "value1"
            },
            "ResourceAccessPolicies": [
                {
                    "ResourceId": "ResourceId1",
                    "Permission": "ro"
                },
                {
                    "ResourceId": "ResourceId2",
                    "Permission": "rw"
                }
            ],
            "AccessSysfs": "false",
            "Execution": {
                "IsolationMode": "GreengrassContainer",
                "RunAs": {
                    "Uid": "1",
                    "Gid": "10"
                }
            }
        }
    },
    "TestLoggerDefinition": {
        "Type": "AWS::Greengrass::LoggerDefinition",
        "Properties": {
            "Name": "DemoTestLoggerDefinition"
        }
    },
    "TestLoggerDefinitionVersion": {
        "Type": "AWS::Greengrass::LoggerDefinitionVersion",
        "Properties": {
            "LoggerDefinitionId": {
                "Ref": "TestLoggerDefinition"
            },
            "Loggers": [
                {
                    "Id": "TestLogger1",

```

```

        "Type": "AWSCloudWatch",
        "Component": "GreengrassSystem",
        "Level": "INFO"
    }
]
},
"TestResourceDefinition": {
    "Type": "AWS::Greengrass::ResourceDefinition",
    "Properties": {
        "Name": "DemoTestResourceDefinition"
    }
},
"TestResourceDefinitionVersion": {
    "Type": "AWS::Greengrass::ResourceDefinitionVersion",
    "Properties": {
        "ResourceDefinitionId": {
            "Ref": "TestResourceDefinition"
        },
        "Resources": [
            {
                "Id": "ResourceId1",
                "Name": "LocalDeviceResource",
                "ResourceDataContainer": {
                    "LocalDeviceResourceData": {
                        "SourcePath": "/dev/TestSourcePath1",
                        "GroupOwnerSetting": {
                            "AutoAddGroupOwner": "false",
                            "GroupOwner": "TestOwner"
                        }
                    }
                }
            },
            {
                "Id": "ResourceId2",
                "Name": "LocalVolumeResourceData",
                "ResourceDataContainer": {
                    "LocalVolumeResourceData": {
                        "SourcePath": "/dev/TestSourcePath2",
                        "DestinationPath": "/volumes/TestDestinationPath2",
                        "GroupOwnerSetting": {
                            "AutoAddGroupOwner": "false",
                            "GroupOwner": "TestOwner"
                        }
                    }
                }
            }
        ]
    }
}

```

```

    }
  }
}

},
"TestSubscriptionDefinition": {
  "Type": "AWS::Greengrass::SubscriptionDefinition",
  "Properties": {
    "Name": "DemoTestSubscriptionDefinition"
  }
},
"TestSubscriptionDefinitionVersion": {
  "Type": "AWS::Greengrass::SubscriptionDefinitionVersion",
  "Properties": {
    "SubscriptionDefinitionId": {
      "Ref": "TestSubscriptionDefinition"
    },
    "Subscriptions": [
      {
        "Id": "TestSubscription1",
        "Source": {
          "Fn::Join": [
            ":",
            [
              "arn:aws:iot",
              {
                "Ref": "AWS::Region"
              },
              {
                "Ref": "AWS::AccountId"
              },
              "thing/TestClientDevice1"
            ]
          ]
        },
        "Subject": "TestSubjectUpdated",
        "Target": {
          "Ref": "LambdaVersionArn"
        }
      }
    ]
  }
},
}

```

```

    "TestGroup": {
      "Type": "AWS::Greengrass::Group",
      "Properties": {
        "Name": "DemoTestGroupNewName",
        "RoleArn": {
          "Fn::Join": [
            ":",
            [
              "arn:aws:iam:",
              {
                "Ref": "AWS::AccountId"
              },
              "role/TestUser"
            ]
          ]
        },
        "InitialVersion": {
          "CoreDefinitionVersionArn": {
            "Ref": "TestCoreDefinitionVersion"
          },
          "DeviceDefinitionVersionArn": {
            "Ref": "TestDeviceDefinitionVersion"
          },
          "FunctionDefinitionVersionArn": {
            "Ref": "TestFunctionDefinitionVersion"
          },
          "SubscriptionDefinitionVersionArn": {
            "Ref": "TestSubscriptionDefinitionVersion"
          },
          "LoggerDefinitionVersionArn": {
            "Ref": "TestLoggerDefinitionVersion"
          },
          "ResourceDefinitionVersionArn": {
            "Ref": "TestResourceDefinitionVersion"
          }
        },
        "Tags": {
          "KeyName0": "value",
          "KeyName1": "value",
          "KeyName2": "value"
        }
      }
    },
  },
},

```

```

"Outputs": {
  "CommandToDeployGroup": {
    "Value": {
      "Fn::Join": [
        " ",
        [
          "groupVersion=$(cut -d'/' -f6 <<<",
          {
            "Fn::GetAtt": [
              "TestGroup",
              "LatestVersionArn"
            ]
          },
          ")",
          "aws --region",
          {
            "Ref": "AWS::Region"
          },
          "greengrass create-deployment --group-id",
          {
            "Ref": "TestGroup"
          },
          "--deployment-type NewDeployment --group-version-id",
          "$groupVersion"
        ]
      ]
    }
  }
}

```

YAML

```

AWSTemplateFormatVersion: 2010-09-09
Description: >-
  AWS IoT Greengrass example template that creates a group version with a core,
  device, function, logger, subscription, and resources.
Parameters:
  CoreCertificateArn:
    Type: String
  DeviceCertificateArn:
    Type: String
  LambdaVersionArn:

```

```
Type: String
Resources:
  TestCore1:
    Type: 'AWS::IoT::Thing'
    Properties:
      ThingName: TestCore1
  TestCoreDefinition:
    Type: 'AWS::Greengrass::CoreDefinition'
    Properties:
      Name: DemoTestCoreDefinition
  TestCoreDefinitionVersion:
    Type: 'AWS::Greengrass::CoreDefinitionVersion'
    Properties:
      CoreDefinitionId: !Ref TestCoreDefinition
      Cores:
        - Id: TestCore1
          CertificateArn: !Ref CoreCertificateArn
          SyncShadow: 'false'
          ThingArn: !Join
            - ':'
            - - 'arn:aws:iot'
              - !Ref 'AWS::Region'
              - !Ref 'AWS::AccountId'
              - thing/TestCore1
  TestClientDevice1:
    Type: 'AWS::IoT::Thing'
    Properties:
      ThingName: TestClientDevice1
  TestDeviceDefinition:
    Type: 'AWS::Greengrass::DeviceDefinition'
    Properties:
      Name: DemoTestDeviceDefinition
  TestDeviceDefinitionVersion:
    Type: 'AWS::Greengrass::DeviceDefinitionVersion'
    Properties:
      DeviceDefinitionId: !GetAtt
        - TestDeviceDefinition
        - Id
      Devices:
        - Id: TestClientDevice1
          CertificateArn: !Ref DeviceCertificateArn
          SyncShadow: 'true'
          ThingArn: !Join
            - ':'
```

```

      - - 'arn:aws:iot'
      - !Ref 'AWS::Region'
      - !Ref 'AWS::AccountId'
      - thing/TestClientDevice1
TestFunctionDefinition:
  Type: 'AWS::Greengrass::FunctionDefinition'
  Properties:
    Name: DemoTestFunctionDefinition
TestFunctionDefinitionVersion:
  Type: 'AWS::Greengrass::FunctionDefinitionVersion'
  Properties:
    FunctionDefinitionId: !GetAtt
      - TestFunctionDefinition
      - Id
    DefaultConfig:
      Execution:
        IsolationMode: GreengrassContainer
    Functions:
      - Id: TestLambda1
        FunctionArn: !Ref LambdaVersionArn
        FunctionConfiguration:
          Pinned: 'true'
          Executable: run.exe
          ExecArgs: argument1
          MemorySize: '512'
          Timeout: '2000'
          EncodingType: binary
          Environment:
            Variables:
              variable1: value1
            ResourceAccessPolicies:
              - ResourceId: ResourceId1
                Permission: ro
              - ResourceId: ResourceId2
                Permission: rw
          AccessSysfs: 'false'
          Execution:
            IsolationMode: GreengrassContainer
            RunAs:
              Uid: '1'
              Gid: '10'
TestLoggerDefinition:
  Type: 'AWS::Greengrass::LoggerDefinition'
  Properties:

```



```
    Name: DemoTestLoggerDefinition
TestLoggerDefinitionVersion:
  Type: 'AWS::Greengrass::LoggerDefinitionVersion'
  Properties:
    LoggerDefinitionId: !Ref TestLoggerDefinition
    Loggers:
      - Id: TestLogger1
        Type: AWSCloudWatch
        Component: GreengrassSystem
        Level: INFO
TestResourceDefinition:
  Type: 'AWS::Greengrass::ResourceDefinition'
  Properties:
    Name: DemoTestResourceDefinition
TestResourceDefinitionVersion:
  Type: 'AWS::Greengrass::ResourceDefinitionVersion'
  Properties:
    ResourceDefinitionId: !Ref TestResourceDefinition
    Resources:
      - Id: ResourceId1
        Name: LocalDeviceResource
        ResourceDataContainer:
          LocalDeviceResourceData:
            SourcePath: /dev/TestSourcePath1
            GroupOwnerSetting:
              AutoAddGroupOwner: 'false'
              GroupOwner: TestOwner
      - Id: ResourceId2
        Name: LocalVolumeResourceData
        ResourceDataContainer:
          LocalVolumeResourceData:
            SourcePath: /dev/TestSourcePath2
            DestinationPath: /volumes/TestDestinationPath2
            GroupOwnerSetting:
              AutoAddGroupOwner: 'false'
              GroupOwner: TestOwner
TestSubscriptionDefinition:
  Type: 'AWS::Greengrass::SubscriptionDefinition'
  Properties:
    Name: DemoTestSubscriptionDefinition
TestSubscriptionDefinitionVersion:
  Type: 'AWS::Greengrass::SubscriptionDefinitionVersion'
  Properties:
    SubscriptionDefinitionId: !Ref TestSubscriptionDefinition
```

```

Subscriptions:
- Id: TestSubscription1
  Source: !Join
    - ':'
    - - 'arn:aws:iot'
      - !Ref 'AWS::Region'
      - !Ref 'AWS::AccountId'
      - thing/TestClientDevice1
  Subject: TestSubjectUpdated
  Target: !Ref LambdaVersionArn

TestGroup:
Type: 'AWS::Greengrass::Group'
Properties:
Name: DemoTestGroupNewName
RoleArn: !Join
  - ':'
  - - 'arn:aws:iam:'
    - !Ref 'AWS::AccountId'
    - role/TestUser
InitialVersion:
CoreDefinitionVersionArn: !Ref TestCoreDefinitionVersion
DeviceDefinitionVersionArn: !Ref TestDeviceDefinitionVersion
FunctionDefinitionVersionArn: !Ref TestFunctionDefinitionVersion
SubscriptionDefinitionVersionArn: !Ref TestSubscriptionDefinitionVersion
LoggerDefinitionVersionArn: !Ref TestLoggerDefinitionVersion
ResourceDefinitionVersionArn: !Ref TestResourceDefinitionVersion
Tags:
KeyName0: value
KeyName1: value
KeyName2: value

Outputs:
CommandToDeployGroup:
Value: !Join
  - ' '
  - - groupVersion=$(cut -d'/' -f6 <<<
    - !GetAtt
      - TestGroup
      - LatestVersionArn
    - );
    - aws --region
    - !Ref 'AWS::Region'
    - greengrass create-deployment --group-id
    - !Ref TestGroup
    - '--deployment-type NewDeployment --group-version-id'

```

- \$groupVersion

Supportato s Regione AWS

Attualmente, puoi creare e gestire AWS IoT Greengrass risorse solo nei seguenti [Regione AWS casi](#):

- Stati Uniti orientali (Ohio)
- Stati Uniti orientali (Virginia settentrionale)
- US West (Oregon)
- Asia Pacifico (Mumbai)
- Asia Pacifico (Seoul)
- Asia Pacifico (Singapore)
- Asia Pacifico (Sydney)
- Asia Pacifico (Tokyo)
- Cina (Pechino)
- Europa (Francoforte)
- Europa (Irlanda)
- Europa (Londra)
- AWS GovCloud (Stati Uniti occidentali)

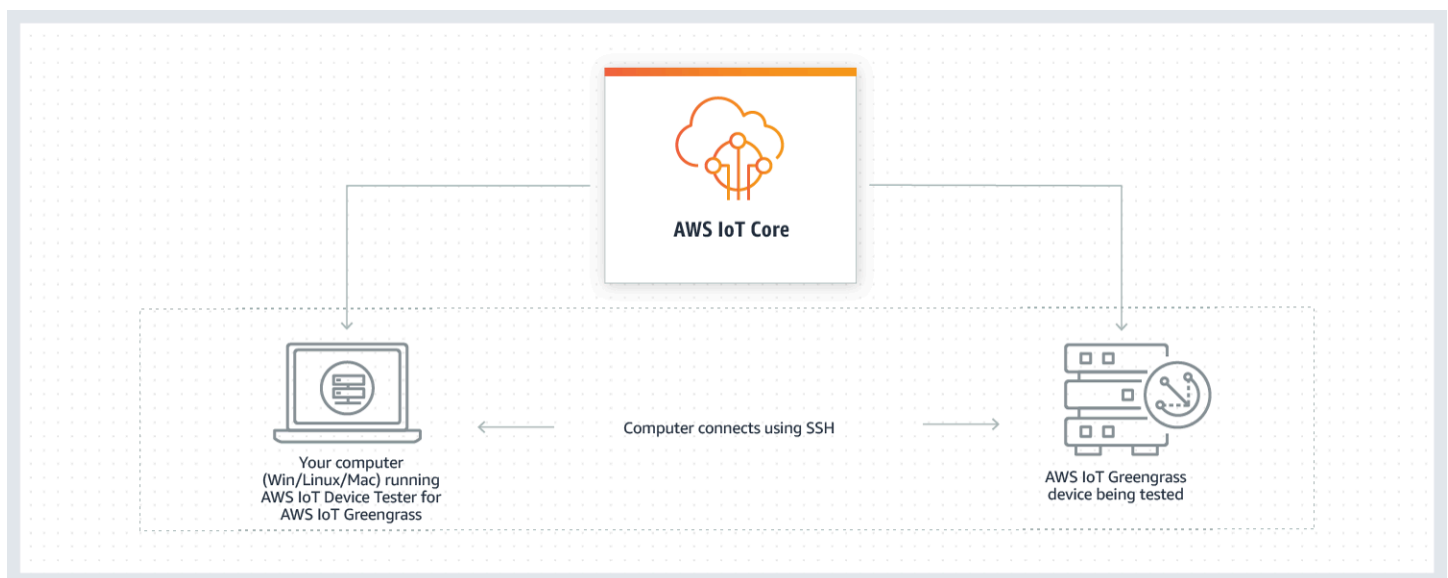
Utilizzo di AWS IoT Device Tester per AWS IoT Greengrass V1

AWS IoT Device Tester (IDT) è un framework di test scaricabile che consente di convalidare i dispositivi IoT. Poiché AWS IoT Greengrass Version 1 è stato spostato in [modalità di manutenzione](#), IDT for AWS IoT Greengrass V1 non genera più report di qualificazione firmati. Non sarà più possibile qualificare nuovi AWS IoT Greengrass V1 dispositivi per inserirli nel [Catalogo dei dispositivi tramite il AWS PartnerAWS Device](#) Qualification Program. Tuttavia, puoi continuare a utilizzare IDT per AWS IoT Greengrass V1 testare i tuoi dispositivi Greengrass V1. [Ti consigliamo di utilizzare IDT per qualificare ed AWS IoT Greengrass V2 elencare i dispositivi Greengrass nel Device Catalog.AWS Partner](#)

IDT for AWS IoT Greengrass viene eseguito sul computer host (Windows, macOS o Linux) collegato al dispositivo da testare. Esegue i test e aggrega i risultati. Inoltre offre un'interfaccia a riga di comando per gestire l'esecuzione di test.

AWS IoT Greengrass suite di qualificazione

Utilizzate IDT per AWS IoT Greengrass verificare che il software AWS IoT Greengrass Core funzioni sul vostro hardware e sia in grado di comunicare con. Cloud AWS Esegue anche end-to-end test con AWS IoT Core. Ad esempio, verifica che il dispositivo sia in grado di inviare e ricevere messaggi MQTT ed elaborarli correttamente.



AWS IoT Device Tester for AWS IoT Greengrass organizza i test utilizzando i concetti di suite di test e gruppi di test.

- Una suite di test è l'insieme di gruppi di test utilizzati per verificare che un dispositivo funzioni con versioni particolari di AWS IoT Greengrass.
- Un gruppo di test è l'insieme di singoli test relativi a una particolare funzionalità, ad esempio distribuzioni di gruppi Greengrass e messaggistica MQTT.

Per ulteriori informazioni, consulta [Usa IDT per gestire la suite di AWS IoT Greengrass qualifiche](#).

Suite di test personalizzate

A partire da IDT v4.0.0, IDT for AWS IoT Greengrass combina una configurazione di configurazione e un formato di risultati standardizzati con un ambiente di suite di test che consente di sviluppare suite di test personalizzate per i dispositivi e il software del dispositivo. Potete aggiungere test personalizzati per la vostra convalida interna o fornirli ai clienti per la verifica dei dispositivi.

Il modo in cui un testwriter configura una suite di test personalizzata determina le configurazioni delle impostazioni necessarie per eseguire suite di test personalizzate. Per ulteriori informazioni, consulta [Usa IDT per sviluppare ed eseguire le tue suite di test](#).

Versioni supportate di AWS IoT Device Tester for AWS IoT Greengrass V1

Poiché AWS IoT Greengrass Version 1 è stato spostato in [modalità di manutenzione](#), IDT for AWS IoT Greengrass V1 non genera più report di qualificazione firmati. Ti consigliamo di utilizzare [IDT](#) per AWS IoT Greengrass V2

Per informazioni su IDT per AWS IoT Greengrass V2, consulta [Using AWS IoT Device Tester for AWS IoT Greengrass V2 nella Device](#) Guide.AWS IoT Greengrass V2

Note

Riceverai una notifica quando avvii un test se IDT for non AWS IoT Greengrass è compatibile con la versione che stai utilizzando. AWS IoT Greengrass

Scaricando il software accetti l'[accordo di licenza di AWS IoT Device Tester](#).

Versioni IDT non supportate per AWS IoT Greengrass

Questo argomento elenca le versioni non supportate di IDT per AWS IoT Greengrass. Le versioni non supportate non ricevono correzioni di bug o aggiornamenti. Per ulteriori informazioni, consulta [the section called “Politica di supporto per AWS IoT Device Tester per AWS IoT Greengrass V1”](#).

IDT v4.4.1 per le versioni v1.11.6, v1.10.5 AWS IoT Greengrass

Note di rilascio:

- Consente di convalidare e qualificare i dispositivi che eseguono il software di base v1.11.6 e v1.10.5. AWS IoT Greengrass
- Contiene correzioni di bug minori.

Versione della suite di test:

GGQ_1.3.1

- Rilasciato il 2021.12.20

IDT v4.1.0 per le versioni v1.11.4, v1.10.4 AWS IoT Greengrass

Note di rilascio:

- Consente di convalidare e qualificare i dispositivi che eseguono il software di base v1.11.4 e v1.10.4. AWS IoT Greengrass
- Risolve un problema a causa del quale i log visualizzati durante un'esecuzione di test utilizzavano tag ridondanti.

Versione della suite di test:

GGQ_1.3.0

- Rilasciato il 23 giugno 2020
- Aggiunge nuovi tentativi per le chiamate API a Lambda, IAM AWS STS e per migliorare la gestione dei problemi di throttling o del server.
- Aggiunge il supporto per Python 3.8 ai test case ML e Docker.

IDT v4.0.2 per le versioni v1.11.1, v1.11.0, v1.10.3 AWS IoT Greengrass

Note di rilascio:

- È stato risolto un problema che impediva a IDT di mascherare gli errori HSI (Hardware Security Integration).

- Consente di sviluppare ed eseguire suite di test personalizzate utilizzando AWS IoT Device Tester for. AWS IoT Greengrass Per ulteriori informazioni, consulta [Usa IDT per sviluppare ed eseguire le tue suite di test.](#)
- Fornisce applicazioni IDT con firma di codice per macOS e Windows. In macOS, se viene visualizzato un messaggio di avviso di sicurezza, potrebbe essere necessario concedere un'eccezione di sicurezza per IDT. Per ulteriori informazioni, consulta [Eccezione di sicurezza su macOS.](#)

 Note

AWS IoT Greengrass non fornisce un Dockerfile o un'immagine Docker per la versione 1.11.1 del software di base. AWS IoT Greengrass Per testare il tuo dispositivo per la qualificazione Docker, usa una versione precedente del software di base. AWS IoT Greengrass

IDT v3.2.0 per le AWS IoT Greengrass versioni v1.11.0, v1.10.1, v1.10.0

Note di rilascio:

- Per impostazione predefinita, IDT esegue solo i test necessari per la qualificazione. Per qualificarsi per le funzionalità aggiuntive, è possibile modificare il file. [device.json](#)
- È stato aggiunto un numero di porta `device.json` che è possibile configurare per le connessioni SSH.
- Docker supporta solo [stream manager](#) e machine learning (ML) senza containerizzazione. Container, Docker e Hardware Security Integration (HSI) non sono disponibili per i dispositivi Docker.
- Ci siamo uniti `device-ml.json` e siamo entrati. `device-hsm.json` `device.json`

IDT v3.1.3 per le AWS IoT Greengrass versioni: v1.10.x, v1.9.x, v1.8.x

Note di rilascio:

- È stato aggiunto il supporto per la qualificazione delle funzionalità ML per AWS IoT Greengrass v1.10.x e v1.9.x. È ora possibile utilizzare IDT per verificare che i dispositivi possano eseguire l'inferenza ML localmente con modelli archiviati e addestrati nel cloud.

- Aggiunto `--stop-on-first-failure` per il comando `run-suite`. È possibile utilizzare questa opzione per configurare IDT in modo che interrompa l'esecuzione al primo errore. Si consiglia di utilizzare questa opzione durante la fase di debug a livello di gruppi di test.
- È stato aggiunto un controllo della deriva dell'orologio per i test MQTT per garantire che il dispositivo sottoposto a test utilizzi l'ora di sistema corretta. Il tempo utilizzato deve rientrare in un intervallo di tempo accettabile.
- Aggiunto `--update-idt` per il comando `run-suite`. È possibile utilizzare questa opzione per impostare la risposta per il prompt di aggiornamento IDT.
- Aggiunto `--update-managed-policy` per il comando `run-suite`. È possibile utilizzare questa opzione per impostare la risposta alla richiesta di aggiornamento della politica gestita.
- È stata aggiunta una correzione di bug per gli aggiornamenti automatici delle versioni della suite di test IDT. La correzione garantisce che IDT possa eseguire le suite di test più recenti disponibili per la tua AWS IoT Greengrass versione.

IDT v3.0.1 per AWS IoT Greengrass

Note di rilascio:

- È stato aggiunto il supporto per la v1.10.1. AWS IoT Greengrass
- Aggiornamenti automatici delle versioni della suite di test IDT. IDT può scaricare le suite di test più recenti disponibili per la tua versione. AWS IoT Greengrass Con questa funzione:
 - Le versioni della suite di test sono definite utilizzando un formato *major.minor.patch*. La versione iniziale della suite di test è GGQ_1.0.0.
 - È possibile scaricare nuove suite di test in modo interattivo nell'interfaccia della riga di comando o impostare il flag `upgrade-test-suite` all'avvio di IDT.

Per ulteriori informazioni, consulta [the section called “Versioni della suite di test”](#).

- Aggiunto `list-supported-products`. È possibile utilizzare questo comando per elencare le versioni AWS IoT Greengrass e delle suite di test supportate dalla versione installata di IDT.
- Aggiunto `list-test-cases`. È possibile utilizzare questo comando per elencare i casi di test disponibili in un gruppo di test.
- Aggiunto `test-id` per il comando `run-suite`. È possibile utilizzare questa opzione per eseguire singoli casi di test in un gruppo di test.

IDT v2.3.0 per v1.10, v1.9.x e AWS IoT Greengrass v1.8.x

Durante i test su un dispositivo fisico, sono supportate le versioni 1.10, 1.9.x e 1.8.x. AWS IoT Greengrass

Durante i test in un contenitore Docker, sono supportate le versioni 1.10 e 1.9.x. AWS IoT Greengrass

Note di rilascio:

- Aggiunta del supporto per [the section called “Esegui AWS IoT Greengrass in un contenitore Docker”](#). Ora puoi usare IDT per qualificare e convalidare che i tuoi dispositivi possano funzionare in un contenitore Docker. AWS IoT Greengrass
- È stata aggiunta una [policy AWS gestita](#) (AWSIoTDeviceTesterForGreengrassFullAccess) che definisce le autorizzazioni necessarie per eseguire Device Tester. AWS IoT Se le nuove versioni richiedono autorizzazioni aggiuntive, le AWS aggiunge a questa policy gestita in modo da non dover aggiornare le autorizzazioni IAM.
- Controlli introdotti per verificare che l'ambiente (ad esempio, connettività dei dispositivi e connettività Internet) sia impostato correttamente prima di eseguire i test case.
- Migliorato il correttore delle dipendenze Greengrass in IDT per renderlo più flessibile durante il controllo della libc sui dispositivi.

IDT v2.2.0 per v1.10, v1.9.x e AWS IoT Greengrass v1.8.x

Note di rilascio:

- È stato aggiunto il AWS IoT Greengrass supporto per la v1.10.
- Aggiunto il supporto per il connettore di [distribuzione dell'applicazione Greengrass Docker](#).
- Aggiunto il supporto per AWS IoT Greengrass [lo stream manager](#).
- È stato aggiunto il supporto per AWS IoT Greengrass la regione Cina (Pechino).

IDT v2.1.0 per AWS IoT Greengrass v1.9.x, v1.8.x e v1.7.x

Note di rilascio:

- È stato aggiunto il supporto per la versione AWS IoT Greengrass 1.9.4.
- È stato aggiunto il supporto per i dispositivi Linux. ARMv6l

IDT v2.0.0 per v1.9.3, v1.9.2, v1.9.1, AWS IoT Greengrass v1.9.0, v1.8.4, v1.8.3 e v1.8.2

Note di rilascio:

- Rimossa la dipendenza su Python per il dispositivo sottoposto a test.
- Tempo di esecuzione della suite di test ridotto di oltre il 50%, rendendo il processo di qualifica più veloce.
- Dimensioni eseguibili ridotte di oltre il 50%, rendendo il download e l'installazione più veloci.
- [Supporto moltiplicatore di timeout](#) migliorato per tutti i casi di test.
- Messaggi post-diagnostici migliorati per risolvere gli errori più rapidamente.
- Aggiornato il modello di criteri di autorizzazione richiesto per eseguire IDT.
- È stato aggiunto il supporto per la versione 1.9.3. AWS IoT Greengrass

IDT v1.3.3 per v1.9.2, v1.9.1, AWS IoT Greengrass v1.9.0, v1.8.3 e v1.8.2

Note di rilascio:

- Aggiunto il supporto per Greengrass v1.9.2 e v1.8.3
- Aggiunto il supporto per Greengrass OpenWrt.
- Aggiunto l'accesso dispositivo tramite nome utente e password SSH.
- Aggiunta una correzione di bug di test nativa per OpenWrt - ARMv7l platform.

IDT v1.2 per v1.8.1 AWS IoT Greengrass

Note di rilascio:

- Aggiunto un moltiplicatore di timeout configurabile per correggere e risolvere i problemi di timeout (ad esempio, connessioni a bassa larghezza di banda).

IDT v1.1 per v1.8.0 AWS IoT Greengrass

Note di rilascio:

- È stato aggiunto il supporto per AWS IoT Greengrass Hardware Security Integration (HSI).
- È stato aggiunto il supporto per AWS IoT Greengrass container e no container.

- Aggiunta la creazione automatizzata dei ruoli di AWS IoT Greengrass servizio.
- Funzione di eliminazione delle risorse di test migliorata.
- Aggiunto report di riepilogo per l'esecuzione di test.

IDT v1.1 per v1.7.1 AWS IoT Greengrass

Note di rilascio:

- È stato aggiunto il supporto per AWS IoT Greengrass Hardware Security Integration (HSI).
- È stato aggiunto il supporto per AWS IoT Greengrass container e no container.
- Aggiunta la creazione automatizzata dei ruoli di AWS IoT Greengrass servizio.
- Funzione di eliminazione delle risorse di test migliorata.
- Aggiunto report di riepilogo per l'esecuzione di test.

IDT v1.0 per v1.6.1 AWS IoT Greengrass

Note di rilascio:

- Aggiunta la correzione di bug del test OTA per la compatibilità con le AWS IoT Greengrass versioni future.

Note

[Se utilizzi IDT v1.0 per AWS IoT Greengrass v1.6.1, devi creare un ruolo di servizio Greengrass.](#) Nelle versioni successive, IDT crea il ruolo del servizio per l'utente.

Usa IDT per gestire la suite di AWS IoT Greengrass qualifiche

È possibile utilizzare AWS IoT Device Tester (IDT) per AWS IoT Greengrass verificare che il software AWS IoT Greengrass Core funzioni sull'hardware e sia in grado di comunicare con. Cloud AWS Esegue anche end-to-end test con. AWS IoT Core Ad esempio, verifica che il dispositivo sia in grado di inviare e ricevere messaggi MQTT ed elaborarli correttamente.

Poiché AWS IoT Greengrass Version 1 è stato spostato in [modalità di manutenzione](#), IDT AWS IoT Greengrass V1 non genera più report di qualificazione firmati. Se desideri aggiungere il tuo hardware al AWS Partner Device Catalog, esegui la suite di AWS IoT Greengrass V2 qualificazione

per generare report di test da inviare. AWS IoT Per ulteriori informazioni, consulta [AWS Device Qualification Program](#) e [Versioni supportate di IDT](#) for. AWS IoT Greengrass V2

Oltre a testare i dispositivi, IDT for AWS IoT Greengrass crea risorse (ad esempio AWS IoT oggetti, AWS IoT Greengrass gruppi, funzioni Lambda e così via) per facilitare Account AWS il processo di qualificazione.

Per creare queste risorse, IDT for AWS IoT Greengrass utilizza le AWS credenziali configurate nel `config.json` file per effettuare chiamate API per conto dell'utente. Il provisioning di queste risorse viene effettuato varie volte nel corso di un test.

Quando utilizzate IDT per eseguire la suite AWS IoT Greengrass di AWS IoT Greengrass qualifiche, IDT esegue le seguenti operazioni:

1. Carica e convalida le configurazioni del dispositivo e delle credenziali.
2. Esegue i test selezionati con le risorse locali e cloud richieste.
3. Esegue la pulizia di risorse locali e cloud.
4. Genera report sui test che indicano se il dispositivo ha superato i test richiesti per la qualificazione.

Versioni della suite di test

IDT for AWS IoT Greengrass organizza i test in suite di test e gruppi di test.

- Una suite di test è l'insieme di gruppi di test utilizzati per verificare che un dispositivo funzioni con versioni particolari di AWS IoT Greengrass.
- Un gruppo di test è l'insieme di singoli test relativi a una particolare funzionalità, ad esempio distribuzioni di gruppi Greengrass e messaggistica MQTT.

A partire da IDT v3.0.0, le versioni delle suite di test sono create utilizzando un formato *major.minor.patch*, ad esempio GGQ_1.0.0. Quando scarichi IDT, il pacchetto include la versione più recente della suite di test.

Important

IDT supporta le tre versioni più recenti della suite di test per la qualifica dei dispositivi. Per ulteriori informazioni, consulta [the section called “Politica di supporto per AWS IoT Device Tester per AWS IoT Greengrass V1”](#).

È possibile eseguire l'`list-supported-products` operazione per elencare le versioni AWS IoT Greengrass e le suite di test supportate dalla versione corrente di IDT. I test delle versioni non supportate della suite di test non sono validi per la qualifica del dispositivo. IDT non stampa i report di qualifica per le versioni non supportate.

Aggiornamenti alle impostazioni di configurazione IDT

Nuovi test potrebbero introdurre nuove impostazioni di configurazione IDT.

- Se le impostazioni sono facoltative, IDT continua a eseguire i test.
- Se le impostazioni sono necessarie, IDT invia una notifica all'utente e interrompe l'esecuzione. Dopo aver configurato le impostazioni, riavviare l'esecuzione del test.

Le impostazioni di configurazione si trovano nella cartella `<device-tester-extract-location>/configs`. Per ulteriori informazioni, consulta [the section called “Configurare le impostazioni IDT”](#).

Se una versione aggiornata della suite di test aggiunge impostazioni di configurazione, IDT crea una copia del file di configurazione originale in `<device-tester-extract-location>/configs`.

Descrizioni dei gruppi di test

IDT v2.0.0 and later

Gruppi di test richiesti per la qualificazione principale

Questi gruppi di test sono necessari per qualificare il AWS IoT Greengrass dispositivo per il AWS Partner Device Catalog.

AWS IoT Greengrass Dipendenze principali

Verifica che il dispositivo soddisfi tutti i requisiti software e hardware per il software AWS IoT Greengrass Core.

Il caso `Software Packages Dependencies` di test in questo gruppo di test non è applicabile quando si esegue il test in un [contenitore Docker](#).

Implementazione

Verifica che le funzioni Lambda possano essere implementate sul tuo dispositivo.

MQTT

Verifica la funzionalità del router di AWS IoT Greengrass messaggi controllando la comunicazione locale tra i dispositivi core di Greengrass e i dispositivi client, che sono dispositivi IoT locali.

Over-the-Air (OTA)

Verifica che il dispositivo sia in grado di eseguire correttamente un aggiornamento OTA del software AWS IoT Greengrass Core.

Questo gruppo di test non è applicabile quando si esegue il test in un [contenitore Docker](#).

Versione

Verifica che la versione AWS IoT Greengrass fornita sia compatibile con la versione di AWS IoT Device Tester in uso.

Gruppi di test facoltativi

Questi gruppi di test sono facoltativi. Se scegli di qualificarti per i test opzionali, il tuo dispositivo viene elencato con funzionalità aggiuntive nel Catalogo dei AWS Partner dispositivi.

Dipendenze del container

Verifica che il dispositivo soddisfi tutti i requisiti software e hardware per eseguire le funzioni Lambda in modalità contenitore su un core Greengrass.

Questo gruppo di test non è applicabile quando si esegue il test in un [contenitore Docker](#).

Container di distribuzione

Verifica che le funzioni Lambda possano essere implementate sul dispositivo ed eseguite in modalità contenitore su un core Greengrass.

Questo gruppo di test non è applicabile quando si esegue il test in un [contenitore Docker](#).

Dipendenze Docker (supportate per IDT v2.2.0 e versioni successive)

Verifica che il dispositivo soddisfi tutte le dipendenze tecniche richieste per utilizzare il connettore di distribuzione delle applicazioni Greengrass Docker per eseguire i contenitori

Questo gruppo di test non è applicabile quando si esegue il test in un [contenitore Docker](#).

Integrazione della sicurezza hardware (HSI)

Verifica che la libreria condivisa HSI fornita possa interfacciarsi con il modulo di sicurezza hardware (HSM) e implementa correttamente il PKCS #11 richiesto. APIs La HSM e la

libreria condivisa devono essere in grado di accedere a un CSR, eseguire operazioni TLS e fornire le lunghezze di chiave e l'algoritmo chiave pubblica corretti.

Dipendenze Stream Manager (supportate per IDT v2.2.0 e versioni successive)

Verifica che il dispositivo soddisfi tutte le dipendenze tecniche richieste per eseguire stream manager. AWS IoT Greengrass

Dipendenze di machine learning (supportate per IDT v3.1.0 e versioni successive)

Verifica che il dispositivo soddisfi tutte le dipendenze tecniche richieste per eseguire localmente l'inferenza di ML.

Test di inferenza di machine learning (supportati per IDT v3.1.0 e versioni successive)

Verifica che l'inferenza ML possa essere eseguita sul dispositivo in esame. Per ulteriori informazioni, consulta [the section called “Facoltativo: configurazione del dispositivo per la qualificazione di ML”](#).

Test del container di inferenza di machine learning (supportati per IDT v3.1.0 e versioni successive)

Verifica che l'inferenza di ML può essere eseguita sul dispositivo dato in fase di test ed eseguito in modalità contenitore su un Greengrass Core. Per ulteriori informazioni, consulta [the section called “Facoltativo: configurazione del dispositivo per la qualificazione di ML”](#).

IDT v1.3.3 and earlier

Gruppi di test richiesti per la qualificazione principale

Questi test sono necessari per qualificare il AWS IoT Greengrass dispositivo per il Device Catalog. AWS Partner

AWS IoT Greengrass Dipendenze principali

Verifica che il dispositivo soddisfi tutti i requisiti software e hardware per il software AWS IoT Greengrass Core.

Combinazione (interazione sicurezza dispositivi)

Questo gruppo di test verifica la funzionalità del Device Certificate Manager di Greengrass Core e di IP Detector modificando le informazioni sulla connettività sul gruppo Greengrass nel cloud. Il gruppo di test ruota il certificato del AWS IoT Greengrass server e verifica che AWS IoT Greengrass consenta le connessioni.

Distribuzione (necessaria per IDT v1.2 e versioni precedenti)

Verifica che le funzioni Lambda possano essere implementate sul tuo dispositivo.

Device Certificate Manager (DCM)

Verifica che il gestore dei certificati del AWS IoT Greengrass dispositivo sia in grado di generare un certificato del server all'avvio e di ruotare i certificati se sono prossimi alla scadenza.

Rilevamento IP (IPD)

Il gruppo di test IPD verifica che le informazioni di connettività core vengano aggiornate quando si verificano modifiche IP in un dispositivo Greengrass Core. Per ulteriori informazioni, consulta [Attivazione del rilevamento automatico dell'IP](#).

Registrazione

Verifica che il servizio di AWS IoT Greengrass registrazione possa scrivere su un file di registro utilizzando una funzione Lambda utente scritta in Python.

MQTT

Verifica la funzionalità del AWS IoT Greengrass router di messaggi inviando messaggi su un argomento indirizzato a due funzioni Lambda.

Nativo

Verifica che sia AWS IoT Greengrass possibile eseguire funzioni Lambda native (compilate).

Over-the-Air (OTA)

Verifica che il dispositivo sia in grado di eseguire correttamente un aggiornamento OTA del software AWS IoT Greengrass Core.

Intrusione

[Verifica che il software AWS IoT Greengrass Core non si avvii se la protezione hardlink/soft link e seccomp non sono abilitati](#). Viene inoltre usato per verificare altre caratteristiche correlate alla sicurezza.

Shadow

Verifica la funzionalità di sincronizzazione cloud shadow e shadow locale.

Spooler

Il gruppo di test spooler verifica che i messaggi MQTT vengano messi in coda con la configurazione spooler predefinita.

Token Exchange Service (TES)

Verifica di AWS IoT Greengrass poter scambiare il certificato di base con credenziali valide.
AWS

Versione

Verifica che la versione AWS IoT Greengrass fornita sia compatibile con la versione di AWS IoT Device Tester in uso.

Gruppi di test facoltativi

Questi test sono facoltativi. Se scegli di qualificarti per i test opzionali, il tuo dispositivo viene elencato con funzionalità aggiuntive nel Catalogo dei AWS Partner dispositivi.

Dipendenze del container

Verifica che il dispositivo soddisfi tutte le dipendenze richieste per eseguire le funzioni Lambda in modalità contenitore.

Integrazione della sicurezza hardware (HSI)

Verifica che la libreria condivisa HSI fornita sia in grado di interfacciarsi con il modulo di sicurezza hardware (HSM) e implementa correttamente il PKCS #11 richiesto. APIs La HSM e la libreria condivisa devono essere in grado di accedere a un CSR, eseguire operazioni TLS e fornire le lunghezze di chiave e l'algoritmo chiave pubblica corretti.

Accesso alle risorse locali

Verifica la funzionalità di accesso alle risorse locali (LRA) AWS IoT Greengrass fornendo accesso a file e directory locali di proprietà di vari utenti e gruppi Linux alle funzioni Lambda containerizzate tramite LRA. AWS IoT Greengrass APIs Alle funzioni Lambda dovrebbe essere consentito o negato l'accesso alle risorse locali in base alla configurazione dell'accesso alle risorse locali.

Rete

Verifica che le connessioni socket possano essere stabilite da una funzione Lambda. Queste connessioni socket devono essere consentite o negate in base alla configurazione di Greengrass Core.

Prerequisiti per l'esecuzione della suite di AWS IoT Greengrass qualifiche

Questa sezione descrive i prerequisiti per l'utilizzo di AWS IoT Device Tester (IDT) per l'esecuzione della suite AWS IoT Greengrass di qualifiche. AWS IoT Greengrass

Scarica la versione più recente di Device Tester per AWS IoT

AWS IoT Greengrass

Scaricate l'[ultima versione](#) di IDT ed estraete il software in una posizione del file system in cui disponete delle autorizzazioni di lettura e scrittura.

Note

IDT non supporta l'esecuzione da parte di più utenti da un percorso condiviso, ad esempio una directory NFS o una cartella condivisa di rete Windows. Si consiglia di estrarre il pacchetto IDT in un'unità locale ed eseguire il file binario IDT sulla workstation locale. In Windows esiste un limite di lunghezza del percorso di 260 caratteri. Se stai usando Windows, estrai IDT in una directory root come C:\ o D:\ per mantenere i percorsi entro il limite di 260 caratteri.

Crea e configura un Account AWS

Prima di poter utilizzare IDT per AWS IoT Greengrass, è necessario eseguire le seguenti operazioni:

1. [Crea un Account AWS](#). Se ne hai già uno Account AWS, vai al passaggio 2.
2. [Configura le autorizzazioni per IDT](#).

Queste autorizzazioni dell'account consentono a IDT di accedere ai AWS servizi e creare AWS risorse, come AWS IoT oggetti, gruppi Greengrass e funzioni Lambda, per tuo conto.

Per creare queste risorse, IDT for AWS IoT Greengrass utilizza le AWS credenziali configurate nel `config.json` file per effettuare chiamate API per conto dell'utente. Il provisioning di queste risorse viene effettuato varie volte nel corso di un test.

Note

Sebbene la maggior parte dei test sia [idonea per il piano gratuito di Amazon Web Services](#), devi fornire una carta di credito quando ti iscrivi a un Account AWS. Per ulteriori informazioni,

consulta [Perché ho bisogno di un metodo di pagamento se il mio account è coperto dal livello gratuito?](#)

Passaggio 1: crea un Account AWS

In questo passaggio, crea e configura un Account AWS. Se ne hai già uno Account AWS, vai [a the section called “Fase 2: configurazione delle autorizzazioni per IDT”](#).

Iscriviti per un Account AWS

Se non ne hai uno Account AWS, completa i seguenti passaggi per crearne uno.

Per iscriverti a un Account AWS

1. Apri la <https://portal.aws.amazon.com/billing/registrazione>.
2. Segui le istruzioni online.

Parte della procedura di registrazione prevede la ricezione di una telefonata o di un messaggio di testo e l'immissione di un codice di verifica sulla tastiera del telefono.

Quando ti iscrivi a un Account AWS, Utente root dell'account AWS viene creato un. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWS nell'account. Come best practice di sicurezza, assegna l'accesso amministrativo a un utente e utilizza solo l'utente root per eseguire [attività che richiedono l'accesso di un utente root](#).

AWS ti invia un'email di conferma dopo il completamento della procedura di registrazione. In qualsiasi momento, puoi visualizzare l'attività corrente del tuo account e gestirlo accedendo a <https://aws.amazon.com/> e scegliendo Il mio account.

Crea un utente con accesso amministrativo

Dopo esserti registrato Account AWS, proteggi Utente root dell'account AWS AWS IAM Identity Center, abilita e crea un utente amministrativo in modo da non utilizzare l'utente root per le attività quotidiane.

Proteggi i tuoi Utente root dell'account AWS

1. Accedi [AWS Management Console](#) come proprietario dell'account scegliendo Utente root e inserendo il tuo indirizzo Account AWS email. Nella pagina successiva, inserisci la password.

Per informazioni sull'accesso utilizzando un utente root, consulta la pagina [Signing in as the root user](#) della Guida per l'utente di Accedi ad AWS .

2. Abilita l'autenticazione a più fattori (MFA) per l'utente root.

Per istruzioni, consulta [Abilitare un dispositivo MFA virtuale per l'utente Account AWS root \(console\)](#) nella Guida per l'utente IAM.

Crea un utente con accesso amministrativo

1. Abilita Centro identità IAM.

Per istruzioni, consulta [Abilitazione di AWS IAM Identity Center](#) nella Guida per l'utente di AWS IAM Identity Center .

2. In IAM Identity Center, assegna l'accesso amministrativo a un utente.

Per un tutorial sull'utilizzo di IAM Identity Center directory come fonte di identità, consulta [Configurare l'accesso utente con l'impostazione predefinita IAM Identity Center directory](#) nella Guida per l'AWS IAM Identity Center utente.

Accesso come utente amministratore

- Per accedere con l'utente IAM Identity Center, utilizza l'URL di accesso che è stato inviato al tuo indirizzo e-mail quando hai creato l'utente IAM Identity Center.

Per informazioni sull'accesso utilizzando un utente IAM Identity Center, consulta [AWS Accedere al portale di accesso](#) nella Guida per l'Accedi ad AWS utente.

Assegna l'accesso a ulteriori utenti

1. In IAM Identity Center, crea un set di autorizzazioni conforme alla best practice dell'applicazione di autorizzazioni con il privilegio minimo.

Segui le istruzioni riportate nella pagina [Creazione di un set di autorizzazioni](#) nella Guida per l'utente di AWS IAM Identity Center .

2. Assegna al gruppo prima gli utenti e poi l'accesso con autenticazione unica (Single Sign-On).

Per istruzioni, consulta [Aggiungere gruppi](#) nella Guida per l'utente di AWS IAM Identity Center .

Fase 2: configurazione delle autorizzazioni per IDT

In questo passaggio, configura le autorizzazioni utilizzate da IDT for per eseguire test e raccogliere dati sull' AWS IoT Greengrass utilizzo di IDT. Puoi utilizzare AWS Management Console or AWS Command Line Interface (AWS CLI) per creare una policy IAM e un utente di test per IDT, quindi allegare le policy all'utente. Se è già stato creato un utente di test per IDT, passare a [the section called “Configura il tuo dispositivo per eseguire test IDT”](#) o [the section called “Facoltativo: configurazione del container Docker”](#).

- [Per configurare le autorizzazioni per IDT \(Console\)](#)
- [Per configurare le autorizzazioni per IDT \(AWS CLI\)](#)

Per configurare le autorizzazioni per IDT (Console)

Attenersi alla seguente procedura per utilizzare la console per configurare le autorizzazioni per IDT per AWS IoT Greengrass.

1. Accedere alla [console IAM](#).
2. Creare un criterio gestito dal cliente che concede le autorizzazioni per creare ruoli con autorizzazioni specifiche.
 - a. Nel riquadro di navigazione, seleziona Policy e quindi Crea policy.
 - b. Nella scheda JSON, sostituire il contenuto del segnaposto con la seguente policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ManageRolePoliciesForIDTGreengrass",
      "Effect": "Allow",
      "Action": [
        "iam:DetachRolePolicy",
        "iam:AttachRolePolicy"
      ],
      "Resource": [
        "arn:aws:iam::*:role/idt-*",
        "arn:aws:iam::*:role/GreengrassServiceRole"
      ],
      "Condition": {
        "ArnEquals": {
```

```

        "iam:PolicyARN": [
            "arn:aws:iam::aws:policy/service-role/
AWSGreengrassResourceAccessRolePolicy",
            "arn:aws:iam::aws:policy/service-role/
GreengrassOTAUpdateArtifactAccess",
            "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"
        ]
    }
},
{
    "Sid": "ManageRolesForIDTGreengrass",
    "Effect": "Allow",
    "Action": [
        "iam:CreateRole",
        "iam>DeleteRole",
        "iam:PassRole",
        "iam:GetRole"
    ],
    "Resource": [
        "arn:aws:iam::*:role/idt-*",
        "arn:aws:iam::*:role/GreengrassServiceRole"
    ]
}
]
}

```

Important

Il criterio seguente concede l'autorizzazione per creare e gestire i ruoli richiesti da IDT per AWS IoT Greengrass. Ciò include le autorizzazioni per allegare le seguenti politiche AWS gestite:

- [AWSGreengrassResourceAccessRolePolicy](#)
- [Greengrass OTAUpdate ArtifactAccess](#)
- [AWSLambdaBasicExecutionRole](#)

- c. Scegliere Next: Tags (Successivo: Tag).
- d. Scegliere Next:Review (Successivo: Rivedi).

- e. Per Nome, immetti **IDTGreengrassIAMPermissions**. In Riepilogo, esaminare le autorizzazioni concesse dai criteri.
 - f. Scegliere Create Policy (Crea policy).
 3. Crea un utente IAM e allega le autorizzazioni richieste da IDT for. AWS IoT Greengrass
 - a. Crea un utente IAM. Segui i passaggi da 1 a 5 in [Creazione di utenti IAM \(console\) nella Guida](#) per l'utente IAM.
 - b. Allega le autorizzazioni al tuo utente IAM:
 - i. Nella pagina Imposta le autorizzazioni, scegli Allega direttamente le politiche esistenti.
 - ii. Cerca la IDTGreengrassIAMPermissionspolitica che hai creato nel passaggio precedente. Selezionare la casella di controllo.
 - iii. Cerca la AWSIoTDeviceTesterForGreengrassFullAccesspolitica. Selezionare la casella di controllo.
-  **Note**

[AWSIoTDeviceTesterForGreengrassFullAccess](#) È una politica AWS gestita che definisce le autorizzazioni richieste da IDT per creare e accedere alle AWS risorse utilizzate per i test. Per ulteriori informazioni, consulta [the section called “AWS politica gestita per IDT”](#).
- c. Scegli Successivo: Tag.
 - d. Scegliere Next:Review per visualizzare un riepilogo delle tue scelte.
 - e. Selezionare Create user (Crea utente).
 - f. Per visualizzare le chiavi di accesso dell'utente (chiave di accesso IDs e chiavi di accesso segrete), scegli Mostra accanto alla password e alla chiave di accesso. Per salvare le chiavi di accesso, scegliere Scarica .csv e salvare il file in una posizione sicura. Utilizzerai queste informazioni in seguito per configurare il file AWS delle credenziali.
 4. Passaggio successivo: configurare il [dispositivo fisico](#).

Per configurare le autorizzazioni per IDT (AWS CLI)

Segui questi passaggi per utilizzare il file AWS CLI per configurare le autorizzazioni per IDT. AWS IoT Greengrass Se sono già state configurate le autorizzazioni nella console, passare a [the section called “Configura il tuo dispositivo per eseguire test IDT”](#) o [the section called “Facoltativo: configurazione del container Docker”](#).

1. Sul tuo computer, installa e configura il file AWS CLI se non è già installato. Segui la procedura descritta in [Installazione di AWS CLI nella Guida AWS Command Line Interface per l'utente](#).

Note

AWS CLI È uno strumento open source che puoi utilizzare per interagire con AWS i servizi dalla tua shell a riga di comando.

2. Creare una policy gestita dal cliente che conceda le autorizzazioni per gestire ruoli IDT e AWS IoT Greengrass .

Linux, macOS, or Unix

```
aws iam create-policy --policy-name IDTGreengrassIAMPermissions --policy-document '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ManageRolePoliciesForIDTGreengrass",
      "Effect": "Allow",
      "Action": [
        "iam:DetachRolePolicy",
        "iam:AttachRolePolicy"
      ],
      "Resource": [
        "arn:aws:iam::*:role/idt-*",
        "arn:aws:iam::*:role/GreengrassServiceRole"
      ],
      "Condition": {
        "ArnEquals": {
          "iam:PolicyARN": [
            "arn:aws:iam::aws:policy/service-role/AWSGreengrassResourceAccessRolePolicy",
            "arn:aws:iam::aws:policy/service-role/GreengrassOTAUpdateArtifactAccess",
```



```

        "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"
    ]
}
},
{
    "Sid": "ManageRolesForIDTGreengrass",
    "Effect": "Allow",
    "Action": [
        "iam:CreateRole",
        "iam>DeleteRole",
        "iam:PassRole",
        "iam:GetRole"
    ],
    "Resource": [
        "arn:aws:iam::*:role/idt-*",
        "arn:aws:iam::*:role/GreengrassServiceRole"
    ]
}
]
}'

```

Windows command prompt

```

aws iam create-policy --policy-name IDTGreengrassIAMPermissions --
policy-document '{"Version": "2012-10-17", "Statement": [{"Sid
": "ManageRolePoliciesForIDTGreengrass", "Effect": "Allow",
"Action": ["iam:DetachRolePolicy", "iam:AttachRolePolicy"],
"Resource": ["arn:aws:iam::*:role/idt-*", "arn:aws:iam::*:role/
GreengrassServiceRole"], "Condition": {"ArnEquals": {"iam:PolicyARN":
["arn:aws:iam::aws:policy/service-role/AWSGreengrassResourceAccessRolePolicy
", "arn:aws:iam::aws:policy/service-role/GreengrassOTAUpdateArtifactAccess
", "arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole"]}}},
{"Sid": "ManageRolesForIDTGreengrass", "Effect": "Allow", "Action":
["iam:CreateRole", "iam>DeleteRole", "iam:PassRole", "iam:GetRole
"], "Resource": ["arn:aws:iam::*:role/idt-*", "arn:aws:iam::*:role/
GreengrassServiceRole"]}]}'

```

 Note

Questo passaggio include un esempio del prompt dei comandi di Windows perché utilizza una sintassi JSON diversa rispetto ai comandi del terminale Linux, macOS o Unix.

3. Crea un utente IAM e allega le autorizzazioni richieste da IDT for. AWS IoT Greengrass

- a. Crea un utente IAM. In questa configurazione di esempio, l'utente viene chiamato `IDTGreengrassUser`.

```
aws iam create-user --user-name IDTGreengrassUser
```

- b. Allega la `IDTGreengrassIAMPermissions` policy che hai creato nel passaggio 2 al tuo utente IAM. Sostituiscilo `<account-id>` nel comando con l'ID del tuo Account AWS.

```
aws iam attach-user-policy --user-name IDTGreengrassUser --policy-arn  
arn:aws:iam::<account-id>:policy/IDTGreengrassIAMPermissions
```

- c. Allega la `AWSIoTDeviceTesterForGreengrassFullAccess` policy al tuo utente IAM.

```
aws iam attach-user-policy --user-name IDTGreengrassUser --policy-arn  
arn:aws:iam::aws:policy/AWSIoTDeviceTesterForGreengrassFullAccess
```

 Note

[`AWSIoTDeviceTesterForGreengrassFullAccess`](#) Si tratta di una policy AWS gestita che definisce le autorizzazioni richieste da IDT per creare e accedere alle AWS risorse utilizzate per i test. Per ulteriori informazioni, consulta [the section called “AWS politica gestita per IDT”](#).

4. Creare una chiave di accesso segreta per l'utente.

```
aws iam create-access-key --user-name IDTGreengrassUser
```

Memorizzare l'output in una posizione sicura. Queste informazioni verranno utilizzate successivamente per configurare il file delle AWS credenziali.

5. Passaggio successivo: configurare il [dispositivo fisico](#).

AWS politica gestita per AWS IoT Device Tester

La politica [AWSIoTDeviceTesterForGreengrassFullAccess](#) gestita consente a IDT di eseguire operazioni e raccogliere metriche di utilizzo. La policy concede le seguenti autorizzazioni IDT:

- `iot-device-tester:CheckVersion`. Verifica se un set di versioni AWS IoT Greengrass, una suite di test e IDT sono compatibili.
- `iot-device-tester:DownloadTestSuite`. Scarica le suite di test.
- `iot-device-tester:LatestIdt`. Ottieni informazioni sulla versione IDT più recente disponibile per il download.
- `iot-device-tester:SendMetrics`. Pubblica i dati di utilizzo raccolti da IDT sui tuoi test.
- `iot-device-tester:SupportedVersion`. Ottieni l'elenco AWS IoT Greengrass e prova le versioni della suite supportate da IDT. Queste informazioni vengono visualizzate nella finestra della riga di comando.

Configura il tuo dispositivo per eseguire test IDT

Per configurare il dispositivo è necessario installare AWS IoT Greengrass le dipendenze, configurare il software AWS IoT Greengrass Core, configurare il computer host per accedere al dispositivo e configurare le autorizzazioni utente sul dispositivo.

Verifica AWS IoT Greengrass le dipendenze sul dispositivo sottoposto a test

Prima che IDT for AWS IoT Greengrass possa testare i tuoi dispositivi, assicurati di aver configurato il dispositivo come descritto in [Guida introduttiva](#). AWS IoT Greengrass Per ulteriori informazioni sulle piattaforme supportate, consulta la pagina relativa alle [piattaforme supportate](#).

Configura il software AWS IoT Greengrass

IDT for AWS IoT Greengrass verifica la compatibilità del dispositivo con una versione specifica di AWS IoT Greengrass. IDT offre due opzioni per i test AWS IoT Greengrass sui tuoi dispositivi:

- Scarica e utilizza una versione del [software AWS IoT Greengrass Core](#). IDT installa il software per tuo conto.
- Utilizzate una versione del software AWS IoT Greengrass Core già installata sul dispositivo.

 Note

Ogni versione di AWS IoT Greengrass ha una versione IDT corrispondente. È necessario scaricare la versione di IDT corrispondente alla versione che AWS IoT Greengrass si sta utilizzando.

Le sezioni seguenti descrivono queste opzioni. È sufficiente eseguirne una.

Opzione 1: scarica il software AWS IoT Greengrass Core e configura AWS IoT Device Tester per utilizzarlo

È possibile scaricare il software AWS IoT Greengrass Core dalla pagina dei download del [software AWS IoT Greengrass principale](#).

1. Trova la corretta architettura e distribuzione Linux, quindi scegli Scarica.
2. Copia il file tar.gz in *<device-tester-extract-location>*/products/greengrass/ggc.

 Note

Non modificate il nome del file AWS IoT Greengrass tar.gz. Non posizionare più file in questa directory per lo stesso sistema operativo e architettura. Ad esempio, se i file `greengrass-linux-armv7l-1.7.1.tar.gz` e `greengrass-linux-armv7l-1.8.1.tar.gz` si trovano nella directory, il test non riesce.

Opzione 2: utilizzare un'installazione esistente di AWS IoT Greengrass con AWS IoT Device Tester

Configura IDT per testare il software AWS IoT Greengrass Core installato sul tuo dispositivo aggiungendo l'attributo `greengrassLocation` al file `device.json` nella *<device-tester-extract-location>*/configs cartella. Per esempio:

```
"greengrassLocation" : "<path-to-greengrass-on-device>"
```

Per ulteriori informazioni sul file `device.json`, consulta [Configura dispositivo.json](#).

Sui dispositivi Linux, la posizione predefinita del software AWS IoT Greengrass Core è `greengrass`.

 Note

Il dispositivo dovrebbe avere un'installazione del software AWS IoT Greengrass Core che non è stata avviata.

Assicurati di aver aggiunto l'utente `ggc_user` e `ggc_group` sul dispositivo. Per ulteriori informazioni, consulta [Configurazione dell'ambiente per AWS IoT Greengrass](#).

Configurazione del computer host per l'accesso al dispositivo sottoposto a test

IDT viene eseguito sul computer host e deve essere in grado di utilizzare SSH per connettersi al dispositivo. Sono disponibili due opzioni per consentire a IDT di ottenere l'accesso SSH ai dispositivi sottoposti a test:

1. Segui le istruzioni contenute in questa pagina per creare una coppia di chiavi SSH e autorizzare la chiave ad accedere al dispositivo sottoposto a test senza specificare una password.
2. Fornisci un nome utente e una password per ogni dispositivo nel file `device.json`. Per ulteriori informazioni, consulta [Configura dispositivo.json](#).

Puoi utilizzare qualsiasi implementazione SSL per creare una chiave SSH. Le seguenti istruzioni mostrano come usare [SSH-KEYGEN](#) o [PuTTYgen](#) (per Windows). Se stai utilizzando un'altra implementazione SSL, consulta la documentazione dell'applicazione.

IDT utilizza chiavi SSH per eseguire l'autenticazione con il dispositivo sottoposto a test.

Per creare una chiave SSH con SSH-KEYGEN

1. Crea una chiave SSH.

Puoi utilizzare il comando Open SSH `ssh-keygen` per creare una coppia di chiavi SSH. Se disponi già di una coppia di chiavi SSH sul computer host, una best practice è creare una coppia di chiavi SSH appositamente per IDT. In questo modo, dopo aver completato il test, il computer host non può più connettersi al dispositivo senza immettere una password. Ciò consente inoltre di limitare l'accesso al dispositivo remoto solo a coloro che ne hanno bisogno.

 Note

Windows non dispone di un client SSH installato. Per informazioni sull'installazione di un client SSH in Windows, consulta [Download del software client SSH](#).

Il comando `ssh-keygen` richiede di specificare un nome e un percorso di archiviazione della coppia di chiavi. Per impostazione predefinita, i file della coppia di chiavi sono `id_rsa` (chiave privata) e `id_rsa.pub` (chiave pubblica). In macOS e Linux, il percorso predefinito di questi file è `~/.ssh/`. In Windows, la posizione predefinita è `C:\Users\<user-name>\.ssh`.

Quando richiesto, immetti una frase chiave per proteggere la chiave SSH. Per ulteriori informazioni, consulta l'argomento relativo alla [generazione di una nuova chiave SSH](#).

2. Aggiungi chiavi SSH autorizzate al dispositivo sottoposto a test.

IDT deve utilizzare la chiave privata SSH per accedere al dispositivo sottoposto al test. Per autorizzare le chiavi SSH private per accedere al dispositivo sottoposto a test, utilizza il comando `ssh-copy-id` dal computer host. Questo comando aggiunge la chiave pubblica al file `~/.ssh/authorized_keys` nel dispositivo sottoposto a test. Per esempio:

```
$ ssh-copy-id <remote-ssh-user>@<remote-device-ip>
```

remote-ssh-user Dov'è il nome utente usato per accedere al dispositivo sottoposto a test e *remote-device-ip* l'indirizzo IP del dispositivo sottoposto a test su cui eseguire i test. Per esempio:

```
ssh-copy-id pi@192.168.1.5
```

Quando richiesto, immetti la password per il nome utente specificato nel comando `ssh-copy-id`.

`ssh-copy-id` presuppone che la chiave pubblica sia denominata `id_rsa.pub` e sia archiviata nella posizione predefinita (in macOS e Linux `~/.ssh/` e in Windows `C:\Users\<user-name>\.ssh`). Se hai attribuito un nome diverso alla chiave pubblica o la hai archiviata in una posizione diversa, devi specificare il percorso completo della chiave pubblica SSH con l'opzione `-i` di `ssh-copy-id`, ad esempio `ssh-copy-id -i ~/my/path/myKey.pub`. Per ulteriori informazioni sulla creazione di chiavi SSH e sulla copia di chiavi pubbliche, consulta [SSH-COPY-ID](#).

Per creare una chiave SSH usando PuTTYgen (solo Windows)

1. Assicurati di aver installato il server e il client OpenSSH sul dispositivo sottoposto a test. Per ulteriori informazioni, consulta [OpenSSH](#).
2. Installa [PuTTYgen](#) sul tuo dispositivo in fase di test.
3. Apri PuTTYgen.
4. Scegli Generate (Genera) e sposta il cursore del mouse all'interno della casella per generare una chiave privata.
5. Dal menu Conversions (Conversioni), scegli Export OpenSSH key (Esporta chiave OpenSSH) e salvare la chiave privata con un'estensione di file .pem.
6. Aggiungi la chiave pubblica al file `/home/<user>/.ssh/authorized_keys` sul dispositivo sottoposto a test.
 - a. Copia il testo della chiave pubblica dalla TTYgen finestra Pu.
 - b. Utilizza PuTTY per creare una sessione sul dispositivo sottoposto a test.
 - i. Da un prompt dei comandi o dalla finestra Windows Powershell, esegui il comando seguente:

```
C:/<path-to-putty>/putty.exe -ssh <user>@<dut-ip-address>
```
 - ii. Quando richiesto, immetti la password del dispositivo.
 - iii. Utilizza vi o un altro editor di testo per aggiungere la chiave pubblica al file `/home/<user>/.ssh/authorized_keys` sul dispositivo sottoposto a test.
7. Aggiorna il file `device.json` con il nome utente, l'indirizzo IP e il percorso al file della chiave privata appena salvato sul computer host per ogni dispositivo sottoposto a test. Per ulteriori informazioni, consulta [the section called "Configura dispositivo.json"](#). Assicurati di fornire il percorso completo e il nome di file per la chiave privata e utilizza le barre ("/"). Ad esempio, per il percorso di Windows `C:\DT\privatekey.pem`, utilizza `C:/DT/privatekey.pem` nel file `device.json`.

Configurazione delle autorizzazioni utente sul dispositivo

IDT esegue operazioni su varie directory e file in un dispositivo sottoposto a test. Alcune di queste operazioni richiedono autorizzazioni elevate (utilizzando sudo). Per automatizzare queste operazioni, IDT for AWS IoT Greengrass deve essere in grado di eseguire comandi con sudo senza che venga richiesta una password.

Segui questi passaggi sul dispositivo sottoposto a test per consentire al comando `sudo` di accedere senza che venga richiesta una password.

Note

`username` fa riferimento all'utente SSH utilizzato da IDT per accedere al dispositivo sottoposto a test.

Per aggiungere l'utente al gruppo `sudo`

1. Sul dispositivo sottoposto a test, esegui `sudo usermod -aG sudo <username>`.
2. Per rendere effettive le modifiche, esci ed esegui di nuovo l'accesso.
3. Per verificare che il nome utente sia stato aggiunto correttamente, esegui `sudo echo test`. Se non viene richiesta una password, l'utente è configurato correttamente.
4. Apri il file `/etc/sudoers`, quindi aggiungi la riga seguente alla fine del file:

```
<ssh-username> ALL=(ALL) NOPASSWD: ALL
```

Configurazione del dispositivo per testare le funzionalità opzionali

Negli argomenti seguenti viene descritto come configurare i dispositivi per eseguire test IDT per le funzionalità opzionali. Seguire questi passaggi di configurazione solo se si desidera testare queste funzionalità. Altrimenti, passare a [the section called “Configurare le impostazioni IDT”](#).

Argomenti

- [Opzionale: configurazione del contenitore Docker per IDT per AWS IoT Greengrass](#)
- [Facoltativo: configurazione del dispositivo per la qualificazione di ML](#)

Opzionale: configurazione del contenitore Docker per IDT per AWS IoT Greengrass

AWS IoT Greengrass fornisce un'immagine Docker e un Dockerfile che semplificano l'esecuzione del software AWS IoT Greengrass Core in un contenitore Docker. Dopo aver configurato il AWS IoT Greengrass contenitore, puoi eseguire test IDT. Attualmente, solo le architetture Docker x86_64 sono supportate per l'esecuzione di IDT per AWS IoT Greengrass.

Questa funzionalità richiede IDT v2.3.0 o versione successiva.

Il processo di configurazione del contenitore Docker per l'esecuzione dei test IDT dipende dall'utilizzo dell'immagine Docker o del Dockerfile fornito da AWS IoT Greengrass

- [Usare l'immagine Docker](#). Nell'immagine Docker sono installati il software Core e le dipendenze AWS IoT Greengrass .
- [Usa il Dockerfile](#). Il Dockerfile contiene il codice sorgente che puoi usare per creare immagini di container personalizzate. AWS IoT Greengrass L'immagine può essere modificata per essere eseguita su diverse architetture di piattaforma o per ridurne le dimensioni.

 Note

AWS IoT Greengrass non fornisce file Dockerfile o immagini Docker per AWS IoT Greengrass la versione principale del software 1.11.1. Per eseguire test IDT su immagini di contenitori personalizzate, l'immagine deve includere le dipendenze definite nel Dockerfile fornito da AWS IoT Greengrass

Le seguenti funzionalità non sono disponibili quando vengono eseguite AWS IoT Greengrass in un contenitore Docker:

- [Connettori](#) eseguiti in modalità container Greengrass. Per eseguire un connettore in un container Docker, il connettore deve essere eseguito in modalità Nessun container. Per trovare i connettori che supportano la modalità Nessun container consulta [the section called “AWS-connettori Greengrass forniti”](#). Alcuni di questi connettori dispongono di un parametro della modalità di isolamento che devi impostare su No container (Nessun contenitore).
- [Risorse volume e dispositivo locale](#). Le funzioni Lambda definite dall'utente che vengono eseguite nel contenitore Docker devono accedere direttamente ai dispositivi e ai volumi sul core.

Configura l'immagine Docker fornita da AWS IoT Greengrass

Segui questi passaggi per configurare l'immagine AWS IoT Greengrass Docker per eseguire i test IDT.

Prerequisiti

Prima di iniziare questo tutorial, devi fare quanto segue.

- È necessario installare il software e le versioni seguenti sul computer host in base alla versione AWS Command Line Interface (AWS CLI) scelta.

AWS CLI version 2

- [Docker](#) versione 18.09 o successiva. Anche le versioni precedenti potrebbero funzionare, ma consigliamo 18.09 o successive.
- AWS CLI versione 2.0.0 o successiva.
 - Per installare la AWS CLI versione 2, vedere [Installazione della AWS CLI versione 2](#).
 - Per configurare AWS CLI, vedere [Configurazione di AWS CLI](#).

Note

Per eseguire l'aggiornamento a una AWS CLI versione 2 successiva su un computer Windows, è necessario ripetere la procedura di [installazione MSI](#).

AWS CLI version 1

- [Docker](#) versione 18.09 o successiva. Anche le versioni precedenti potrebbero funzionare, ma consigliamo 18.09 o successive.
- [Python](#) versione 3.6 o successiva.
- [pip](#) versione 18.1 o successiva.
- AWS CLI versione 1.17.10 o successiva
 - Per installare la AWS CLI versione 1, vedi [Installazione della AWS CLI versione 1](#).
 - Per configurare AWS CLI, vedere [Configurazione di AWS CLI](#).
 - Per eseguire l'aggiornamento alla versione più recente della AWS CLI versione 1, esegui il comando seguente.

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

Note

Se utilizzi l'[installazione MSI](#) della AWS CLI versione 1 su Windows, tieni presente quanto segue:

- Se l'installazione della AWS CLI versione 1 non riesce a installare botocore, prova a utilizzare l'installazione di [Python](#) e pip.

- Per eseguire l'aggiornamento a una AWS CLI versione 1 successiva, è necessario ripetere il processo di installazione MSI.
- Per accedere alle risorse di Amazon Elastic Container Registry (Amazon ECR), devi concedere la seguente autorizzazione.
 - Amazon ECR richiede agli utenti di concedere l'`ecr:GetAuthorizationToken` autorizzazione tramite una policy AWS Identity and Access Management (IAM) prima di poter autenticarsi in un registro e inviare o estrarre immagini da un repository Amazon ECR. Per ulteriori informazioni, consulta [Amazon ECR Repository Policy Examples e Accessing One Amazon ECR Repository nella Amazon Elastic Container Registry User Guide](#).
1. Scaricare l'immagine Docker e configurare il contenitore. Puoi scaricare l'immagine predefinita da [Docker Hub](#) o Amazon [Elastic Container Registry](#) (Amazon ECR) ed eseguirla su piattaforme Windows, macOS e Linux (x86_64).

Per scaricare l'immagine Docker da Amazon ECR, completa tutti i passaggi indicati di seguito. [the section called “Ottieni l'immagine del AWS IoT Greengrass contenitore da Amazon ECR”](#)

Quindi, tornare a questo argomento per continuare la configurazione.

2. Solo utenti Linux: assicurarsi che l'utente che esegue IDT disponga dell'autorizzazione per eseguire comandi Docker. Per ulteriori informazioni, vedere [Gestisci finestra mobile come utente non root](#) nella documentazione Docker.
3. Per eseguire il AWS IoT Greengrass contenitore, usa il comando relativo al tuo sistema operativo:

Linux

```
docker run --rm --init -it -d --name aws-iot-greengrass \
-p 8883:8883 \
-v <host-path-to-kernel-config-file>:<container-path> \
<image-repository>:<tag>
```

- Sostituirelo `<host-path-to-kernel-config-file>` con il percorso del file di configurazione del kernel sull'host e `<container-path>` con il percorso in cui il volume è montato nel contenitore.

Il file di configurazione del kernel sull'host di solito si trova in `/proc/config.gz` o `/boot/config-<kernel-release-date>`. Puoi correre `uname -r` per trovare il *<kernel-release-date>* valore.

Esempio: per montare il file di configurazione da `/boot/config-<kernel-release-date>`

```
-v /boot/config-4.15.0-74-generic:/boot/config-4.15.0-74-generic \
```

Esempio: per montare il file di configurazione da `proc/config.gz`

```
-v /proc/config.gz:/proc/config.gz \
```

- Sostituisci *<image-repository>*: *<tag>* nel comando con il nome del repository e il tag dell'immagine di destinazione.

Esempio: indicare la versione più recente del software AWS IoT Greengrass Core

```
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

Per ottenere l'elenco delle immagini AWS IoT Greengrass Docker, esegui il seguente comando.

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --  
repository-name aws-iot-greengrass
```

macOS

```
docker run --rm --init -it -d --name aws-iot-greengrass \  
-p 8883:8883 \  
<image-repository>:<tag>
```

- Sostituisci *<image-repository>*: *<tag>* nel comando con il nome del repository e il tag dell'immagine di destinazione.

Esempio: indicare la versione più recente del software AWS IoT Greengrass Core

```
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

Per ottenere l'elenco delle immagini AWS IoT Greengrass Docker, esegui il seguente comando:

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --  
repository-name aws-iot-greengrass
```

Windows

```
docker run --rm --init -it -d --name aws-iot-greengrass \  
-p 8883:8883 \  
<image-repository>:<tag>
```

- Sostituisci *<image-repository>: <tag>* nel comando con il nome del repository e il tag dell'immagine di destinazione.

Esempio: indicare la versione più recente del software AWS IoT Greengrass Core

```
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

Per ottenere l'elenco delle immagini AWS IoT Greengrass Docker, esegui il seguente comando:

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --  
repository-name aws-iot-greengrass
```

Important

Durante il test con IDT, non includete l'--entrypoint /greengrass-entrypoint.sh \argomento utilizzato per eseguire l'immagine per uso generale AWS IoT Greengrass .

4. Passaggio successivo: [configura le AWS credenziali e device.json](#) il file.

Configura il dockerfile fornito da AWS IoT Greengrass

Segui questi passaggi per configurare l'immagine Docker creata dal AWS IoT Greengrass Dockerfile per eseguire i test IDT.

1. Da [the section called “AWS IoT Greengrass Software Docker”](#), scaricare il pacchetto Dockerfile sul computer host ed estrarlo.
2. Aprire README.md. I tre passaggi successivi si riferiscono alle sezioni di questo file.
3. Assicurarsi di soddisfare i requisiti nella sezione Prerequisiti.
4. Solo utenti Linux: completa i passaggi Abilita la protezione Symlink e Hardlink e Abilita l'inoltro di rete. IPv4
5. Per creare l'immagine Docker, completa tutti i passaggi del Passaggio 1. Crea l'immagine AWS IoT Greengrass Docker. Quindi, tornare a questo argomento per continuare la configurazione.
6. Per eseguire il AWS IoT Greengrass contenitore, usa il comando per il tuo sistema operativo:

Linux

```
docker run --rm --init -it -d --name aws-iot-greengrass \
-p 8883:8883 \
-v <host-path-to-kernel-config-file>:<container-path> \
<image-repository>:<tag>
```

- Sostituirelo *<host-path-to-kernel-config-file>* con il percorso del file di configurazione del kernel sull'host e *<container-path>* con il percorso in cui il volume è montato nel contenitore.

Il file di configurazione del kernel sull'host di solito si trova in `/proc/config.gz` o `/boot/config-<kernel-release-date>`. Puoi correre `uname -r` per trovare il *<kernel-release-date>* valore.

Esempio: per montare il file di configurazione da `/boot/config-<kernel-release-date>`

```
-v /boot/config-4.15.0-74-generic:/boot/config-4.15.0-74-generic \
```

Esempio: per montare il file di configurazione da `proc/config.gz`

```
-v /proc/config.gz:/proc/config.gz \
```

- Sostituisci *<image-repository>*: *<tag>* nel comando con il nome del repository e il tag dell'immagine di destinazione.

Esempio: indicare la versione più recente del software AWS IoT Greengrass Core

```
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

Per ottenere l'elenco delle immagini AWS IoT Greengrass Docker, esegui il seguente comando.

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --  
repository-name aws-iot-greengrass
```

macOS

```
docker run --rm --init -it -d --name aws-iot-greengrass \  
-p 8883:8883 \  
<image-repository>:<tag>
```

- Sostituisci *<image-repository>*: *<tag>* nel comando con il nome del repository e il tag dell'immagine di destinazione.

Esempio: indicare la versione più recente del software AWS IoT Greengrass Core

```
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

Per ottenere l'elenco delle immagini AWS IoT Greengrass Docker, esegui il seguente comando:

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --  
repository-name aws-iot-greengrass
```

Windows

```
docker run --rm --init -it -d --name aws-iot-greengrass \  
-p 8883:8883 \  
<image-repository>:<tag>
```

- Sostituisci *<image-repository>: <tag>* nel comando con il nome del repository e il tag dell'immagine di destinazione.

Esempio: indicare la versione più recente del software AWS IoT Greengrass Core

```
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

Per ottenere l'elenco delle immagini AWS IoT Greengrass Docker, esegui il seguente comando:

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --  
repository-name aws-iot-greengrass
```

Important

Durante il test con IDT, non includete l'--entrypoint /greengrass-entrypoint.sh \argomento utilizzato per eseguire l'immagine per uso generale AWS IoT Greengrass .

7. Passaggio successivo: [configura le AWS credenziali e device.json](#) il file.

Risoluzione dei problemi di configurazione del contenitore Docker per IDT per AWS IoT Greengrass

Utilizza le seguenti informazioni per risolvere i problemi relativi all'esecuzione di un contenitore Docker per IDT a scopo di test. AWS IoT Greengrass

ATTENZIONE: errore durante il caricamento della configurazione file:/home/user/.docker/config.json - stat /home/<user>/.docker/config.json: autorizzazione negata

Se viene visualizzato questo errore durante l'esecuzione di comandi docker su Linux, eseguire il seguente comando. Sostituisci il comando seguente *<user>* con l'utente che esegue IDT.


```
sudo chown <user>:<user> /home/<user>/.docker -R
sudo chmod g+rx /home/<user>/.docker -R
```

Facoltativo: configurazione del dispositivo per la qualificazione di ML

IDT for AWS IoT Greengrass fornisce test di qualificazione dell'apprendimento automatico (ML) per verificare che i tuoi dispositivi siano in grado di eseguire l'inferenza ML localmente utilizzando modelli addestrati sul cloud.

Per eseguire i test di qualificazione di ML, è necessario innanzitutto configurare i dispositivi come descritto in [the section called “Configura il tuo dispositivo per eseguire test IDT”](#). Seguire quindi la procedura descritta in questo argomento per installare le dipendenze per i framework ML che si desidera eseguire.

IDT v3.1.0 o versione successiva è necessario per eseguire i test per la qualificazione di ML.

Installazione delle dipendenze del framework ML

Tutte le dipendenze framework ML devono essere installate nella directory `/usr/local/lib/python3.x/site-packages`. Per assicurarsi che siano installate nella directory corretta, si consiglia di utilizzare le autorizzazioni root `sudo` durante l'installazione delle dipendenze. Gli ambienti virtuali non sono supportati per i test di qualificazione.

Note

Se stai testando funzioni Lambda che vengono eseguite con la [containerizzazione](#) (in modalità contenitore Greengrass), la creazione di collegamenti simbolici per le librerie Python sotto non è supportata. `/usr/local/lib/python3.x` Per evitare errori, è necessario installare le dipendenze nella directory corretta.

Seguire i passaggi per installare le dipendenze per il framework di destinazione:

- [Installa le dipendenze MXNet](#)
- [the section called “Installa TensorFlow le dipendenze”](#)
- [Installare le dipendenze di DLR](#)

MXNet Installa le dipendenze di Apache

I test di qualificazione IDT per questo framework hanno le seguenti dipendenze:

- Python 3.6 o Python 3.7.

Note

Se si sta usando Python 3.6, occorre creare un collegamento simbolico dai dati binari di Python 3.7 a Python 3.6. Questo configura il dispositivo in modo che soddisfi il requisito Python per AWS IoT Greengrass. Per esempio:

```
sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
```

- Apache MXNet v1.2.1 o versione successiva.
- NumPy. La versione deve essere compatibile con la tua MXNet versione.

Installazione MXNet

Segui le istruzioni riportate nella MXNet documentazione per l'[installazione MXNet](#).

Note

Se Python 2.x e Python 3.x sono entrambi installati sul dispositivo, utilizzare Python 3.x nei comandi eseguiti per installare le dipendenze.

Convalida dell'installazione MXNet

Scegliete una delle seguenti opzioni per convalidare l' MXNet installazione.

Opzione 1: usare il protocollo SSH nel dispositivo ed eseguire gli script

1. SSH nel tuo dispositivo.
2. Eseguire gli script seguenti per verificare che le dipendenze siano installate correttamente.

```
sudo python3.7 -c "import mxnet; print(mxnet.__version__)"
```

```
sudo python3.7 -c "import numpy; print(numpy.__version__)"
```

L'output stampa il numero di versione e lo script dovrebbe essere privo di errori.

Opzione 2: eseguire il test di dipendenza IDT

1. Assicurarsi che `device.json` sia configurato per la qualificazione ML. Per ulteriori informazioni, consulta [the section called “Configurare device.json per la qualificazione ML”](#).
2. Eseguire il test delle dipendenze per il framework.

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id mldependencies --test-id mxnet_dependency_check
```

Il riepilogo del test visualizza un risultato PASSED per `mldependencies`.

Installa TensorFlow le dipendenze

I test di qualificazione IDT per questo framework hanno le seguenti dipendenze:

- Python 3.6 o Python 3.7.

Note

Se si sta usando Python 3.6, occorre creare un collegamento simbolico dai dati binari di Python 3.7 a Python 3.6. Questo configura il dispositivo in modo che soddisfi il requisito Python per AWS IoT Greengrass. Per esempio:

```
sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
```

- TensorFlow 1.x.

Installazione TensorFlow

Segui le istruzioni nella TensorFlow documentazione per installare TensorFlow 1.x [con pip](#) o [dal codice sorgente](#).

 Note

Se Python 2.x e Python 3.x sono entrambi installati sul dispositivo, utilizzare Python 3.x nei comandi eseguiti per installare le dipendenze.

Convalida dell'installazione TensorFlow

Scegliete una delle seguenti opzioni per convalidare l' TensorFlow installazione.

Opzione 1: usare il protocollo SSH nel dispositivo ed eseguire uno script

1. SSH nel tuo dispositivo.
2. Esegui lo script seguente per verificare che la dipendenza sia installata correttamente.

```
sudo python3.7 -c "import tensorflow; print(tensorflow.__version__)"
```

L'output stampa il numero di versione e lo script dovrebbe essere privo di errori.

Opzione 2: eseguire il test di dipendenza IDT

1. Assicurarsi che `device.json` sia configurato per la qualificazione ML. Per ulteriori informazioni, consulta [the section called “Configurare device.json per la qualificazione ML”](#).
2. Esegui il test delle dipendenze per il framework.

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id mldependencies --test-id tensorflow_dependency_check
```

Il riepilogo del test visualizza un risultato PASSED per `mldependencies`.

Installa le dipendenze di Amazon SageMaker AI Neo Deep Learning Runtime (DLR)

I test di qualificazione IDT per questo framework hanno le seguenti dipendenze:

- Python 3.6 o Python 3.7.

 Note

Se si sta usando Python 3.6, occorre creare un collegamento simbolico dai dati binari di Python 3.7 a Python 3.6. Questo configura il dispositivo in modo che soddisfi il requisito Python per AWS IoT Greengrass. Per esempio:

```
sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
```

- SageMaker AI Neo DLR.
- numpy.

Dopo aver installato le dipendenze di test DLR, è necessario [compilare il modello](#).

Installazione di DLR

Seguire le istruzioni contenute nella documentazione di DLR per [installare Neo DLR](#).

 Note

Se Python 2.x e Python 3.x sono entrambi installati sul dispositivo, utilizzare Python 3.x nei comandi eseguiti per installare le dipendenze.

Convalida dell'installazione di DLR

Scegliere una delle seguenti opzioni per convalidare l'installazione di DLR.

Opzione 1: usare il protocollo SSH nel dispositivo ed eseguire gli script

1. SSH nel tuo dispositivo.
2. Eseguire gli script seguenti per verificare che le dipendenze siano installate correttamente.

```
sudo python3.7 -c "import dlr; print(dlr.__version__)"
```

```
sudo python3.7 -c "import numpy; print(numpy.__version__)"
```

L'output stampa il numero di versione e lo script dovrebbe essere privo di errori.

Opzione 2: eseguire il test di dipendenza IDT

1. Assicurarsi che `device.json` sia configurato per la qualificazione ML. Per ulteriori informazioni, consulta [the section called “Configurare device.json per la qualificazione ML”](#).
2. Eseguire il test delle dipendenze per il framework.

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id mldependencies --test-id dlr_dependency_check
```

Il riepilogo del test visualizza un risultato PASSED per `mldependencies`.

Compilare il modello DLR

È necessario compilare il modello DLR prima di poterlo utilizzare per i test di qualificazione di ML. Per i passaggi, scegliere una delle seguenti opzioni:

Opzione 1: usa Amazon SageMaker AI per compilare il modello

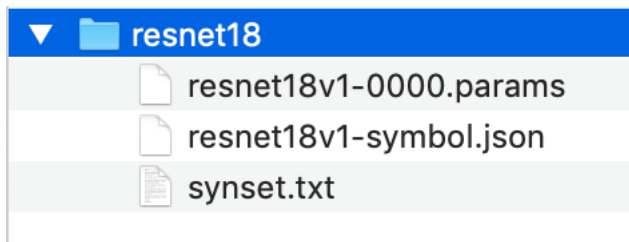
Segui questi passaggi per utilizzare l' SageMaker intelligenza artificiale per compilare il modello ML fornito da IDT. Questo modello è preaddestrato con Apache. MXNet

1. Verifica che il tipo di dispositivo sia supportato dall' SageMaker IA. Per ulteriori informazioni, consulta le [opzioni del dispositivo di destinazione](#) nell'Amazon SageMaker AI API Reference. Se il tuo tipo di dispositivo non è attualmente supportato dall' SageMaker intelligenza artificiale, segui i passaggi indicati in [the section called “Opzione 2: utilizzare TVM per compilare il modello DLR”](#).

Note

L'esecuzione del test DLR con un modello compilato dall' SageMaker IA potrebbe richiedere 4 o 5 minuti. Non arrestare IDT durante questo periodo.

2. Scarica il file tarball che contiene il modello non compilato e preaddestrato MXNet per DLR:
 - [dlr-noncompiled-model-1,0. tar.gz](#)
3. Decomprimere il tarball. Questo comando genera la seguente struttura di directory.

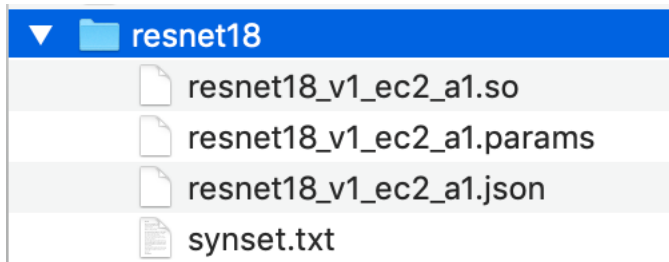


4. Spostare `synset.txt` fuori dalla directory `resnet18`. Prendere nota del nuovo percorso. Copiare questo file nella directory del modello compilato in un secondo momento.
5. Comprimere il contenuto della directory `resnet18`.

```
tar cvfz model.tar.gz resnet18v1-symbol.json resnet18v1-0000.params
```

6. Carica il file compresso in un bucket Amazon S3 nel Account AWS tuo, quindi segui i passaggi [in Compila un modello \(console\) per creare un processo di compilazione](#).
 - a. Per Input configuration (Configurazione di input), utilizzare i seguenti valori:
 - Per Data input configuration (Configurazione di input dati), immettere `{"data": [1, 3, 224, 224]}`.
 - Per Machine learning framework (Framework di machine learning), selezionare MXNet.
 - b. Per Output configuration (Configurazione di output), utilizzare i seguenti valori:
 - Per la posizione di uscita S3, inserisci il percorso del bucket o della cartella Amazon S3 in cui desideri archiviare il modello compilato.
 - Per Target device (Dispositivo di destinazione), scegliere il tipo di dispositivo.
7. Scaricare il modello compilato dal percorso di output specificato, quindi decomprimere il file.
8. Copiare `synset.txt` nella directory del modello compilato.
9. Modificare il nome della directory del modello compilato in `resnet18`.

La directory del modello compilato deve avere la seguente struttura della directory.



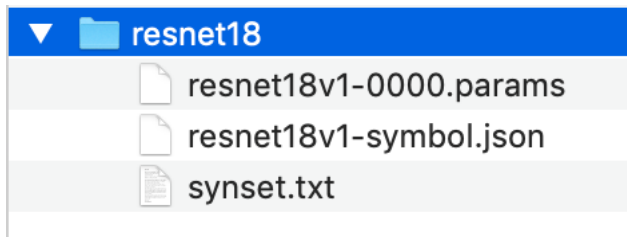
Opzione 2: utilizzare TVM per compilare il modello DLR

Seguire questi passaggi per utilizzare TVM per compilare il modello di ML fornito da IDT. Questo modello è già stato addestrato con Apache MXNet, quindi devi installarlo MXNet sul computer o sul dispositivo su cui compili il modello. [Per l'installazione MXNet, segui le istruzioni nella documentazione. MXNet](#)

Note

Si consiglia di compilare il modello sul dispositivo di destinazione. Questa pratica è facoltativa, ma può contribuire a garantire la compatibilità e a ridurre i potenziali problemi.

1. Scaricate il file tarball che contiene il MXNet modello non compilato e preaddestrato per DLR:
 - [dlr-noncompiled-model-1,0. tar.gz](#)
2. Decomprimere il tarball. Questo comando genera la seguente struttura di directory.



3. Seguire le istruzioni contenute nella documentazione di TVM per [creare e installare TVM dal sorgente per la piattaforma](#).
4. Dopo aver creato TVM, eseguire la compilazione di TVM per il modello resnet18. I seguenti passaggi sono basati sul [Tutorial di avvio rapido per la compilazione di modelli di deep learning](#) contenuto nella documentazione di TVM.
 - a. Aprire il file `relay_quick_start.py` dal repository di TVM clonato.
 - b. Aggiornare il codice che [definisce una rete neurale in relay](#). È possibile utilizzare una delle seguenti opzioni:
 - Opzione 1: utilizzare `mxnet.gluon.model_zoo.vision.get_model` per ottenere il modulo relay e i parametri:

```
from mxnet.gluon.model_zoo.vision import get_model
```



```
block = get_model('resnet18_v1', pretrained=True)
mod, params = relay.frontend.from_mxnet(block, {"data": data_shape})
```

- Opzione 2: dal modello non compilato scaricato al passaggio 1, copiare i seguenti file nella stessa directory del file `relay_quick_start.py`. Questi file contengono il modulo `relay` e i parametri.

- `resnet18v1-symbol.json`
- `resnet18v1-0000.params`

- c. Aggiornare il codice che [salva e carica il modulo compilato](#) per utilizzare il codice seguente.

```
from tvml.contrib import util
path_lib = "deploy_lib.so"
# Export the model library based on your device architecture
lib.export_library("deploy_lib.so", cc="aarch64-linux-gnu-g++")
with open("deploy_graph.json", "w") as fo:
    fo.write(graph)
with open("deploy_param.params", "wb") as fo:
    fo.write(relay.save_param_dict(params))
```

- d. Costruire il modello:

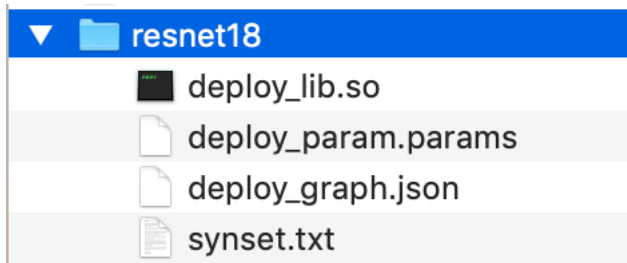
```
python3 tutorials/relay_quick_start.py --build-dir ./model
```

Questo comando genera i seguenti file.

- `deploy_graph.json`
- `deploy_lib.so`
- `deploy_param.params`

5. Copiare i file del modello generati in una directory denominata `resnet18`. Questa è la directory del modello compilato.
6. Copiare la directory del modello compilato nel computer host. Copiare quindi `synset.txt` dal modello non compilato scaricato nel passaggio 1 nella directory del modello compilato.

La directory del modello compilato deve avere la seguente struttura della directory.



Successivamente, [configura](#) le credenziali e il file `AWS device.json`

Configurare le impostazioni IDT per eseguire la suite di AWS IoT Greengrass qualifiche

Prima di eseguire i test, è necessario configurare le impostazioni per le AWS credenziali e i dispositivi sul computer host.

Configura le tue credenziali AWS

È necessario configurare le credenziali utente IAM nel `<device-tester-extract-location>/configs/config.json` file. Utilizza le credenziali per l'IDT per l'AWS IoT Greengrass utente creato in [the section called “Crea e configura un Account AWS”](#) Puoi specificare le credenziali in uno dei due modi seguenti:

- File delle credenziali
- Variabili di ambiente

Configura le AWS credenziali con un file di credenziali

IDT usa lo stesso file delle credenziali di AWS CLI. Per ulteriori informazioni, consulta l'argomento relativo ai [file di configurazione e delle credenziali](#).

La posizione del file delle credenziali varia in base al sistema operativo in uso:

- macOS, Linux: `~/.aws/credentials`
- Windows: `C:\Users\UserName\.aws\credentials`

Aggiungi AWS le tue credenziali al `credentials` file nel seguente formato:

```
[default]
```

```
aws_access_key_id = <your_access_key_id>
aws_secret_access_key = <your_secret_access_key>
```

Per configurare IDT in modo AWS IoT Greengrass che utilizzi AWS le credenziali del tuo `credentials` file, modifica il `config.json` file come segue:

```
{
  "awsRegion": "us-west-2",
  "auth": {
    "method": "file",
    "credentials": {
      "profile": "default"
    }
  }
}
```

Note

Se non usi il default AWS profilo, assicurati di cambiarne il nome nel `fileconfig.json`. Per ulteriori informazioni, consulta l'articolo relativo ai [profili denominati](#).

Configura AWS le credenziali con variabili di ambiente

Le variabili di ambiente sono variabili gestite dal sistema operativo e utilizzate dai comandi di sistema. Non vengono salvate se chiudi la sessione SSH. IDT for AWS IoT Greengrass può utilizzare le variabili di `AWS_SECRET_ACCESS_KEY` ambiente `AWS_ACCESS_KEY_ID` e per memorizzare le credenziali. AWS

Per impostare queste variabili su Linux, macOS o Unix, utilizza `export`:

```
export AWS_ACCESS_KEY_ID=<your_access_key_id>
export AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

Per impostare queste variabili su Windows, utilizza `set`:

```
set AWS_ACCESS_KEY_ID=<your_access_key_id>
set AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

Per configurare IDT per utilizzare le variabili di ambiente, modifica la sezione `auth` nel file `config.json`. Ecco un esempio:

```
{
  "awsRegion": "us-west-2",
  "auth": {
    "method": "environment"
  }
}
```

Configura dispositivo.json

Oltre alle AWS credenziali, IDT for AWS IoT Greengrass necessita di informazioni sui dispositivi su cui vengono eseguiti i test (ad esempio, indirizzo IP, informazioni di accesso, sistema operativo e architettura della CPU).

Devi fornire queste informazioni utilizzando il modello `device.json` situato in `<device_tester_extract_location>/configs/device.json`:

Physical device

```
[
  {
    "id": "<pool-id>",
    "sku": "<sku>",
    "features": [
      {
        "name": "os",
        "value": "linux | ubuntu | openwrt"
      },
      {
        "name": "arch",
        "value": "x86_64 | armv6l | armv7l | aarch64"
      },
      {
        "name": "container",
        "value": "yes | no"
      },
      {
        "name": "docker",
        "value": "yes | no"
      }
    ]
  }
]
```

```

    {
      "name": "streamManagement",
      "value": "yes | no"
    },
    {
      "name": "hsi",
      "value": "yes | no"
    },
    {
      "name": "ml",
      "value": "mxnet | tensorflow | dlr | mxnet,dlr,tensorflow | no"
    },
    ***** Remove the section below if the device is not qualifying for ML
    *****
    {
      "name": "mlLambdaContainerizationMode",
      "value": "container | process | both"
    },
    {
      "name": "processor",
      "value": "cpu | gpu"
    },
    ],
    ***** Remove the section below if the device is not qualifying for HSI
    *****
    "hsm": {
      "p11Provider": "/path/to/pkcs11ProviderLibrary",
      "slotLabel": "<slot_label>",
      "slotUserPin": "<slot_pin>",
      "privateKeyLabel": "<key_label>",
      "opensslEngine": "/path/to/openssl/engine"
    },
    ***** Remove the section below if the device is not qualifying for ML
    *****
    "machineLearning": {
      "dlrModelPath": "/path/to/compiled/dlr/model",
      "environmentVariables": [
        {
          "key": "<environment-variable-name>",
          "value": "<Path:$PATH>"
        }
      ]
    }
  }
}

```

```

    }
  ],
  "deviceResources": [
    {
      "name": "<resource-name>",
      "path": "<resource-path>",
      "type": "device | volume"
    }
  ]
},
*****
"kernelConfigLocation": "",
"greengrassLocation": "",
"devices": [
  {
    "id": "<device-id>",
    "connectivity": {
      "protocol": "ssh",
      "ip": "<ip-address>",
      "port": 22,
      "auth": {
        "method": "pki | password",
        "credentials": {
          "user": "<user-name>",
          "privKeyPath": "/path/to/private/key",
          "password": "<password>"
        }
      }
    }
  }
]
}
]

```

Note

Specificare `privKeyPath` solo se `method` è impostato su `pki`.
Specificare `password` solo se `method` è impostato su `password`.

Docker container

```
[
  {
    "id": "<pool-id>",
    "sku": "<sku>",
    "features": [
      {
        "name": "os",
        "value": "linux | ubuntu | openwrt"
      },
      {
        "name": "arch",
        "value": "x86_64"
      },
      {
        "name": "container",
        "value": "no"
      },
      {
        "name": "docker",
        "value": "no"
      },
      {
        "name": "streamManagement",
        "value": "yes | no"
      },
      {
        "name": "hsi",
        "value": "no"
      },
      {
        "name": "ml",
        "value": "mxnet | tensorflow | dlr | mxnet,dlr,tensorflow | no"
      },
      ***** Remove the section below if the device is not qualifying for ML
      *****
      {
        "name": "mlLambdaContainerizationMode",
        "value": "process"
      },
      {
        "name": "processor",
        "value": "cpu | gpu"
      }
    ]
  }
]
```

```

    },

*****

    ],
    ***** Remove the section below if the device is not qualifying for ML
*****
    "machineLearning": {
        "dlrModelPath": "/path/to/compiled/dlr/model",
        "environmentVariables": [
            {
                "key": "<environment-variable-name>",
                "value": "<Path:$PATH>"
            }
        ],
        "deviceResources": [
            {
                "name": "<resource-name>",
                "path": "<resource-path>",
                "type": "device | volume"
            }
        ]
    },

*****

    "kernelConfigLocation": "",
    "greengrassLocation": "",
    "devices": [
        {
            "id": "<device-id>",
            "connectivity": {
                "protocol": "docker",
                "containerId": "<container-name | container-id>",
                "containerUser": "<user>"
            }
        }
    ]
}
]

```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

id

Un ID alfanumerico definito dall'utente che identifica in modo univoco una raccolta di dispositivi denominata un pool di dispositivi. I dispositivi che appartengono a un pool devono avere lo stesso hardware. Durante l'esecuzione di una suite di test, i dispositivi del pool vengono utilizzati per parallelizzare il carico di lavoro. Più dispositivi vengono utilizzati per eseguire diversi test.

sku

Un valore alfanumerico che identifica in modo univoco il dispositivo sottoposto a test. Il codice SKU viene utilizzato per tenere traccia delle schede qualificate.

Note

Se desideri inserire la tua scheda nel AWS Partner Device Catalog, lo SKU che specifichi qui deve corrispondere allo SKU che utilizzi nel processo di pubblicazione.

features

Un array contenente le caratteristiche supportate del dispositivo. Tutte le funzionalità sono obbligatorie.

os e arch

Combinazioni di sistemi operativi (OS) e architetture supportate:

- `linux, x86_64`
- `linux, armv6l`
- `linux, armv7l`
- `linux, aarch64`
- `ubuntu, x86_64`
- `openwrt, armv7l`
- `openwrt, aarch64`

Note

Se utilizzi IDT per testare l' AWS IoT Greengrass esecuzione in un contenitore Docker, è supportata solo l'architettura Docker x86_64.

container

Verifica che il dispositivo soddisfi tutti i requisiti software e hardware per eseguire le funzioni Lambda in modalità contenitore su un core Greengrass.

Il valore valido è o. yes no

docker

Verifica che il dispositivo soddisfi tutte le dipendenze tecniche richieste per utilizzare il connettore di distribuzione delle applicazioni Greengrass Docker per eseguire i contenitori

Il valore valido è o. yes no

streamManagement

Verifica che il dispositivo soddisfi tutte le dipendenze tecniche richieste per eseguire AWS IoT Greengrass lo stream manager.

Il valore valido è yes o. no

hsi

Verifica che la libreria condivisa HSI fornita sia in grado di interfacciarsi con il modulo di sicurezza hardware (HSM) e implementa correttamente il PKCS #11 richiesto. APIs La HSM e la libreria condivisa devono essere in grado di accedere a un CSR, eseguire operazioni TLS e fornire le lunghezze di chiave e l'algoritmo chiave pubblica corretti.

Il valore valido è o. yes no

ml

Verifica che il dispositivo soddisfi tutte le dipendenze tecniche richieste per eseguire localmente l'inferenza di ML.

Il valore valido può essere qualsiasi combinazione di mxnet, tensorflow, dlr, e no (ad esempiomxnet,mxnet,tensorflow,mxnet,tensorflow,dlr,ono).

mlLambdaContainerizationMode

Verifica che il dispositivo soddisfi tutte le dipendenze tecniche richieste per eseguire l'inferenza ML in modalità contenitore su un dispositivo Greengrass.

Il valore valido ècontainer, o. process both

processor

Verifica che il dispositivo soddisfi tutti i requisiti hardware per il tipo di processore specificato.

Il valore valido è `cpu` o `gpu`.

Note

Se non desideri utilizzare la funzionalità `container`, `docker`, `streamManager` o `hsm`, puoi impostare la `ml` funzione corrispondente `value` a `no`.

Docker supporta solo la qualificazione delle funzionalità per `streamManagement` e `ml`.

machineLearning

Facoltativo. Informazioni di configurazione per i test di qualificazione ML. Per ulteriori informazioni, consulta [the section called “Configurare device.json per la qualificazione ML”](#).

hsm

Facoltativo. Informazioni di configurazione per i test con un AWS IoT Greengrass Hardware Security Module (HSM). Altrimenti, la proprietà `hsm` dovrebbe essere omessa. Per ulteriori informazioni, consulta [Integrazione della sicurezza hardware](#).

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `ssh`.

`hsm.p11Provider`

Il percorso assoluto della libreria `libdl-loadable` dell'implementazione PKCS#11.

`hsm.slotLabel`

L'etichetta dello slot utilizzata per identificare il modulo hardware.

`hsm.slotUserPin`

Il PIN utente utilizzato per autenticare il AWS IoT Greengrass core del modulo.

`hsm.privateKeyLabel`

L'etichetta utilizzata per identificare la chiave nel modulo hardware.

`hsm.openSSLEngine`

Il percorso assoluto del file `.so` del motore OpenSSL che consente di abilitare il supporto PKCS#11 su OpenSSL. Utilizzato dall'agente di aggiornamento AWS IoT Greengrass OTA.

`devices.id`

Un identificativo univoco definito dall'utente del dispositivo sottoposto a test.

`connectivity.protocol`

Il protocollo di comunicazione utilizzato per comunicare con questo dispositivo. Attualmente, gli unici valori supportati sono `ssh` per i dispositivi fisici e `docker` per i contenitori Docker.

`connectivity.ip`

L'indirizzo IP del dispositivo sottoposto a test.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `ssh`.

`connectivity.containerId`

L'ID contenitore o il nome del contenitore Docker in fase di test.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `docker`.

`connectivity.auth`

Informazioni di autenticazione per la connessione.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `ssh`.

`connectivity.auth.method`

Il metodo di autorizzazione utilizzato per accedere a un dispositivo con un determinato protocollo di connettività.

I valori supportati sono:

- `pki`
- `password`

`connectivity.auth.credentials`

Le credenziali utilizzate per l'autenticazione.

`connectivity.auth.credentials.password`

La password utilizzata per l'accesso al dispositivo da testare.

Questo valore si applica solo se `connectivity.auth.method` è impostato su `password`.

`connectivity.auth.credentials.privKeyPath`

Il percorso completo alla chiave privata utilizzata per accedere al dispositivo sottoposto a test.

Questo valore si applica solo se `connectivity.auth.method` è impostato su `pki`.

`connectivity.auth.credentials.user`

Il nome utente per l'accesso al dispositivo sottoposto a test.

`connectivity.auth.credentials.privKeyPath`

Il percorso completo della chiave privata utilizzata per accedere al dispositivo sottoposto a test.

`connectivity.port`

Facoltativo. Il numero di porta da utilizzare per le connessioni SSH.

Il valore predefinito è 22.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `sussh`.

`greengrassLocation`

La posizione del software AWS IoT Greengrass Core sui tuoi dispositivi.

Per i dispositivi fisici, questo valore viene utilizzato solo quando si utilizza un'installazione esistente di AWS IoT Greengrass. Utilizza questo attributo per comunicare a IDT di utilizzare la versione del software AWS IoT Greengrass Core installata sui dispositivi.

Quando esegui test in un contenitore Docker dall'immagine Docker o dal Dockerfile fornito da AWS IoT Greengrass, imposta questo valore su `/greengrass`

`kernelConfigLocation`

Facoltativo. Il percorso del file di configurazione del kernel. AWS IoT Device Tester utilizza questo file per verificare se i dispositivi hanno le funzionalità del kernel richieste abilitate. Se non specificato, IDT utilizza i seguenti percorsi per cercare il file di configurazione del kernel: `and`.
`/proc/config.gz` `/boot/config-<kernel-version>` AWS IoT Device Tester utilizza il primo percorso che trova.

Configurare device.json per la qualificazione ML

In questa sezione vengono descritte le proprietà facoltative nel file di configurazione del dispositivo che si applicano alla qualificazione ML. Se si prevede di eseguire test per la qualificazione ML, è necessario definire le proprietà applicabili al caso d'uso.

È possibile utilizzare il modello `device-ml.json` per definire le impostazioni di configurazione per il dispositivo. Questo modello contiene le proprietà ML facoltative. È inoltre possibile utilizzare `device.json` e aggiungere le proprietà di qualificazione ML. Questi file si trovano in `<device-tester-extract-location>/configs` e comprendono le proprietà di qualificazione ML. Se si utilizza `device-ml.json`, è necessario rinominare il file in `device.json` prima di eseguire test IDT.

Per informazioni sulle proprietà di configurazione del dispositivo che non si applicano alla qualificazione ML, vedere [the section called “Configura dispositivo.json”](#).

ml nell'array features

I framework ML supportati dalla tua scheda. Questa proprietà richiede IDT v3.1.0 o versione successiva.

- Se la scheda supporta un solo framework, specificare il framework. Per esempio:

```
{
  "name": "ml",
  "value": "mxnet"
}
```

- Se la scheda supporta più framework, specificare i framework come elenco separato da virgole. Per esempio:

```
{
  "name": "ml",
  "value": "mxnet,tensorflow"
}
```

mlLambdaContainerizationMode nell'array features

La [modalità di containerizzazione](#) con cui si desidera eseguire il test. Questa proprietà richiede IDT v3.1.0 o versione successiva.

- Scegli `process` di eseguire il codice di inferenza ML con una funzione Lambda non containerizzata. Questa opzione richiede la versione 1.10.x o successiva. AWS IoT Greengrass
- Scegli `container` di eseguire il codice di inferenza ML con una funzione Lambda containerizzata.
- Scegliere `both` per eseguire il codice di inferenza di ML con entrambe le modalità. Questa opzione richiede AWS IoT Greengrass la versione 1.10.x o successiva.

`processor` nell'array `features`

Indica l'acceleratore hardware supportato dalla scheda. Questa proprietà richiede IDT v3.1.0 o versione successiva.

- Scegliere `cpu` se la tua scheda utilizza una CPU come processore.
- Scegliere `gpu` se la scheda utilizza una GPU come processore.

`machineLearning`

Facoltativo. Informazioni di configurazione per i test di qualificazione ML. Questa proprietà richiede IDT v3.1.0 o versione successiva.

`dlrModelPath`

Necessario per utilizzare il framework `dlr`. Il percorso assoluto della directory del modello compilato DLR, che deve essere denominato `resnet18`. Per ulteriori informazioni, consulta [the section called “Compilare il modello DLR”](#).

Note

Di seguito è riportato un esempio di percorso su macOS: `/Users/<user>/Downloads/resnet18`.

`environmentVariables`

Un array di coppie chiave-valore che possono passare dinamicamente le impostazioni ai test di inferenza di ML. Facoltativo per i dispositivi CPU. È possibile utilizzare questa sezione per aggiungere variabili di ambiente specifiche del framework richieste dal tipo di dispositivo. Per informazioni su questi requisiti, vedere il sito Web ufficiale del framework o del dispositivo. Ad esempio, per eseguire test di MXNet inferenza su alcuni dispositivi, potrebbero essere necessarie le seguenti variabili di ambiente.

```
"environmentVariables": [  
  ...  
  {  
    "key": "PYTHONPATH",  
    "value": "$MXNET_HOME/python:$PYTHONPATH"  
  },  
  {  
    "key": "MXNET_HOME",  
    "value": "$HOME/mxnet/"  
  },  
  ...  
]
```

 Note

Il `value` campo potrebbe variare in base all' MXNet installazione.

Se stai testando funzioni Lambda eseguite con [containerizzazione](#) su dispositivi GPU, aggiungi le variabili di ambiente per la libreria GPU. Ciò consente alla GPU di eseguire calcoli. Per utilizzare librerie GPU diverse, vedere la documentazione ufficiale della libreria o del dispositivo.

 Note

Configurare le seguenti chiavi se la funzione `m1LambdaContainerizationMode` è impostata su `container` o `both`.

```
"environmentVariables": [  
  {  
    "key": "PATH",  
    "value": "<path/to/software/bin>:$PATH"  
  },  
  {  
    "key": "LD_LIBRARY_PATH",  
    "value": "<path/to/ld/lib>"  
  },  
  ...  
]
```


deviceResources

Richiesto dai dispositivi GPU. Contiene [risorse locali](#) a cui è possibile accedere tramite le funzioni Lambda. Utilizzare questa sezione per aggiungere risorse locali per dispositivi e volumi.

- Specificare "type": "device" per le risorse del dispositivo. Per i dispositivi GPU, le risorse del dispositivo devono essere file di dispositivo relativi alla GPU sotto /dev.

Note

La directory /dev/shm è un'eccezione. Può essere configurato esclusivamente come risorsa di volume.

- Specificare "type": "volume" per le risorse di volume.

Esegui la suite AWS IoT Greengrass di qualificazione

Dopo avere [impostato la configurazione richiesta](#), puoi iniziare i test. Il runtime delle suite di test completa dipende dall'hardware. Per riferimento, per completare la suite di test completa su un Raspberry Pi 3B sono necessari circa 30 minuti.

Nei comandi run-suite di esempio riportati di seguito viene illustrato come eseguire i test di qualifica per un pool di dispositivi. Un pool di dispositivi è un insieme di dispositivi identici.

IDT v3.0.0 and later

Eseguire tutti i gruppi di test in una suite di test specificata.

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1.0.0 --pool-id <pool-id>
```

Utilizzare il comando list-suites per elencare le suite di test presenti nella cartella tests.

Eseguire un gruppo di test specifico in una suite di test.

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1.0.0 --group-id <group-id> --pool-id <pool-id>
```

Utilizzare il comando list-groups per elencare i gruppi di test in una suite di test.

Eseguire un test case specifico in un gruppo di test.

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id <group-id> --test-id <test-id>
```

Eseguire più casi di test in un gruppo di test.

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id <group-id> --test-id <test-id1>,<test-id2>
```

Elencare i casi di test in un gruppo di test.

```
devicetester_[linux | mac | win_x86-64] list-test-cases --group-id <group-id>
```

Le opzioni per il comando `run-suite` sono facoltative. Ad esempio, è possibile omettere `pool-id` se si dispone di un solo pool di dispositivi definito nel file `device.json`. In alternativa, è possibile omettere `suite-id` se si desidera eseguire l'ultima versione della suite di test nella cartella `tests`.

Note

IDT chiede se è disponibile online una versione più recente della suite di test. Per ulteriori informazioni, consulta [the section called “Imposta il comportamento di aggiornamento predefinito”](#).

Per ulteriori informazioni su `run-suite` e altri comandi IDT, consulta [the section called “Comandi IDT”](#).

IDT v2.3.0 and earlier

Eseguire tutti i gruppi di test in una suite specificata.

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1 --pool-id <pool-id>
```

Eseguire un gruppo di test specifico.

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1 --group-id <group-id> --pool-id <pool-id>
```

`suite-id` e `pool-id` sono facoltativi se si esegue una singola suite di test in un pool di dispositivi singolo. Ciò significa che nel file `device.json` è stato definito solo un pool di dispositivi.

Verifica le dipendenze di Greengrass

Si consiglia di eseguire il gruppo di test dello strumento di controllo delle dipendenze per verificare che tutte le dipendenze di Greengrass siano installate prima di eseguire gruppi di test correlati. Per esempio:

- Eseguire `ggcdependencies` prima dei gruppi di test di qualifica del core.
- Eseguire `containerdependencies` prima dei gruppi di test specifici del contenitore.
- Eseguire `dockerdependencies` prima dei gruppi di test specifici del Docker.
- Eseguire `ggcstreammanagementdependencies` prima dei gruppi di test specifici di Stream Manager.

Imposta il comportamento di aggiornamento predefinito

Quando si avvia un'esecuzione di test, IDT verifica online una versione più recente della suite di test. Se una è disponibile, IDT richiede di eseguire l'aggiornamento alla versione più recente disponibile. È possibile impostare il flag `upgrade-test-suite` (o `u`) per controllare il comportamento di aggiornamento predefinito. I valori validi sono:

- `y`. IDT scarica e utilizza l'ultima versione disponibile.
- `n` (impostazione predefinita). IDT utilizza la versione specificata nell'opzione `suite-id`. Se non `suite-id` è specificato, IDT utilizza la versione più recente nella `tests` cartella.

Se non si include il flag `upgrade-test-suite`, IDT richiede quando è disponibile un aggiornamento e attende 30 secondi per l'input (`y` o `n`). Se non viene inserito alcun input, viene impostato in modo predefinito su `n` e continua a eseguire i test.

Gli esempi seguenti mostrano casi d'uso comuni per questa funzionalità:

Utilizzare automaticamente i test più recenti disponibili per un gruppo di test.

```
devicetester_linux run-suite -u y --group-id mqtt --pool-id DevicePool1
```

Eseguire test in una versione specifica della suite di test.

```
devicetester_linux run-suite -u n --suite-id GGQ_1.0.0 --group-id mqtt --pool-id DevicePool1
```

Richiedere aggiornamenti al runtime.

```
devicetester_linux run-suite --pool-id DevicePool1
```

IDT per i comandi AWS IoT Greengrass

I comandi IDT si trovano nella directory *<device-tester-extract-location>/bin*. Utilizzarli per le seguenti operazioni:

IDT v3.0.0 and later

help

Elenca le informazioni sul comando specificato.

list-groups

Elenca i gruppi in una determinata suite di test.

list-suites

Elenca le suite di test disponibili.

list-supported-products

Elenca i prodotti supportati, in questo caso AWS IoT Greengrass le versioni, e le versioni della suite di test per la versione IDT corrente.

list-test-cases

Elenca i casi di test in un determinato gruppo di test. È supportata la seguente opzione:

- **group-id**. Il gruppo di test da cercare. Questa opzione è obbligatoria e deve specificare un singolo gruppo.

run-suite

Esegue una suite di test in un determinato pool di dispositivi. Di seguito sono riportate alcune opzioni supportate:

- `suite-id`. La versione della suite di test da eseguire. Se non specificato, IDT utilizza la versione più recente nella cartella `tests`.
- `group-id`. I gruppi di test da eseguire, sotto forma di elenco separato da virgole. Se non specificato, IDT esegue tutti i gruppi di test nella suite di test.
- `test-id`. I casi di test da eseguire, come elenco separato da virgole. Quando specificato, `group-id` deve specificare un singolo gruppo.
- `pool-id`. Il pool di dispositivi da testare. È necessario specificare un pool se nel file `device.json` sono stati definiti più pool di dispositivi.
- `upgrade-test-suite`. Controlla come vengono gestiti gli aggiornamenti delle versioni della suite di test. A partire da IDT v3.0.0, IDT controlla online le versioni aggiornate della suite di test. Per ulteriori informazioni, consulta [the section called “Versioni della suite di test”](#).
- `stop-on-first-failure`. Configura IDT per interrompere l'esecuzione al primo errore. Questa opzione deve essere utilizzata con `group-id` per eseguire il debug dei gruppi di test specificati. Non utilizzare questa opzione quando si esegue una suite di test completa per generare un rapporto di qualifica.
- `update-idt`. Imposta la risposta alla richiesta di aggiornamento di IDT. `Y` as input interrompe l'esecuzione del test se IDT rileva l'esistenza di una versione più recente. `N` come input continua l'esecuzione del test.
- `update-managed-policy`. `Y` come input interrompe l'esecuzione del test se IDT rileva che la policy gestita dell'utente non è aggiornata. `N` come input continua l'esecuzione del test.

Per ulteriori informazioni sulle opzioni `run-suite`, utilizzare l'opzione `help`:

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

IDT v2.3.0 and earlier

help

Elenca le informazioni sul comando specificato.

list-groups

Elenca i gruppi in una determinata suite di test.

list-suites

Elenca le suite di test disponibili.

run-suite

Esegue una suite di test in un determinato pool di dispositivi.

Per ulteriori informazioni sulle opzioni run-suite, utilizzare l'opzione help:

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

Informazioni su risultati e log

Questa sezione descrive come visualizzare e interpretare i log e i report dei risultati di IDT.

Visualizzazione dei risultati

Durante l'esecuzione, IDT scrive gli errori nella console, i file di log e i report di test. Al termine della suite di test di qualifica, IDT genera due report di test. Questi report sono disponibili in *<device-tester-extract-location>/results/<execution-id>/*. Entrambi i report acquisiscono i risultati dall'esecuzione della suite di test di qualifica.

`awsiotdevicetester_report.xml` È il rapporto del test di qualificazione che invii AWS per inserire il tuo AWS Partner dispositivo nel Catalogo dispositivi. Il report contiene i seguenti elementi:

- La versione di IDT.
- La AWS IoT Greengrass versione che è stata testata.
- Il codice SKU e il nome del pool di dispositivi specificato nel file `device.json`.
- Le caratteristiche del pool di dispositivi specificato nel file `device.json`.
- Il riepilogo aggregato dei risultati dei test.
- Un'analisi dei risultati dei test da parte delle librerie sottoposte a test in base alle caratteristiche del dispositivo (ad esempio accesso alle risorse locali, shadow, MQTT e così via).

Il `GGQ_Result.xml` rapporto è in [formato JUnit XML](#). Puoi eseguire l'integrazione in piattaforme di integrazione e distribuzione continue come [Jenkins](#), [Bambù](#) e così via. Il report contiene i seguenti elementi:

- Riepilogo aggregato dei risultati dei test.
- Suddivisione dei risultati del test in base alla AWS IoT Greengrass funzionalità testata.

Interpretazione dei report IDT

La sezione dei report in `awsiotdevicetester_report.xml` o `awsiotdevicetester_report.xml` elenca i test eseguiti e i risultati.

Il primo tag XML `<testsuites>` contiene il riepilogo dell'esecuzione dei test. Per esempio:

```
<testsuites name="GGQ results" time="2299" tests="28" failures="0" errors="0" disabled="0">
```

Attributi utilizzati nel tag `<testsuites>`

`name`

Il nome della suite di test.

`time`

Il tempo, espresso in secondi, impiegato per eseguire la suite di qualifica.

`tests`

Il numero di test eseguiti.

`failures`

Il numero di test eseguiti ma non superati.

`errors`

Il numero di test che IDT non è stato in grado di eseguire.

`disabled`

Questo attributo non è utilizzato e si può ignorare.

Il file `awsiotdevicetester_report.xml` contiene un tag `<awsproduct>` con le informazioni relative al prodotto sottoposto a test e le caratteristiche del prodotto che sono state convalidate dopo l'esecuzione di una suite di test.

Attributi utilizzati nel tag `<awsproduct>`

name

Il nome del prodotto sottoposto a test.

version

La versione del prodotto sottoposto a test.

features

Le caratteristiche convalidate. Le caratteristiche contrassegnate come `required` sono necessarie per inviare la scheda per la qualifica. Il seguente frammento di codice mostra come questa informazione viene visualizzata nel file `awsiotdevicetester_report.xml`.

```
<feature name="aws-iot-greengrass-no-container" value="supported" type="required"></feature>
```

Caratteristiche contrassegnate come `optional` non sono necessarie per la qualifica. I seguenti snippet mostrano caratteristiche facoltative.

```
<feature name="aws-iot-greengrass-container" value="supported" type="optional"></feature>

<feature name="aws-iot-greengrass-hsi" value="not-supported" type="optional"></feature>
```

Se non si riscontrano errori nei test o nelle funzionalità richieste, il dispositivo soddisfa i requisiti tecnici per funzionare AWS IoT Greengrass e può interagire con i servizi. AWS IoT Se desideri inserire il tuo dispositivo nel Catalogo dei AWS Partner dispositivi, puoi utilizzare questo rapporto come prova di idoneità.

In caso di esiti negativi o errori nei test, puoi identificare il test non riuscito esaminando i tag XML `<testsuites>`. I tag XML `<testsuite>` all'interno del tag `<testsuites>` mostrano il riepilogo dei risultati dei test per un gruppo di test. Per esempio:


```
<testsuite name="combination" package="" tests="1" failures="0" time="161" disabled="0"
errors="0" skipped="0">
```

Il formato è simile al tag `<testsuites>`, ma con un attributo `skipped` che non viene utilizzato e che è possibile ignorare. All'interno di ogni tag XML `<testsuite>` ci sono tag `<testcase>` per ciascuno dei test eseguiti per un gruppo di test. Per esempio:

```
<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>
```

Attributi utilizzati nel tag `<testcase>`

`name`

Il nome del test.

`attempts`

Il numero di volte che IDT ha eseguito il test.

Quando un test non riesce o si verifica un errore, i tag `<failure>` o `<error>` vengono aggiunti al tag `<testcase>` con informazioni per la risoluzione dei problemi. Per esempio:

```
<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>
```

Visualizzazione dei registri

IDT genera log dall'esecuzione di test in `<devicetester-extract-location>/results/<execution-id>/logs`. Vengono generate due serie di log:

`test_manager.log`

Registri generati dal componente Test Manager di AWS IoT Device Tester (ad esempio, registri relativi alla configurazione, al sequenziamento dei test e alla generazione di report).

```
<test_case_id>.log (for example, ota.log)
```

I log del gruppo di test, inclusi i log del dispositivo sottoposto a test. Quando un test ha esito negativo, viene creato un file tar.gz contenente i log del dispositivo sottoposto a test per il test creato (ad esempio ota_prod_test_1_ggc_logs.tar.gz).

Per ulteriori informazioni, consulta [IDT per la risoluzione dei problemi AWS IoT Greengrass](#).

Usa IDT per sviluppare ed eseguire le tue suite di test

A partire da IDT v4.0.0, IDT for AWS IoT Greengrass combina una configurazione di configurazione e un formato di risultati standardizzati con un ambiente di suite di test che consente di sviluppare suite di test personalizzate per i dispositivi e il software del dispositivo. Potete aggiungere test personalizzati per la vostra convalida interna o fornirli ai clienti per la verifica dei dispositivi.

Utilizza IDT per sviluppare ed eseguire suite di test personalizzate, come segue:

Per sviluppare suite di test personalizzate

- Crea suite di test con logica di test personalizzata per il dispositivo Greengrass che desideri testare.
- Fornisci a IDT le tue suite di test personalizzate per i test runner. Includi informazioni sulle configurazioni di impostazioni specifiche per le tue suite di test.

Per eseguire suite di test personalizzate

- Configura il dispositivo che desideri testare.
- Implementa le configurazioni delle impostazioni come richiesto dalle suite di test che desideri utilizzare.
- Usa IDT per eseguire le tue suite di test personalizzate.
- Visualizza i risultati dei test e i registri di esecuzione per i test eseguiti da IDT.

Scarica l'ultima versione di AWS IoT Device Tester per AWS IoT Greengrass

Scarica l'[ultima versione](#) di IDT ed estrai il software in una posizione del file system in cui disponi delle autorizzazioni di lettura e scrittura.

 Note

IDT non supporta l'esecuzione da parte di più utenti da un percorso condiviso, ad esempio una directory NFS o una cartella condivisa di rete Windows. Si consiglia di estrarre il pacchetto IDT in un'unità locale ed eseguire il file binario IDT sulla workstation locale. In Windows esiste un limite di lunghezza del percorso di 260 caratteri. Se stai usando Windows, estrai IDT in una directory root come C:\ o D:\ per mantenere i percorsi entro il limite di 260 caratteri.

Flusso di lavoro per la creazione della suite

Le suite di test sono composte da tre tipi di file:

- File di configurazione JSON che forniscono a IDT informazioni su come eseguire la suite di test.
- Esegui il test dei file eseguibili utilizzati da IDT per eseguire i test case.
- File aggiuntivi necessari per eseguire i test.

Completa i seguenti passaggi di base per creare test IDT personalizzati:

1. [Crea file di configurazione JSON](#) per la tua suite di test.
2. [Crea eseguibili di test case](#) che contengano la logica di test per la tua suite di test.
3. Verifica e documenta le [informazioni di configurazione richieste ai test runner per](#) eseguire la suite di test.
4. Verifica che IDT sia in grado di eseguire la tua suite di test e produrre [i risultati dei test come previsto](#).

Per creare rapidamente una suite personalizzata di esempio ed eseguirla, segui le istruzioni riportate in [Tutorial: crea ed esegui la suite di test IDT di esempio](#).

Per iniziare a creare una suite di test personalizzata in Python, vedi. [Tutorial: Sviluppa una semplice suite di test IDT](#)

Tutorial: crea ed esegui la suite di test IDT di esempio

Il download di AWS IoT Device Tester include il codice sorgente per una suite di test di esempio. Puoi completare questo tutorial per creare ed eseguire la suite di test di esempio e capire come utilizzare AWS IoT Device Tester per AWS IoT Greengrass eseguire suite di test personalizzate.

In questo tutorial, completerai i seguenti passaggi:

1. [Crea la suite di test di esempio](#)
2. [Usa IDT per eseguire la suite di test di esempio](#)

Prerequisiti

Per completare questo tutorial, è necessario quanto segue:

- Requisiti del computer host
 - Versione più recente di AWS IoT Device Tester
 - [Python 3.7](#) o successivo

Per verificare la versione di Python installata sul tuo computer, esegui il seguente comando:

```
python3 --version
```

In Windows, se l'utilizzo di questo comando restituisce un errore, usalo `python --version` invece. Se il numero di versione restituito è 3.7 o superiore, esegui il comando seguente in un terminale Powershell da impostare `python3` come alias per il comando. `python`

```
Set-Alias -Name "python3" -Value "python"
```

Se non viene restituita alcuna informazione sulla versione o se il numero di versione è inferiore a 3.7, segui le istruzioni in [Downloading Python per installare Python 3.7+](#). Per ulteriori informazioni, consulta la documentazione di [Python](#).

- [urllib3](#)

Per verificare che `urllib3` sia installato correttamente, esegui il seguente comando:

```
python3 -c 'import urllib3'
```

Se non `urllib3` è installato, esegui il seguente comando per installarlo:

```
python3 -m pip install urllib3
```

- Requisiti per il dispositivo
- Un dispositivo con un sistema operativo Linux e una connessione di rete alla stessa rete del computer host.

Ti consigliamo di utilizzare un [Raspberry Pi](#) con sistema operativo Raspberry Pi. Assicurati di aver configurato [SSH](#) sul tuo Raspberry Pi per connetterti in remoto ad esso.

Configura le informazioni sul dispositivo per IDT

Configura le informazioni sul dispositivo per consentire a IDT di eseguire il test. È necessario aggiornare il `device.json` modello che si trova nella `<device-tester-extract-location>/configs` cartella con le seguenti informazioni.

```
[
  {
    "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>"
            }
          }
        }
      }
    ]
  }
]
```

```
]
```

Nell'`devicesoggetto`, fornite le seguenti informazioni:

`id`

Un identificatore univoco definito dall'utente per il dispositivo.

`connectivity.ip`

L'indirizzo IP del dispositivo.

`connectivity.port`

Facoltativo. Il numero di porta da utilizzare per le connessioni SSH al dispositivo.

`connectivity.auth`

Informazioni di autenticazione per la connessione.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `ssh`.

`connectivity.auth.method`

Il metodo di autorizzazione utilizzato per accedere a un dispositivo con un determinato protocollo di connettività.

I valori supportati sono:

- `pki`
- `password`

`connectivity.auth.credentials`

Le credenziali utilizzate per l'autenticazione.

`connectivity.auth.credentials.user`

Il nome utente utilizzato per accedere al dispositivo.

`connectivity.auth.credentials.privKeyPath`

Il percorso completo della chiave privata utilizzata per accedere al dispositivo.

Questo valore si applica solo se `connectivity.auth.method` è impostato su `pki`.

`devices.connectivity.auth.credentials.password`

La password utilizzata per accedere al dispositivo.

Questo valore si applica solo se `connectivity.auth.method` è impostato su `password`.

Note

Specificare `privKeyPath` solo se `method` è impostato su `pki`.

Specificare `password` solo se `method` è impostato su `password`.

Crea la suite di test di esempio

La *<device-tester-extract-location>*/samples/python cartella contiene file di configurazione di esempio, codice sorgente e IDT Client SDK che puoi combinare in una suite di test utilizzando gli script di build forniti. Il seguente albero di directory mostra la posizione di questi file di esempio:

```
<device-tester-extract-location>
### ...
### tests
### samples
#   ### ...
#   ### python
#       ### configuration
#       ### src
#       ### build-scripts
#           ### build.sh
#           ### build.ps1
### sdks
    ### ...
    ### python
        ### idt_client
```

Per creare la suite di test, esegui i seguenti comandi sul tuo computer host:

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
```

In questo modo viene creata la suite di test di esempio nella IDTSampleSuitePython_1.0.0 cartella all'interno della <device-tester-extract-location>/tests cartella. Esamina i file nella IDTSampleSuitePython_1.0.0 cartella per capire come è strutturata la suite di test di esempio e guarda vari esempi di eseguibili per test case e file JSON di configurazione dei test.

Passaggio successivo: usa IDT per [eseguire la suite di test di esempio che hai creato](#).

Usa IDT per eseguire la suite di test di esempio

Per eseguire la suite di test di esempio, esegui i seguenti comandi sul tuo computer host:

```
cd <device-tester-extract-location>/bin  
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id IDTSampleSuitePython
```

IDT esegue la suite di test di esempio e trasmette i risultati alla console. Al termine dell'esecuzione del test, vengono visualizzate le seguenti informazioni:

```
===== 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
```


Risoluzione dei problemi

Utilizza le seguenti informazioni per risolvere eventuali problemi relativi al completamento del tutorial.

Il test case non viene eseguito correttamente

Se il test non viene eseguito correttamente, IDT trasmette i log degli errori alla console per aiutarti a risolvere i problemi relativi all'esecuzione del test. [Assicurati di soddisfare tutti i prerequisiti per questo tutorial.](#)

Impossibile connettersi al dispositivo in prova

Verificare quanto segue:

- Il `device.json` file contiene l'indirizzo IP, la porta e le informazioni di autenticazione corretti.
- Puoi connetterti al tuo dispositivo tramite SSH dal tuo computer host.

Tutorial: Sviluppa una semplice suite di test IDT

Una suite di test combina quanto segue:

- Esecuibili di test che contengono la logica di test
- File di configurazione JSON che descrivono la suite di test

Questo tutorial mostra come usare IDT per AWS IoT Greengrass sviluppare una suite di test Python che contenga un singolo test case. In questo tutorial, completerai i seguenti passaggi:

1. [Crea una directory per la suite di test](#)
2. [Crea file di configurazione JSON](#)
3. [Crea l'eseguibile del test case](#)
4. [Esegui la suite di test](#)

Prerequisiti

Per completare questo tutorial, è necessario quanto segue:

- Requisiti del computer host
 - Versione più recente di AWS IoT Device Tester

- [Python 3.7](#) o successivo

Per verificare la versione di Python installata sul tuo computer, esegui il seguente comando:

```
python3 --version
```

In Windows, se l'utilizzo di questo comando restituisce un errore, usalo `python --version` invece. Se il numero di versione restituito è 3.7 o superiore, esegui il comando seguente in un terminale Powershell da impostare `python3` come alias per il comando. `python`

```
Set-Alias -Name "python3" -Value "python"
```

Se non viene restituita alcuna informazione sulla versione o se il numero di versione è inferiore a 3.7, segui le istruzioni in [Downloading Python per installare Python 3.7+](#). Per ulteriori informazioni, consulta la documentazione di [Python](#).

- [urllib3](#)

Per verificare che `urllib3` sia installato correttamente, esegui il seguente comando:

```
python3 -c 'import urllib3'
```

Se non `urllib3` è installato, esegui il seguente comando per installarlo:

```
python3 -m pip install urllib3
```

- Requisiti per il dispositivo
- Un dispositivo con un sistema operativo Linux e una connessione di rete alla stessa rete del computer host.

Ti consigliamo di utilizzare un [Raspberry Pi](#) con sistema operativo Raspberry Pi. Assicurati di aver configurato [SSH](#) sul tuo Raspberry Pi per connetterti in remoto ad esso.

Crea una directory per la suite di test

IDT separa logicamente i casi di test in gruppi di test all'interno di ciascuna suite di test. Ogni test case deve essere all'interno di un gruppo di test. Per questo tutorial, crea una cartella chiamata `MyTestSuite_1.0.0` e crea il seguente albero di directory all'interno di questa cartella:

```
MyTestSuite_1.0.0
### suite
    ### myTestGroup
        ### myTestCase
```

Crea file di configurazione JSON

La tua suite di test deve contenere i seguenti file di [configurazione JSON](#) richiesti:

File JSON richiesti

`suite.json`

Contiene informazioni sulla suite di test. Per informazioni, consulta [Configura suite.json](#).

`group.json`

Contiene informazioni su un gruppo di test. È necessario creare un `group.json` file per ogni gruppo di test nella suite di test. Per informazioni, consulta [Configura group.json](#).

`test.json`

Contiene informazioni su un test case. È necessario creare un `test.json` file per ogni test case nella suite di test. Per informazioni, consulta [Configura test.json](#).

1. Nella `MyTestSuite_1.0.0/suite` cartella, create un `suite.json` file con la seguente struttura:

```
{
  "id": "MyTestSuite_1.0.0",
  "title": "My Test Suite",
  "details": "This is my test suite.",
  "userDataRequired": false
}
```

2. Nella `MyTestSuite_1.0.0/myTestGroup` cartella, create un `group.json` file con la seguente struttura:

```
{
  "id": "MyTestGroup",
  "title": "My Test Group",
```

```

    "details": "This is my test group.",
    "optional": false
  }

```

3. Nella `MyTestSuite_1.0.0/myTestGroup/myTestCase` cartella, create un `test.json` file con la seguente struttura:

```

{
  "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"
      ]
    }
  }
}

```

L'albero delle cartelle della `MyTestSuite_1.0.0` cartella dovrebbe ora avere il seguente aspetto:

```

MyTestSuite_1.0.0
### suite
### suite.json
### myTestGroup
### group.json
### myTestCase

```

```
### test.json
```

Scarica l'SDK del client IDT

Utilizzi l'[SDK del client IDT](#) per consentire a IDT di interagire con il dispositivo sottoposto a test e di riportare i risultati del test. Per questo tutorial, utilizzerai la versione Python dell'SDK.

Dalla *<device-tester-extract-location>/sdks/python/* cartella, copia la `idt_client` cartella nella tua `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase` cartella.

Per verificare che l'SDK sia stato copiato correttamente, esegui il comando seguente.

```
cd MyTestSuite_1.0.0/suite/myTestGroup/myTestCase
python3 -c 'import idt_client'
```

Crea l'eseguibile del test case

Gli eseguibili del test case contengono la logica di test che si desidera eseguire. Una suite di test può contenere più eseguibili di test case. Per questo tutorial, creerai un solo eseguibile di test case.

1. Crea il file della suite di test.

Nella `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase` cartella, crea un `myTestCase.py` file con il seguente contenuto:

```
from idt_client import *

def main():
    # Use the client SDK to communicate with IDT
    client = Client()

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

2. Utilizza le funzioni dell'SDK del client per aggiungere la seguente logica di test al tuo `myTestCase.py` file:

- a. Esegui un comando SSH sul dispositivo sottoposto a test.

```
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. Invia il risultato del test a 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()
```

Configura le informazioni sul dispositivo per IDT

Configura le informazioni sul dispositivo per consentire a IDT di eseguire il test. È necessario aggiornare il `device.json` modello che si trova nella `<device-tester-extract-location>/configs` cartella con le seguenti informazioni.

```
[
  {
    "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>"
            }
          }
        }
      }
    ]
  }
]
```

Nell'`devices` oggetto, fornite le seguenti informazioni:

`id`

Un identificatore univoco definito dall'utente per il dispositivo.

`connectivity.ip`

L'indirizzo IP del dispositivo.

`connectivity.port`

Facoltativo. Il numero di porta da utilizzare per le connessioni SSH al dispositivo.

`connectivity.auth`

Informazioni di autenticazione per la connessione.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `ssh`.

`connectivity.auth.method`

Il metodo di autorizzazione utilizzato per accedere a un dispositivo con un determinato protocollo di connettività.

I valori supportati sono:

- `pki`
- `password`

`connectivity.auth.credentials`

Le credenziali utilizzate per l'autenticazione.

`connectivity.auth.credentials.user`

Il nome utente utilizzato per accedere al dispositivo.

`connectivity.auth.credentials.privKeyPath`

Il percorso completo della chiave privata utilizzata per accedere al dispositivo.

Questo valore si applica solo se `connectivity.auth.method` è impostato su `pki`.

`devices.connectivity.auth.credentials.password`

La password utilizzata per accedere al dispositivo.

Questo valore si applica solo se `connectivity.auth.method` è impostato su `password`.

Note

Specificare `privKeyPath` solo se `method` è impostato su `pki`.
Specificare `password` solo se `method` è impostato su `password`.

Esegui la suite di test

Dopo aver creato la suite di test, devi assicurarti che funzioni come previsto. Completa i seguenti passaggi per eseguire la suite di test con il pool di dispositivi esistente a tale scopo.

1. Copia la MyTestSuite_1.0.0 cartella in *<device-tester-extract-location>/tests*.
2. Esegui i comandi seguenti:

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id MyTestSuite
```

IDT esegue la tua suite di test e trasmette i risultati alla console. Al termine dell'esecuzione del test, vengono visualizzate le seguenti informazioni:

```
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
```

Risoluzione dei problemi

Utilizza le seguenti informazioni per risolvere eventuali problemi relativi al completamento del tutorial.

Il test case non viene eseguito correttamente

Se il test non viene eseguito correttamente, IDT trasmette i log degli errori alla console per aiutarti a risolvere i problemi relativi all'esecuzione del test. Prima di controllare i log degli errori, verifica quanto segue:

- [L'SDK del client IDT si trova nella cartella corretta, come descritto in questo passaggio.](#)
- Soddisfi tutti i [prerequisiti](#) per questo tutorial.

Impossibile connettersi al dispositivo in prova

Verificare quanto segue:

- Il `device.json` file contiene l'indirizzo IP, la porta e le informazioni di autenticazione corretti.
- Puoi connetterti al tuo dispositivo tramite SSH dal tuo computer host.

Crea file di configurazione della suite di test IDT

Questa sezione descrive i formati in cui crei i file di configurazione JSON da includere quando scrivi una suite di test personalizzata.

File JSON richiesti

`suite.json`

Contiene informazioni sulla suite di test. Per informazioni, consulta [Configura suite.json](#).

`group.json`

Contiene informazioni su un gruppo di test. È necessario creare un `group.json` file per ogni gruppo di test nella suite di test. Per informazioni, consulta [Configura group.json](#).

`test.json`

Contiene informazioni su un test case. È necessario creare un `test.json` file per ogni test case nella suite di test. Per informazioni, consulta [Configura test.json](#).

File JSON opzionali

state_machine.json

Definisce come vengono eseguiti i test quando IDT esegue la suite di test. Per informazioni, consulta [Configura state_machine.json](#).

userdata_schema.json

Definisce lo schema per il [userdata.jsonfile](#) che i test runner possono includere nella configurazione delle impostazioni. Il `userdata.json` file viene utilizzato per tutte le informazioni di configurazione aggiuntive necessarie per eseguire il test ma che non sono presenti nel `device.json` file. Per informazioni, consulta [Configura userdata_schema.json](#).

I file di configurazione JSON vengono inseriti nel file *<custom-test-suite-folder>* come mostrato qui.

```
<custom-test-suite-folder>
### suite
  ### suite.json
  ### state_machine.json
  ### userdata_schema.json
  ### <test-group-folder>
    ### group.json
    ### <test-case-folder>
      ### test.json
```

Configura suite.json

Il `suite.json` file imposta le variabili di ambiente e determina se i dati utente sono necessari per eseguire la suite di test. Utilizzate il seguente modello per configurare il *<custom-test-suite-folder>/suite/suite.json* file:

```
{
  "id": "<suite-name>_<suite-version>",
  "title": "<suite-title>",
  "details": "<suite-details>",
  "userDataRequired": true | false,
  "environmentVariables": [
    {
      "key": "<name>",
```

```
        "value": "<value>",
    },
    ...
    {
        "key": "<name>",
        "value": "<value>",
    }
]
}
```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

id

Un ID univoco definito dall'utente per la suite di test. Il valore di `id` deve corrispondere al nome della cartella della suite di test in cui si trova il `suite.json` file. Il nome e la versione della suite devono inoltre soddisfare i seguenti requisiti:

- `<suite-name>` non può contenere caratteri di sottolineatura.
- `<suite-version>` è indicato come `x.x.x`, dove `x` è un numero.

L'ID viene visualizzato nei rapporti di test generati da IDT.

title

Un nome definito dall'utente per il prodotto o la funzionalità testata da questa suite di test. Il nome viene visualizzato nella CLI IDT per i test runner.

details

Una breve descrizione dello scopo della suite di test.

userDataRequired

Definisce se i test runner devono includere informazioni personalizzate in un `userdata.json` file. Se imposti questo valore su `true`, devi anche includere il [userdata_schema.jsonfile](#) nella cartella della suite di test.

environmentVariables

Facoltativo. Una serie di variabili di ambiente da impostare per questa suite di test.

environmentVariables.key

Il nome della variabile di ambiente.

`environmentVariables.value`

Il valore della variabile di ambiente.

Configura group.json

Il `group.json` file definisce se un gruppo di test è obbligatorio o facoltativo. Utilizzate il seguente modello per configurare il `<custom-test-suite-folder>/suite/<test-group>/group.json` file:

```
{
  "id": "<group-id>",
  "title": "<group-title>",
  "details": "<group-details>",
  "optional": true | false,
}
```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

`id`

Un ID univoco definito dall'utente per il gruppo di test. Il valore di `id` deve corrispondere al nome della cartella del gruppo di test in cui si trova il `group.json` file e non deve contenere caratteri di sottolineatura (`_`). L'ID viene utilizzato nei report di test generati da IDT.

`title`

Un nome descrittivo per il gruppo di test. Il nome viene visualizzato nella CLI IDT per i test runner.

`details`

Una breve descrizione dello scopo del gruppo di test.

`optional`

Facoltativo. Imposta `true` per visualizzare questo gruppo di test come gruppo opzionale dopo che IDT ha terminato l'esecuzione dei test richiesti. Il valore predefinito è `false`.

Configura test.json

Il `test.json` file determina gli eseguibili del test case e le variabili di ambiente utilizzate da un test case. Per ulteriori informazioni sulla creazione di eseguibili per i test case, vedere. [Crea eseguibili per test case IDT](#)

Utilizzate il seguente modello per configurare il `<custom-test-suite-folder>/suite/<test-group>/<test-case>/test.json` file:

```
{
  "id": "<test-id>",
  "title": "<test-title>",
  "details": "<test-details>",
  "requireDUT": true | false,
  "requiredResources": [
    {
      "name": "<resource-name>",
      "features": [
        {
          "name": "<feature-name>",
          "version": "<feature-version>",
          "jobSlots": <job-slots>
        }
      ]
    }
  ],
  "execution": {
    "timeout": <timeout>,
    "mac": {
      "cmd": "<path/to/executable>",
      "args": [
        "<argument>"
      ],
    },
    "linux": {
      "cmd": "<path/to/executable>",
      "args": [
        "<argument>"
      ],
    },
    "win": {
      "cmd": "<path/to/executable>",
      "args": [
```

```
        "<argument>"
    ]
  },
  "environmentVariables": [
    {
      "key": "<name>",
      "value": "<value>",
    }
  ]
}
```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

`id`

Un ID univoco definito dall'utente per il test case. Il valore di `id` deve corrispondere al nome della cartella del test case in cui si trova il `test.json` file e non deve contenere caratteri di sottolineatura (`_`). L'ID viene utilizzato nei report di test generati da IDT.

`title`

Un nome descrittivo per il test case. Il nome viene visualizzato nella CLI IDT per i test runner.

`details`

Una breve descrizione dello scopo del test case.

`requireDUT`

Facoltativo. Imposta `true` se è necessario un dispositivo per eseguire questo test, altrimenti imposta su `false`. Il valore predefinito è `true`. I test runner configureranno i dispositivi che useranno per eseguire il test nel proprio `device.json` file.

`requiredResources`

Facoltativo. Un array che fornisce informazioni sui dispositivi di risorse necessari per eseguire questo test.

`requiredResources.name`

Il nome univoco da assegnare al dispositivo di risorse durante l'esecuzione di questo test.

`requiredResources.features`

Una serie di funzionalità del dispositivo di risorse definite dall'utente.

`requiredResources.features.name`

Il nome della funzionalità. La funzionalità del dispositivo per cui si desidera utilizzare questo dispositivo. Questo nome viene confrontato con il nome della funzionalità fornito dal test runner nel `resource.json` file.

`requiredResources.features.version`

Facoltativo. La versione della funzionalità. Questo valore viene confrontato con la versione della funzionalità fornita dal test runner nel `resource.json` file. Se non viene fornita una versione, la funzionalità non viene verificata. Se non è richiesto un numero di versione per la funzionalità, lascia vuoto questo campo.

`requiredResources.features.jobSlots`

Facoltativo. Il numero di test simultanei che questa funzionalità può supportare. Il valore predefinito è 1. Se desideri che IDT utilizzi dispositivi distinti per le singole funzionalità, ti consigliamo di impostare questo valore su 1.

`execution.timeout`

La quantità di tempo (in millisecondi) che IDT attende prima che il test finisca. Per ulteriori informazioni sull'impostazione di questo valore, vedere [Crea eseguibili per test case IDT](#)

`execution.os`

Gli eseguibili del test case da eseguire in base al sistema operativo del computer host che esegue IDT. I valori supportati sono `linux`, `mac` e `win`.

`execution.os.cmd`

Il percorso dell'eseguibile del test case che si desidera eseguire per il sistema operativo specificato. Questa posizione deve trovarsi nel percorso di sistema.

`execution.os.args`

Facoltativo. Gli argomenti da fornire per eseguire l'eseguibile del test case.

`environmentVariables`

Facoltativo. Una serie di variabili di ambiente impostate per questo test case.

`environmentVariables.key`

Il nome della variabile di ambiente.

`environmentVariables.value`

Il valore della variabile di ambiente.

 Note

Se specificate la stessa variabile di ambiente nel `test.json` file e nel `suite.json` file, il valore nel `test.json` file ha la precedenza.

Configura `state_machine.json`

Una macchina a stati è un costrutto che controlla il flusso di esecuzione della suite di test. Determina lo stato iniziale di una suite di test, gestisce le transizioni di stato in base a regole definite dall'utente e continua la transizione attraverso tali stati fino a raggiungere lo stato finale.

Se la vostra suite di test non include una macchina a stati definita dall'utente, IDT genererà una macchina a stati per voi. La macchina a stati predefinita svolge le seguenti funzioni:

- Fornisce ai test runner la possibilità di selezionare ed eseguire gruppi di test specifici, anziché l'intera suite di test.
- Se non sono selezionati gruppi di test specifici, esegue ogni gruppo di test nella suite di test in ordine casuale.
- Genera report e stampa un riepilogo della console che mostra i risultati dei test per ogni gruppo di test e test case.

Per ulteriori informazioni sul funzionamento della macchina a stati IDT, vedere [Configurare la macchina a stati IDT](#).

Configura `userdata_schema.json`

Il `userdata_schema.json` file determina lo schema in cui i test runner forniscono i dati degli utenti. I dati utente sono necessari se la suite di test richiede informazioni che non sono presenti nel `device.json` file. Ad esempio, i test potrebbero richiedere credenziali di rete Wi-Fi, porte aperte specifiche o certificati che un utente deve fornire. Queste informazioni possono essere fornite a IDT come parametro di input chiamato `userdata`, il cui valore è un `userdata.json` file, che gli utenti creano nella propria `<device-tester-extract-location>/config` cartella. Il formato del `userdata.json` file si basa sul `userdata_schema.json` file incluso nella suite di test.

Per indicare che i test runner devono fornire un `userdata.json` file:

1. Nel `suite.json` file, imposta `suuserDataRequired: true`
2. Nel tuo `<custom-test-suite-folder>`, crea un `userdata_schema.json` file.
3. Modifica il `userdata_schema.json` file per creare uno schema [JSON IETF Draft v4](#) valido.

Quando IDT esegue la suite di test, legge automaticamente lo schema e lo usa per convalidare il `userdata.json` file fornito dal test runner. [Se valido, il contenuto del `userdata.json` file è disponibile sia nel contesto IDT che nel contesto della macchina a stati.](#)

Configurare la macchina a stati IDT

Una macchina a stati è un costrutto che controlla il flusso di esecuzione della suite di test. Determina lo stato iniziale di una suite di test, gestisce le transizioni di stato in base a regole definite dall'utente e continua la transizione attraverso tali stati fino a raggiungere lo stato finale.

Se la vostra suite di test non include una macchina a stati definita dall'utente, IDT genererà una macchina a stati per voi. La macchina a stati predefinita svolge le seguenti funzioni:

- Fornisce ai test runner la possibilità di selezionare ed eseguire gruppi di test specifici, anziché l'intera suite di test.
- Se non sono selezionati gruppi di test specifici, esegue ogni gruppo di test nella suite di test in ordine casuale.
- Genera report e stampa un riepilogo della console che mostra i risultati dei test per ogni gruppo di test e test case.

La macchina a stati per una suite di test IDT deve soddisfare i seguenti criteri:

- Ogni stato corrisponde a un'azione che IDT deve intraprendere, ad esempio eseguire un gruppo di test o produrre un file di report.
- La transizione a uno stato esegue l'azione associata allo stato.
- Ogni stato definisce la regola di transizione per lo stato successivo.
- Lo stato finale deve essere `Succeed` o `Fail`.

Formato macchina a stati

È possibile utilizzare il seguente modello per configurare il proprio *<custom-test-suite-folder>/suite/state_machine.json* file:

```
{
  "Comment": "<description>",
  "StartAt": "<state-name>",
  "States": {
    "<state-name>": {
      "Type": "<state-type>",
      // Additional state configuration
    }

    // Required states
    "Succeed": {
      "Type": "Succeed"
    },
    "Fail": {
      "Type": "Fail"
    }
  }
}
```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

Comment

Una descrizione della macchina a stati.

StartAt

Il nome dello stato in cui IDT inizia a eseguire la suite di test. Il valore di `StartAt` deve essere impostato su uno degli stati elencati nell'`States` oggetto.

States

Un oggetto che associa i nomi di stato definiti dall'utente a stati IDT validi. Ciascuno stato. *state-name* l'oggetto contiene la definizione di uno stato valido mappato su. *state-name*

L'`States` oggetto deve includere gli `Fail` stati `Succeed` e. Per informazioni sugli stati validi, vedere [Stati e definizioni di stato validi](#).

Stati e definizioni di stato validi

Questa sezione descrive le definizioni di stato di tutti gli stati validi che possono essere utilizzati nella macchina a stati IDT. Alcuni dei seguenti stati supportano configurazioni a livello di test case. Tuttavia, si consiglia di configurare le regole di transizione degli stati a livello di gruppo di test anziché a livello di test case, a meno che non sia assolutamente necessario.

Definizioni di stato

- [RunTask](#)
- [Choice](#)
- [Parallel](#)
- [AddProductFeatures](#)
- [Report](#)
- [LogMessage](#)
- [SelectGroup](#)
- [Fail](#)
- [Succeed](#)

RunTask

Lo RunTask stato esegue casi di test da un gruppo di test definito nella suite di test.

```
{
  "Type": "RunTask",
  "Next": "<state-name>",
  "TestGroup": "<group-id>",
  "TestCases": [
    "<test-id>"
  ],
  "ResultVar": "<result-name>"
}
```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

Next

Il nome dello stato a cui passare dopo l'esecuzione delle azioni nello stato corrente.

TestGroup

Facoltativo. L'ID del gruppo di test da eseguire. Se questo valore non è specificato, IDT esegue il gruppo di test selezionato dal test runner.

TestCases

Facoltativo. Una serie di test case del gruppo IDs specificato in. TestGroup In base ai valori di TestGroup and TestCases, IDT determina il comportamento di esecuzione del test nel modo seguente:

- Quando entrambi TestGroup TestCases sono specificati, IDT esegue i casi di test specificati dal gruppo di test.
- Quando TestCases sono specificati ma non TestGroup sono specificati, IDT esegue i casi di test specificati.
- Quando TestGroup è specificato, ma non TestCases è specificato, IDT esegue tutti i casi di test all'interno del gruppo di test specificato.
- Quando nessuno dei due TestGroup TestCases è specificato, IDT esegue tutti i casi di test dal gruppo di test selezionato dal test runner dalla CLI IDT. Per abilitare la selezione di gruppo per i test runner, è necessario includere entrambi RunTask gli stati nel file. Choice statemachine.json Per un esempio di come funziona, vedi [Example state machine: Esegui gruppi di test selezionati dall'utente](#).

Per ulteriori informazioni sull'abilitazione dei comandi CLI IDT per i test runner, vedere. [the section called “Abilita i comandi CLI IDT”](#)

ResultVar

Il nome della variabile di contesto da impostare con i risultati dell'esecuzione del test. Non specificate questo valore se non avete specificato un valore per TestGroup. IDT imposta il valore della variabile definita in true o in false base ResultVar a quanto segue:

- Se il nome della variabile è nel formato `text_text_passed`, il valore viene impostato in base al fatto che tutti i test del primo gruppo di test siano stati superati o ignorati.
- In tutti gli altri casi, il valore è impostato in base al fatto che tutti i test di tutti i gruppi di test siano stati superati o saltati.

In genere, si utilizza RunTask lo stato per specificare un ID del gruppo di test senza specificare il singolo test case IDs, in modo che IDT esegua tutti i casi di test nel gruppo di test specificato. Tutti i casi di test eseguiti da questo stato vengono eseguiti in parallelo, in ordine casuale. Tuttavia, se tutti

i casi di test richiedono l'esecuzione di un dispositivo ed è disponibile un solo dispositivo, i test case verranno invece eseguiti in sequenza.

Gestione errori

Se uno dei gruppi di test o dei test case IDs specificati non è valido, questo stato genera l'errore di `RunTaskError` esecuzione. Se lo stato rileva un errore di esecuzione, imposta anche la `hasExecutionError` variabile nel contesto della macchina a stati su. `true`

Choice

Lo `Choice` stato consente di impostare dinamicamente lo stato successivo a cui passare in base a condizioni definite dall'utente.

```
{
  "Type": "Choice",
  "Default": "<state-name>",
  "FallthroughOnError": true | false,
  "Choices": [
    {
      "Expression": "<expression>",
      "Next": "<state-name>"
    }
  ]
}
```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

Default

Lo stato predefinito a cui passare se nessuna delle espressioni definite in `Choices` può essere valutata. `true`

FallthroughOnError

Facoltativo. Specifica il comportamento quando lo stato rileva un errore nella valutazione delle espressioni. Imposta su `true` se desideri saltare un'espressione se la valutazione genera un errore. Se nessuna espressione corrisponde, la macchina a stati passa allo `Default` stato. Se il `FallthroughOnError` valore non è specificato, il valore predefinito è. `false`

Choices

Un array di espressioni e stati per determinare a quale stato passare dopo aver eseguito le azioni nello stato corrente.

Choices.Expression

Una stringa di espressione che restituisce un valore booleano. Se l'espressione restituisce `true`, la macchina a stati passa allo stato definito in `Choices.Next`. Le stringhe di espressione recuperano i valori dal contesto della macchina a stati e quindi eseguono operazioni su di essi per ottenere un valore booleano. Per informazioni sull'accesso al contesto della macchina a stati, vedere [Contesto della macchina a stati](#).

Choices.Next

Il nome dello stato a cui passare se l'espressione definita in `Choices.Expression` restituisce `true`.

Gestione errori

Lo `Choice` stato può richiedere la gestione degli errori nei seguenti casi:

- Alcune variabili nelle espressioni di scelta non esistono nel contesto della macchina a stati.
- Il risultato di un'espressione non è un valore booleano.
- Il risultato di una ricerca JSON non è una stringa, un numero o un valore booleano.

Non è possibile utilizzare un `Catch` blocco per gestire gli errori in questo stato. Se si desidera interrompere l'esecuzione della macchina a stati quando rileva un errore, è necessario impostare su `FallthroughOnError` `false`. Tuttavia, ti consigliamo di impostare `true`, `FallthroughOnError` a seconda del caso d'uso, di eseguire una delle seguenti operazioni:

- Se in alcuni casi si prevede che una variabile a cui stai accedendo non esista, utilizza il valore `Default` e i `Choices` blocchi aggiuntivi per specificare lo stato successivo.
- Se una variabile a cui stai accedendo deve sempre esistere, imposta lo `Default` stato su `Fail`.

Parallel

Lo `Parallel` stato consente di definire ed eseguire nuove macchine a stati in parallelo tra loro.

```
{
  "Type": "Parallel",
  "Next": "<state-name>",
  "Branches": [
```

```
    <state-machine-definition>  
  ]  
}
```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

Next

Il nome dello stato a cui passare dopo l'esecuzione delle azioni nello stato corrente.

Branches

Una serie di definizioni di macchine a stati da eseguire. Ogni definizione di macchina a stati deve contenere `StartAt`, `Succeed` i propri `Fail` stati. Le definizioni delle macchine a stati in questo array non possono fare riferimento a stati al di fuori della propria definizione.

Note

Poiché ogni macchina a stati della filiale condivide lo stesso contesto della macchina a stati, l'impostazione delle variabili in un ramo e la successiva lettura di tali variabili da un altro ramo potrebbe causare un comportamento imprevisto.

Lo `Parallel` stato passa allo stato successivo solo dopo aver eseguito tutte le macchine a stati della filiale. Ogni stato che richiede un dispositivo aspetterà di funzionare finché il dispositivo non sarà disponibile. Se sono disponibili più dispositivi, questo stato esegue casi di test da più gruppi in parallelo. Se non sono disponibili dispositivi sufficienti, i test case verranno eseguiti in sequenza. Poiché i casi di test vengono eseguiti in ordine casuale quando vengono eseguiti in parallelo, è possibile utilizzare dispositivi diversi per eseguire i test dello stesso gruppo di test.

Gestione errori

Assicurati che sia la macchina a stati della filiale che la macchina a stati principale passino allo `Fail` stato per gestire gli errori di esecuzione.

Poiché le macchine a stati delle filiali non trasmettono errori di esecuzione alla macchina a stato principale, non è possibile utilizzare un `Catch` blocco per gestire gli errori di esecuzione nelle macchine a stato delle filiali. Utilizzate invece il `hasExecutionErrors` valore nel contesto della macchina a stati condivisa. Per un esempio di come funziona, vedi [Esempio di macchina a stati: esecuzione di due gruppi di test in parallelo](#).

AddProductFeatures

Lo AddProductFeatures stato consente di aggiungere funzionalità del prodotto al `awsiotdevicetester_report.xml` file generato da IDT.

Una funzionalità del prodotto è costituita da informazioni definite dall'utente su criteri specifici che un dispositivo potrebbe soddisfare. Ad esempio, la funzionalità del MQTT prodotto può indicare che il dispositivo pubblica correttamente i messaggi MQTT. Nel rapporto, le caratteristiche del prodotto sono impostate come o come valore personalizzato `supportednot-supported`, in base al superamento dei test specificati.

Note

Lo AddProductFeatures stato non genera report da solo. Questo stato deve passare allo [Reportstato](#) per generare report.

```
{
  "Type": "Parallel",
  "Next": "<state-name>",
  "Features": [
    {
      "Feature": "<feature-name>",
      "Groups": [
        "<group-id>"
      ],
      "OneOfGroups": [
        "<group-id>"
      ],
      "TestCases": [
        "<test-id>"
      ],
      "IsRequired": true | false,
      "ExecutionMethods": [
        "<execution-method>"
      ]
    }
  ]
}
```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

Next

Il nome dello stato a cui passare dopo l'esecuzione delle azioni nello stato corrente.

Features

Una serie di caratteristiche del prodotto da mostrare nel `awsiotdevicetester_report.xml` file.

Feature

Il nome della funzionalità

FeatureValue

Facoltativo. Il valore personalizzato da utilizzare nel rapporto anziché `supported`. Se questo valore non è specificato, in base ai risultati del test, il valore della funzionalità viene impostato su `supported` o `not-supported`.

Se utilizzate un valore personalizzato per `FeatureValue`, potete testare la stessa funzionalità con condizioni diverse e IDT concatena i valori delle caratteristiche per le condizioni supportate. Ad esempio, il seguente estratto mostra la funzionalità con due valori di `MyFeature` funzionalità separati:

```
...
{
  "Feature": "MyFeature",
  "FeatureValue": "first-feature-supported",
  "Groups": ["first-feature-group"]
},
{
  "Feature": "MyFeature",
  "FeatureValue": "second-feature-supported",
  "Groups": ["second-feature-group"]
},
...
```

Se entrambi i gruppi di test superano il test, il valore della funzionalità viene impostato su `first-feature-supported`, `second-feature-supported`

Groups

Facoltativo. Una serie di gruppi di test IDs. Tutti i test all'interno di ogni gruppo di test specificato devono essere superati affinché la funzionalità sia supportata.

OneOfGroups

Facoltativo. Una serie di gruppi di test IDs. Tutti i test all'interno di almeno uno dei gruppi di test specificati devono essere superati affinché la funzionalità sia supportata.

TestCases

Facoltativo. Una serie di test case IDs. Se si specifica questo valore, si applica quanto segue:

- Tutti i casi di test specificati devono essere superati affinché la funzionalità sia supportata.
- `Groups` deve contenere un solo ID del gruppo di test.
- `OneOfGroups` non deve essere specificato.

IsRequired

Facoltativo. Imposta `false` su per contrassegnare questa funzionalità come opzionale nel rapporto. Il valore predefinito è `true`.

ExecutionMethods

Facoltativo. Una serie di metodi di esecuzione che corrispondono al `protocol` valore specificato nel `device.json` file. Se viene specificato questo valore, i test runner devono specificare un `protocol` valore che corrisponda a uno dei valori di questo array per includere la funzionalità nel report. Se questo valore non è specificato, la funzionalità verrà sempre inclusa nel rapporto.

Per utilizzare lo `AddProductFeatures` stato, è necessario impostare il valore di `ResultVar` in the `RunTask` state su uno dei seguenti valori:

- Se hai specificato un singolo test case IDs, imposta `ResultVar` su `group-id_test-id_passed`.
- Se non hai specificato un singolo test case IDs, imposta `ResultVar` su `group-id_passed`.

Lo `AddProductFeatures` stato verifica i risultati dei test nel modo seguente:

- Se non è stato specificato alcun test case IDs, il risultato per ogni gruppo di test viene determinato dal valore della `group-id_passed` variabile nel contesto della macchina a stati.
- Se hai specificato il test case IDs, il risultato di ciascuno dei test viene determinato dal valore della `group-id_test-id_passed` variabile nel contesto della macchina a stati.

Gestione errori

Se un ID di gruppo fornito in questo stato non è un ID di gruppo valido, questo stato genera l'errore di `AddProductFeaturesError` esecuzione. Se lo stato rileva un errore di esecuzione, imposta anche la `hasExecutionErrors` variabile nel contesto della macchina a stati su `true`

Report

Lo `Report` stato genera i file `suite-name_Report.xml` `andawsiotdevicetester_report.xml`. Questo stato trasmette inoltre il report alla console.

```
{
  "Type": "Report",
  "Next": "<state-name>"
}
```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

Next

Il nome dello stato a cui passare dopo l'esecuzione delle azioni nello stato corrente.

Dovresti sempre passare `Report` allo stato verso la fine del flusso di esecuzione del test in modo che i test runner possano visualizzare i risultati del test. In genere, lo stato successivo a questo è `Succeed`.

Gestione errori

Se questo stato riscontra problemi con la generazione dei report, genera l'errore di `ReportError` esecuzione.

LogMessage

Lo `LogMessage` stato genera il `test_manager.log` file e trasmette il messaggio di registro alla console.

```
{
  "Type": "LogMessage",
  "Next": "<state-name>"
  "Level": "info | warn | error"
  "Message": "<message>"
}
```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

Next

Il nome dello stato a cui passare dopo l'esecuzione delle azioni nello stato corrente.

Level

Il livello di errore al quale creare il messaggio di registro. Se si specifica un livello non valido, questo stato genera un messaggio di errore e lo elimina.

Message

Il messaggio da registrare.

SelectGroup

Lo `SelectGroup` stato aggiorna il contesto della macchina a stati per indicare quali gruppi sono selezionati. I valori impostati da questo stato vengono utilizzati da tutti `Choice` gli stati successivi.

```
{
  "Type": "SelectGroup",
  "Next": "<state-name>"
  "TestGroups": [
    <group-id>
  ]
}
```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

Next

Il nome dello stato a cui passare dopo l'esecuzione delle azioni nello stato corrente.

TestGroups

Una serie di gruppi di test che verranno contrassegnati come selezionati. Per ogni ID del gruppo di test in questo array, la `group-id_selected` variabile è impostata su `true` nel contesto. Assicuratevi di fornire un gruppo di test valido IDs perché IDT non verifica l'esistenza dei gruppi specificati.

Fail

Lo `Fail` stato indica che la macchina a stati non è stata eseguita correttamente. Si tratta di uno stato finale per la macchina a stati e ogni definizione di macchina a stati deve includere questo stato.

```
{
  "Type": "Fail"
}
```

Succeed

Lo Succeed stato indica che la macchina a stati è stata eseguita correttamente. Questo è uno stato finale per la macchina a stati e ogni definizione di macchina a stati deve includere questo stato.

```
{
  "Type": "Succeed"
}
```

Contesto della macchina a stati

Il contesto della macchina a stati è un documento JSON di sola lettura che contiene dati disponibili per la macchina a stati durante l'esecuzione. Il contesto della macchina a stati è accessibile solo dalla macchina a stati e contiene informazioni che determinano il flusso di test. Ad esempio, è possibile utilizzare le informazioni configurate dai test runner nel `userdata.json` file per determinare se è necessario eseguire un test specifico.

Il contesto della macchina a stati utilizza il seguente formato:

```
{
  "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

Informazioni sul pool di dispositivi selezionato per l'esecuzione del test. Per un pool di dispositivi selezionato, queste informazioni vengono recuperate dal corrispondente elemento dell'array del pool di dispositivi di primo livello definito nel `device.json` file.

userData

Informazioni contenute nel file. `userdata.json`

config

Le informazioni bloccano il `config.json` file.

suiteFailed

Il valore è impostato su `false` quando viene avviata la macchina a stati. Se un gruppo di test fallisce in uno `RunTask` stato, questo valore viene impostato `true` per la durata residua dell'esecuzione della macchina a stati.

specificTestGroups

Se il test runner seleziona gruppi di test specifici da eseguire anziché l'intera suite di test, questa chiave viene creata e contiene l'elenco dei gruppi di test specifici. IDs

specificTestCases

Se il test runner seleziona casi di test specifici da eseguire anziché l'intera suite di test, questa chiave viene creata e contiene l'elenco di casi di test specifici. IDs

hasExecutionErrors

Non esce all'avvio della macchina a stati. Se uno stato rileva errori di esecuzione, questa variabile viene creata e impostata `true` per la durata residua dell'esecuzione della macchina a stati.

È possibile interrogare il contesto utilizzando JSONPath la notazione. La sintassi per le JSONPath interrogazioni nelle definizioni di stato è. `{{$.query}}` È possibile utilizzare JSONPath le query come stringhe segnaposto in alcuni stati. IDT sostituisce le stringhe segnaposto con il valore della query valutata dal contesto. JSONPath È possibile utilizzare i segnaposto per i seguenti valori:

- Il `TestCases` valore in `RunTask` stati.
- Lo `Choice` stato `Expression` del valore.

Quando accedete ai dati dal contesto della macchina a stati, assicuratevi che siano soddisfatte le seguenti condizioni:

- I percorsi JSON devono iniziare con \$.
- Ogni valore deve restituire una stringa, un numero o un valore booleano.

Per ulteriori informazioni sull'utilizzo della JSONPath notazione per accedere ai dati dal contesto, vedere. [Usa il contesto IDT](#)

Errori di esecuzione

Gli errori di esecuzione sono errori nella definizione della macchina a stati che la macchina a stati incontra durante l'esecuzione di uno stato. IDT registra le informazioni su ogni errore nel `test_manager.log` file e trasmette il messaggio di registro alla console.

È possibile utilizzare i seguenti metodi per gestire gli errori di esecuzione:

- Aggiungere un [Catchblocco](#) nella definizione dello stato.
- Controlla il valore del [hasExecutionErrorsvalore](#) nel contesto della macchina a stati.

Cattura

Per utilizzarloCatch, aggiungi quanto segue alla definizione dello stato:

```
"Catch": [  
  {  
    "ErrorEquals": [  
      "<error-type>"  
    ]  
    "Next": "<state-name>"  
  }  
]
```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

Catch.ErrorEquals

Una serie di tipi di errore da catturare. Se un errore di esecuzione corrisponde a uno dei valori specificati, la macchina a stati passa allo stato specificato inCatch.Next. Consultate la definizione di ogni stato per informazioni sul tipo di errore che produce.

Catch.Next

Lo stato successivo a cui passare se lo stato corrente rileva un errore di esecuzione che corrisponde a uno dei valori specificati in `Catch.ErrorEquals`

I blocchi `Catch` vengono gestiti in sequenza fino a quando non ne corrisponde uno. Se gli errori non corrispondono a quelli elencati nei blocchi `Catch`, le macchine a stati continuano a essere eseguite. Poiché gli errori di esecuzione sono il risultato di definizioni di stato errate, si consiglia di passare allo stato `Fail` quando uno stato rileva un errore di esecuzione.

`hasExecutionError`

Quando alcuni stati riscontrano errori di esecuzione, oltre a emettere l'errore, impostano anche il `hasExecutionError` valore su `true` nel contesto della macchina a stati. È possibile utilizzare questo valore per rilevare quando si verifica un errore e quindi utilizzare uno `Choice` stato per trasferire la macchina a stati allo `Fail` stato.

Questo metodo presenta le seguenti caratteristiche.

- La macchina a stati non si avvia con alcun valore assegnato a `hasExecutionError` e questo valore non è disponibile finché non viene impostato da uno stato particolare. Ciò significa che è necessario impostare in modo esplicito il `FallthroughOnError` to `false` per gli `Choice` stati che accedono a questo valore per evitare che la macchina a stati si fermi se non si verificano errori di esecuzione.
- Una volta impostato su `true`, `hasExecutionError` viene mai impostato su `false` o rimosso dal contesto. Ciò significa che questo valore è utile solo la prima volta che viene impostato su e per tutti gli stati successivi non fornisce un valore significativo. `true`
- Il `hasExecutionError` valore è condiviso con tutte le macchine a stati delle filiali presenti `Parallel` nello stato, il che può generare risultati imprevisti a seconda dell'ordine in cui viene effettuato l'accesso.

A causa di queste caratteristiche, non è consigliabile utilizzare questo metodo se è possibile utilizzare invece un blocco `Catch`.

Esempi di macchine a stati

Questa sezione fornisce alcuni esempi di configurazioni di macchine a stati.

Esempi

- [Esempio di macchina a stati: Esegui un singolo gruppo di test](#)
- [Esempio di macchina a stati: esegue gruppi di test selezionati dall'utente](#)
- [Esempio di macchina a stati: esegui un singolo gruppo di test con le caratteristiche del prodotto](#)
- [Esempio di macchina a stati: esecuzione di due gruppi di test in parallelo](#)

Esempio di macchina a stati: Esegui un singolo gruppo di test

Questa macchina a stati:

- Esegue il gruppo di test con `idGroupA`, che deve essere presente nella suite in un `group.json` file.
- Verifica la presenza di errori di esecuzione e le `Fail` eventuali transizioni rilevate.
- Genera un rapporto e passa a `Succeed` se non ci sono errori e `Fail` altro.

```
{
  "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"
  }
}
}

```

Esempio di macchina a stati: esegue gruppi di test selezionati dall'utente

Questa macchina a stati:

- Verifica se il test runner ha selezionato gruppi di test specifici. La macchina a stati non verifica la presenza di casi di test specifici perché i test runner non possono selezionare casi di test senza selezionare anche un gruppo di test.
- Se sono selezionati gruppi di test:
 - Esegue i casi di test all'interno dei gruppi di test selezionati. A tale scopo, la macchina a stati non specifica esplicitamente alcun gruppo di test o caso di test nello RunTask stato.
 - Genera un rapporto dopo aver eseguito tutti i test e le uscite.
- Se i gruppi di test non sono selezionati:
 - Esegue i test nel gruppo di testGroupA.
 - Genera report ed uscite.

```

{
  "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"
    }
}
}
}

```

Esempio di macchina a stati: esegui un singolo gruppo di test con le caratteristiche del prodotto

Questa macchina a stati:

- Esegue il gruppo di testGroupA.
- Verifica la presenza di errori di esecuzione e le Fail eventuali transizioni rilevate.
- Aggiunge la FeatureThatDependsOnGroupA funzionalità al `awsiotdevicetester_report.xml` file:
 - Se ha GroupA esito positivo, la funzionalità è impostata `susupported`.
 - La funzionalità non è contrassegnata come opzionale nel rapporto.
- Genera un rapporto e passa a Succeed se non ci sono errori e altro Fail

```

{
  "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"
        }
    }
}

```

Esempio di macchina a stati: esecuzione di due gruppi di test in parallelo

Questa macchina a stati:

- Esegue i gruppi GroupA e GroupB test in parallelo. Le `ResultVar` variabili memorizzate nel contesto dagli `RunTask` stati nelle macchine a stati della filiale di sono disponibili per lo `AddProductFeatures` stato.
- Verifica la presenza di errori di esecuzione e le `Fail` eventuali transizioni rilevate. Questa macchina a stati non utilizza un `Catch` blocco perché tale metodo non rileva errori di esecuzione nelle macchine a stati delle filiali.
- Aggiunge funzionalità al `awsiotdevicetester_report.xml` file in base ai gruppi che passano

- Se ha GroupA esito positivo, la funzionalità è impostata susupported.
- La funzionalità non è contrassegnata come opzionale nel rapporto.
- Genera un rapporto e passa a Succeed se non ci sono errori e altro Fail

Se due dispositivi sono configurati nel pool di dispositivi, entrambi GroupA GroupB possono funzionare contemporaneamente. Tuttavia, se uno GroupA o GroupB più test sono inclusi, entrambi i dispositivi possono essere assegnati a tali test. Se è configurato un solo dispositivo, i gruppi di test verranno eseguiti in sequenza.

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

Crea eseguibili per test case IDT

È possibile creare e inserire gli eseguibili dei test case in una cartella di test suite nei seguenti modi:

- Per le suite di test che utilizzano argomenti o variabili di ambiente dei `test.json` file per determinare quali test eseguire, è possibile creare un singolo test case eseguibile per l'intera suite di test o un eseguibile di test per ogni gruppo di test nella suite di test.
- Per una suite di test in cui desideri eseguire test specifici in base a comandi specifici, crei un eseguibile di test case per ogni test case della suite di test.

In qualità di scrittore di test, puoi determinare quale approccio è appropriato per il tuo caso d'uso e strutturare di conseguenza il tuo eseguibile del test case. Assicurati di fornire il percorso eseguibile corretto del test case in ogni `test.json` file e che l'eseguibile specificato funzioni correttamente.

Quando tutti i dispositivi sono pronti per l'esecuzione di un test case, IDT legge i seguenti file:

- Il test case `test.json` per il test case selezionato determina i processi da avviare e le variabili di ambiente da impostare.
- La suite `suite.json` for the test determina le variabili di ambiente da impostare.

IDT avvia il processo eseguibile di test richiesto in base ai comandi e agli argomenti specificati nel `test.json` file e passa le variabili di ambiente richieste al processo.

Usa l'IDT Client SDK

Il client IDT ti SDKs consente di semplificare il modo in cui scrivi la logica di test nel tuo eseguibile di test con comandi API che puoi utilizzare per interagire con IDT e i tuoi dispositivi sottoposti a test. IDT attualmente fornisce quanto segue: SDKs

- SDK del client IDT per Python
- SDK del client IDT per Go

Questi si SDKs trovano nella cartella. `<device-tester-extract-location>/sdks` Quando crei un nuovo eseguibile del test case, devi copiare l'SDK che desideri utilizzare nella cartella che contiene l'eseguibile del test case e fare riferimento all'SDK nel tuo codice. Questa sezione fornisce una breve descrizione dei comandi API disponibili che puoi utilizzare negli eseguibili del test case.

In questa sezione

- [Interazione con il dispositivo](#)
- [Interazione IDT](#)

- [Interazione con l'host](#)

Interazione con il dispositivo

I seguenti comandi consentono di comunicare con il dispositivo sottoposto a test senza dover implementare ulteriori funzioni di interazione con il dispositivo e di gestione della connettività.

ExecuteOnDevice

Consente alle suite di test di eseguire comandi shell su un dispositivo che supporta connessioni shell SSH o Docker.

CopyToDevice

Consente alle suite di test di copiare un file locale dalla macchina host che esegue IDT in una posizione specificata su un dispositivo che supporta connessioni shell SSH o Docker.

ReadFromDevice

Consente alle suite di test di leggere dalla porta seriale dei dispositivi che supportano le connessioni UART.

Note

Poiché IDT non gestisce le connessioni dirette ai dispositivi effettuate utilizzando le informazioni di accesso ai dispositivi provenienti dal contesto, consigliamo di utilizzare questi comandi API di interazione con i dispositivi negli eseguibili dei test case. Tuttavia, se questi comandi non soddisfano i requisiti del test case, potete recuperare le informazioni di accesso al dispositivo dal contesto IDT e utilizzarle per stabilire una connessione diretta al dispositivo dalla suite di test.

Per stabilire una connessione diretta, recuperate le informazioni nei campi `device.connectivity` e nei `resource.devices.connectivity` campi per il dispositivo in esame e per i dispositivi di risorse, rispettivamente. Per ulteriori informazioni sull'utilizzo del contesto IDT, vedere. [Usa il contesto IDT](#)

Interazione IDT

I seguenti comandi consentono alle suite di test di comunicare con IDT.

PollForNotifications

Consente alle suite di test di verificare la presenza di notifiche da IDT.

GetContextValue e GetContextString

Consente alle suite di test di recuperare valori dal contesto IDT. Per ulteriori informazioni, consulta [Usa il contesto IDT](#).

SendResult

Consente alle suite di test di riportare i risultati dei test case a IDT. Questo comando deve essere chiamato alla fine di ogni test case in una suite di test.

Interazione con l'host

Il comando seguente consente alle suite di test di comunicare con la macchina host.

PollForNotifications

Consente alle suite di test di verificare la presenza di notifiche da IDT.

GetContextValue e GetContextString

Consente alle suite di test di recuperare valori dal contesto IDT. Per ulteriori informazioni, consulta [Usa il contesto IDT](#).

ExecuteOnHost

Consente alle suite di test di eseguire comandi sulla macchina locale e consente a IDT di gestire il ciclo di vita eseguibile del test case.

Abilita i comandi CLI IDT

Il `run-suite` comando IDT CLI fornisce diverse opzioni che consentono a test runner di personalizzare l'esecuzione del test. Per consentire ai test runner di utilizzare queste opzioni per eseguire la suite di test personalizzata, è necessario implementare il supporto per la CLI IDT. Se non si implementa il supporto, i test runner saranno comunque in grado di eseguire i test, ma alcune opzioni della CLI non funzioneranno correttamente. Per fornire un'esperienza cliente ideale, si consiglia di implementare il supporto per i seguenti argomenti per il `run-suite` comando nella CLI IDT:

timeout-multiplier

Specifica un valore maggiore di 1,0 che verrà applicato a tutti i timeout durante l'esecuzione dei test.

I test runner possono utilizzare questo argomento per aumentare il timeout per i test case che desiderano eseguire. Quando un test runner specifica questo argomento nel proprio `run-suite` comando, IDT lo utilizza per calcolare il valore della variabile di ambiente `IDT_TEST_TIMEOUT` e imposta il campo nel contesto IDT. `config.timeoutMultiplier` Per supportare questo argomento, devi fare quanto segue:

- Invece di utilizzare direttamente il valore di timeout del `test.json` file, leggete la variabile di ambiente `IDT_TEST_TIMEOUT` per ottenere il valore di timeout calcolato correttamente.
- Recuperate il `config.timeoutMultiplier` valore dal contesto IDT e applicatelo a timeout di esecuzione prolungati.

Per ulteriori informazioni sull'uscita anticipata a causa di eventi di timeout, consulta. [Specificate il comportamento di uscita](#)

stop-on-first-failure

Specifica che IDT deve interrompere l'esecuzione di tutti i test in caso di errore.

Quando un test runner specifica questo argomento nel proprio `run-suite` comando, IDT interromperà l'esecuzione dei test non appena riscontra un errore. Tuttavia, se i test case vengono eseguiti in parallelo, ciò può portare a risultati imprevisti. Per implementare il supporto, assicuratevi che se IDT rileva questo evento, la logica di test indichi a tutti i casi di test in esecuzione di interrompersi, ripulisca le risorse temporanee e riporti un risultato del test a IDT. Per ulteriori informazioni sull'uscita anticipata in caso di guasto, consulta. [Specificate il comportamento di uscita](#)

group-id e test-id

Specifica che IDT deve eseguire solo i gruppi di test o i casi di test selezionati.

I test runner possono utilizzare questi argomenti con il loro `run-suite` comando per specificare il seguente comportamento di esecuzione del test:

- Esegui tutti i test all'interno dei gruppi di test specificati.
- Esegui una selezione di test all'interno di un gruppo di test specificato.

Per supportare questi argomenti, la macchina a stati della suite di test deve includere un set specifico `RunTask` di `Choice` stati nella macchina a stati. Se non utilizzate una macchina a stati

personalizzata, la macchina a stati IDT predefinita include gli stati richiesti e non è necessario intraprendere ulteriori azioni. Tuttavia, se utilizzate una macchina a stati personalizzata, usatela [Esempio di macchina a stati: esegue gruppi di test selezionati dall'utente](#) come esempio per aggiungere gli stati richiesti nella vostra macchina a stati.

Per ulteriori informazioni sui comandi CLI IDT, vedere. [Esegui il debug ed esegui suite di test personalizzate](#)

Scrivere registri degli eventi

Durante l'esecuzione del test, invii dati `stdout` e `stderr` scrivi registri degli eventi e messaggi di errore nella console. Per informazioni sul formato dei messaggi della console, vedere [Formato dei messaggi della console](#).

Quando IDT termina l'esecuzione della suite di test, queste informazioni sono disponibili anche nel `test_manager.log` file che si trova nella `<devicetester-extract-location>/results/<execution-id>/logs` cartella.

È possibile configurare ogni test case in modo che scriva i log dell'esecuzione del test, inclusi i log del dispositivo sottoposto a test, nel `<group-id>_<test-id>` file che si trova nella cartella. `<device-tester-extract-location>/results/<execution-id>/logs` A tale scopo, recuperate il percorso del file di registro dal contesto IDT con la `testData.logFilePath` query, create un file in quel percorso e scrivete il contenuto desiderato. IDT aggiorna automaticamente il percorso in base al test case in esecuzione. Se si sceglie di non creare il file di registro per un test case, non viene generato alcun file per quel test case.

È inoltre possibile configurare il file eseguibile di testo per creare file di registro aggiuntivi, se necessario, nella `<device-tester-extract-location>/logs` cartella. Ti consigliamo di specificare prefissi univoci per i nomi dei file di registro in modo che i file non vengano sovrascritti.

Segnala i risultati a IDT

IDT scrive i risultati dei test nei file `awsiotdevicetester_report.xml` e `suite-name_report.xml`. Questi file di report si trovano in `<device-tester-extract-location>/results/<execution-id>/`. Entrambi i report acquisiscono i risultati dell'esecuzione della suite di test. Per ulteriori informazioni sugli schemi utilizzati da IDT per questi report, vedere [Esamina i risultati e i registri dei test IDT](#)

Per compilare il contenuto del `suite-name_report.xml` file, è necessario utilizzare il `SendResult` comando per riportare i risultati dei test a IDT prima del termine dell'esecuzione del test. Se IDT non è in grado di individuare i risultati di un test, genera un errore per il test case. Il seguente estratto di Python mostra i comandi per inviare il risultato di un test a IDT:

```
request-variable = SendResultRequest(TestResult(result))
client.send_result(request-variable)
```

Se non riporti i risultati tramite l'API, IDT cerca i risultati dei test nella cartella test artifacts. Il percorso di questa cartella viene memorizzato nel `testData.testArtifactsPath` file nel contesto IDT. In questa cartella, IDT utilizza il primo file XML in ordine alfabetico che individua come risultato del test.

Se la logica del test produce risultati JUnit XML, è possibile scrivere i risultati del test in un file XML nella cartella artifacts per fornire i risultati direttamente a IDT anziché analizzarli e quindi utilizzare l'API per inviarli a IDT.

Se utilizzate questo metodo, assicuratevi che la logica del test riassume accuratamente i risultati del test e formatti il file dei risultati nello stesso formato del file. `suite-name_report.xml` IDT non esegue alcuna convalida dei dati forniti, con le seguenti eccezioni:

- IDT ignora tutte le proprietà del tag. `testsuites` Al contrario, calcola le proprietà del tag in base ai risultati di altri gruppi di test riportati.
- All'interno `testsuites` deve esistere almeno un `testsuite` tag.

Poiché IDT utilizza la stessa cartella Artifacts per tutti i casi di test e non elimina i file dei risultati tra le esecuzioni dei test, questo metodo potrebbe anche portare a segnalazioni errate se IDT legge il file errato. Si consiglia di utilizzare lo stesso nome per il file dei risultati XML generato in tutti i casi di test per sovrascrivere i risultati di ogni test case e assicurarsi che siano disponibili i risultati corretti per l'uso da IDT. Sebbene nella suite di test sia possibile utilizzare un approccio misto alla reportistica, ovvero utilizzare un file di risultati XML per alcuni casi di test e inviare i risultati tramite l'API per altri, non consigliamo questo approccio.

Specificate il comportamento di uscita

Configura il tuo eseguibile testuale in modo che esca sempre con un codice di uscita pari a 0, anche se un test case riporta un errore o un errore. Utilizza codici di uscita diversi da zero solo per indicare che un test case non è stato eseguito o se l'eseguibile del test case non è in grado di comunicare

alcun risultato a IDT. Quando IDT riceve un codice di uscita diverso da zero, indica che il test case ha riscontrato un errore che ne ha impedito l'esecuzione.

IDT potrebbe richiedere o aspettarsi che un test case smetta di funzionare prima del termine nei seguenti eventi. Utilizzate queste informazioni per configurare l'eseguibile del test case in modo da rilevare ciascuno di questi eventi dal test case:

Timeout

Si verifica quando un test case viene eseguito per un periodo più lungo del valore di timeout specificato nel `test.json` file. Se il test runner ha utilizzato l'`timeout-multiplier` argomento per specificare un moltiplicatore di timeout, IDT calcola il valore di timeout con il moltiplicatore.

Per rilevare questo evento, utilizzate la variabile di ambiente `IDT_TEST_TIMEOUT`. Quando un test runner avvia un test, IDT imposta il valore della variabile di ambiente `IDT_TEST_TIMEOUT` sul valore di timeout calcolato (in secondi) e passa la variabile all'eseguibile del test case. È possibile leggere il valore della variabile per impostare un timer appropriato.

Interrompere

Si verifica quando il test runner interrompe IDT. Ad esempio, premendo `Ctrl+C`.

Poiché i terminali propagano i segnali a tutti i processi secondari, nei casi di test è sufficiente configurare un gestore di segnali per rilevare i segnali di interruzione.

In alternativa, puoi interrogare periodicamente l'API per verificare il valore del `CancellationRequested` booleano nella risposta dell'API. `PollForNotifications`
Quando IDT riceve un segnale di interruzione, imposta il valore del valore booleano su `CancellationRequested true`.

Smettila al primo errore

Si verifica quando un test case in esecuzione in parallelo al test case corrente ha esito negativo e il test runner ha utilizzato l'`stop-on-first-failure` argomento per specificare che IDT deve interrompersi in caso di errore.

Per rilevare questo evento, è possibile interrogare periodicamente l'API per verificare il valore del valore `CancellationRequested` booleano nella risposta dell'API. `PollForNotifications`
Quando IDT rileva un errore ed è configurato per interrompersi al primo errore, imposta il valore del valore booleano su `CancellationRequested true`.

Quando si verifica uno di questi eventi, IDT attende 5 minuti per terminare l'esecuzione di tutti i test case attualmente in esecuzione. Se tutti i casi di test in esecuzione non si chiudono entro 5 minuti, IDT impone l'interruzione di ciascuno dei relativi processi. Se IDT non ha ricevuto i risultati dei test prima della fine dei processi, contrassegnerà i casi di test come scaduti. Come best practice, dovrete assicurarvi che i test case eseguano le seguenti azioni quando si verificano uno degli eventi:

1. Interrompi l'esecuzione della normale logica di test.
2. Pulisci tutte le risorse temporanee, come gli artefatti di test sul dispositivo sottoposto a test.
3. Segnala un risultato del test a IDT, ad esempio un errore o un fallimento del test.
4. Esci.

Usa il contesto IDT

Quando IDT esegue una suite di test, la suite di test può accedere a un set di dati che possono essere utilizzati per determinare la modalità di esecuzione di ciascun test. Questi dati sono chiamati contesto IDT. Ad esempio, la configurazione dei dati utente fornita dai test runner in un `userdata.json` file viene resa disponibile alle suite di test nel contesto IDT.

Il contesto IDT può essere considerato un documento JSON di sola lettura. Le suite di test possono recuperare dati e scrivere dati nel contesto utilizzando tipi di dati JSON standard come oggetti, matrici, numeri e così via.

Schema contestuale

Il contesto IDT utilizza il seguente formato:

```
{
  "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

Informazioni dal [config.jsonfile](#). Il config campo contiene anche il seguente campo aggiuntivo:

`config.timeoutMultiplier`

Il moltiplicatore per l'eventuale valore di timeout utilizzato dalla suite di test. Questo valore è specificato dal test runner dalla CLI IDT. Il valore predefinito è 1.

device

Informazioni sul dispositivo selezionato per l'esecuzione del test. Queste informazioni sono equivalenti all'elemento dell'`devicesarray` nel [device.jsonfile](#) per il dispositivo selezionato.

devicePool

Informazioni sul pool di dispositivi selezionato per l'esecuzione del test. Queste informazioni sono equivalenti all'elemento dell'array del pool di dispositivi di primo livello definito nel `device.json` file per il pool di dispositivi selezionato.

resource

Informazioni sui dispositivi di risorse contenute nel `resource.json` file.

`resource.devices`

Queste informazioni sono equivalenti all'`devicesarray` definito nel `resource.json` file. Ogni `devices` elemento include il seguente campo aggiuntivo:

`resource.device.name`

Il nome del dispositivo di risorse. Questo valore è impostato sul `requiredResource.name` valore del `test.json` file.

`testData.awsCredentials`

Le AWS credenziali utilizzate dal test per connettersi al AWS cloud. Queste informazioni sono ottenute dal `config.json` file.

`testData.logFilePath`

Il percorso del file di registro in cui il test case scrive i messaggi di registro. La suite di test crea questo file se non esiste.

`userData`

Informazioni fornite dal test runner nel [userdata.jsonfile](#).

Accedere ai dati nel contesto

Puoi interrogare il contesto utilizzando la JSONPath notazione dei tuoi file JSON e del tuo file eseguibile di testo con `and`. `GetContextValue` `GetContextString` APIs La sintassi con cui le JSONPath stringhe accedono al contesto IDT varia come segue:

- In `suite.json` e `test.json`, si usa. `{{query}}` Cioè, non usate l'elemento root `$`. per iniziare l'espressione.
- In `statemachine.json`, si usa `{{$.query}}`.
- Nei comandi API, si utilizza *query* o `{{$.query}}`, a seconda del comando. Per ulteriori informazioni, consultate la documentazione in linea in. SDKs

La tabella seguente descrive gli operatori di un' JSONPathespressione tipica:

Operatore	Descrizione
\$	Lelemento radice. Poiché il valore di contesto di primo livello per IDT è un oggetto, viene in genere utilizzato \$. per avviare le query.
.childName	Accede all'elemento secondario con il nome childName di un oggetto. Se applicato a un array, restituisce un nuovo array con questo operatore applicato a ciascun elemento. Il nome dell'elemento distingue tra maiuscole e minuscole. Ad esempio, la query per accedere al awsRegion valore nell'configoggetto è\$.config.awsRegion .
[start:end]	Filtra gli elementi da una matrice, recuperando gli elementi che iniziano dall'startindice e risalgono all'endindice, entrambi inclusi.
[index1, index2, ... , indexN]	Filtra gli elementi da un array, recuperando gli elementi solo dagli indici specificati.
[?(expr)]	Filtra gli elementi di un array utilizzando l'espressione. expr Questa espressione deve restituire un valore booleano.

Per creare espressioni di filtro, utilizzate la seguente sintassi:

```
<jsonpath> | <value> operator <jsonpath> | <value>
```

In questa sintassi:

- jsonpathè un programma JSONPath che utilizza la sintassi JSON standard.
- valueè qualsiasi valore personalizzato che utilizza la sintassi JSON standard.
- operatorè uno dei seguenti operatori:
 - <(Meno di)
 - <=(Minore o uguale a)

- `==(Uguale a)`

Se il valore JSONPath o nell'espressione è un array, un valore booleano o un valore di oggetto, questo è l'unico operatore binario supportato che è possibile utilizzare.

- `>=(Maggiore o uguale a)`
- `>(Maggiore di)`
- `~=`(Corrispondenza delle espressioni regolari). Per utilizzare questo operatore in un'espressione di filtro, il valore JSONPath o sul lato sinistro dell'espressione deve restituire una stringa e il lato destro deve essere un valore di modello che segue la [RE2sintassi](#).

È possibile utilizzare JSONPath le interrogazioni nel formato `{{query}}` come stringhe segnaposto all'interno dei `environmentVariables` campi `args` e nei campi `test.json` dei file.

`environmentVariables.suite.json` IDT esegue una ricerca contestuale e popola i campi con il valore valutato della query. Ad esempio, nel `suite.json` file, è possibile utilizzare stringhe segnaposto per specificare i valori delle variabili di ambiente che cambiano con ogni test case e IDT popolerà le variabili di ambiente con il valore corretto per ogni test case. Tuttavia, quando utilizzate stringhe segnaposto nei `suite.json` file `test.json` e, alle vostre query valgono le seguenti considerazioni:

- È necessario che ogni occorrenza della `devicePool` chiave nella query sia scritta interamente in lettere minuscole. Cioè, usa `devicepool` invece.
- Per gli array, è possibile utilizzare solo matrici di stringhe. Inoltre, gli array utilizzano un formato non standard. `item1, item2, ..., itemN` Se l'array contiene solo un elemento, viene serializzato come `item`, rendendolo indistinguibile da un campo di stringhe.
- Non è possibile utilizzare i segnaposto per recuperare oggetti dal contesto.

Alla luce di queste considerazioni, consigliamo, quando possibile, di utilizzare l'API per accedere al contesto nella logica di test anziché le stringhe segnaposto nei file e nei file. `test.json` `suite.json` Tuttavia, in alcuni casi potrebbe essere più comodo utilizzare i JSONPath segnaposto per recuperare singole stringhe da impostare come variabili di ambiente.

Configurare le impostazioni per i test runner

Per eseguire suite di test personalizzate, i test runner devono configurare le proprie impostazioni in base alla suite di test che desiderano eseguire. Le impostazioni vengono specificate in base ai modelli di file di configurazione JSON presenti nella `<device-tester-extract-location>/`

configs/ cartella. Se necessario, i test runner devono anche impostare AWS le credenziali che IDT utilizzerà per connettersi al cloud. AWS

In qualità di scrittore di test, dovrai configurare questi file per eseguire il [debug](#) della tua suite di test. È necessario fornire istruzioni ai test runner in modo che possano configurare le seguenti impostazioni in base alle esigenze per eseguire le suite di test.

Configura dispositivo.json

Il `device.json` file contiene informazioni sui dispositivi su cui vengono eseguiti i test (ad esempio, indirizzo IP, informazioni di accesso, sistema operativo e architettura della CPU).

I test runner possono fornire queste informazioni utilizzando il seguente `device.json` file modello che si trova nella `<device-tester-extract-location>/configs/` cartella.

```
[
  {
    "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>",
}

]

}
```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

id

Un ID alfanumerico definito dall'utente che identifica in modo univoco una raccolta di dispositivi denominata un pool di dispositivi. I dispositivi che appartengono a un pool devono avere lo stesso hardware. Durante l'esecuzione di una suite di test, i dispositivi del pool vengono utilizzati per parallelizzare il carico di lavoro. Più dispositivi vengono utilizzati per eseguire diversi test.

sku

Un valore alfanumerico che identifica in modo univoco il dispositivo sottoposto a test. Lo SKU viene utilizzato per tracciare i dispositivi qualificati.

Note

Se desideri inserire la tua scheda nel AWS Partner Device Catalog, lo SKU che specifichi qui deve corrispondere allo SKU che utilizzi nella procedura di pubblicazione.

features

Facoltativo. Un array contenente le caratteristiche supportate del dispositivo. Le funzionalità del dispositivo sono valori definiti dall'utente che configuri nella tua suite di test. È necessario fornire ai test runner informazioni sui nomi e sui valori delle funzionalità da includere nel `device.json` file. Ad esempio, se desiderate testare un dispositivo che funge da server MQTT per altri dispositivi, potete configurare la logica di test per convalidare livelli supportati specifici per una funzionalità denominata. `MQTT_QOS` I test runner forniscono questo nome di funzionalità e impostano il valore della funzionalità sui livelli QOS supportati dal proprio dispositivo. È possibile recuperare le informazioni fornite dal contesto [IDT con la `devicePool.features` query o dal contesto](#) della [macchina a stati](#) con la query. `pool.features`

`features.name`

Il nome della funzionalità.

`features.value`

I valori delle funzionalità supportate.

`features.configs`

Impostazioni di configurazione, se necessarie, per la funzionalità.

`features.config.name`

Il nome dell'impostazione di configurazione.

`features.config.value`

I valori di impostazione supportati.

devices

Una serie di dispositivi nel pool da testare. È richiesto almeno un dispositivo.

`devices.id`

Un identificativo univoco definito dall'utente del dispositivo sottoposto a test.

`connectivity.protocol`

Il protocollo di comunicazione utilizzato per comunicare con questo dispositivo. Ogni dispositivo in un pool deve utilizzare lo stesso protocollo.

Attualmente, gli unici valori supportati sono `ssh` e `uart` per i dispositivi fisici e `docker` per i contenitori Docker.

`connectivity.ip`

L'indirizzo IP del dispositivo sottoposto a test.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `ssh`.

`connectivity.port`

Facoltativo. Il numero di porta da utilizzare per le connessioni SSH.

Il valore predefinito è 22.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `ssh`.

`connectivity.auth`

Informazioni di autenticazione per la connessione.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `ssh`.

`connectivity.auth.method`

Il metodo di autorizzazione utilizzato per accedere a un dispositivo con un determinato protocollo di connettività.

I valori supportati sono:

- `pki`
- `password`

`connectivity.auth.credentials`

Le credenziali utilizzate per l'autenticazione.

`connectivity.auth.credentials.password`

La password utilizzata per l'accesso al dispositivo da testare.

Questo valore si applica solo se `connectivity.auth.method` è impostato su `password`.

`connectivity.auth.credentials.privKeyPath`

Il percorso completo alla chiave privata utilizzata per accedere al dispositivo sottoposto a test.

Questo valore si applica solo se `connectivity.auth.method` è impostato su `pki`.

`connectivity.auth.credentials.user`

Il nome utente per l'accesso al dispositivo sottoposto a test.

`connectivity.serialPort`

Facoltativo. La porta seriale a cui è collegato il dispositivo.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `uart`.

`connectivity.containerId`

L'ID contenitore o il nome del contenitore Docker in fase di test.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `docker`.

`connectivity.containerUser`

Facoltativo. Il nome da utente a utente all'interno del contenitore. Il valore predefinito è l'utente fornito nel Dockerfile.

Il valore predefinito è 22.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `docker`.

Note

Per verificare se i test runner configurano la connessione errata del dispositivo per un test, è possibile recuperarlo `pool.Devices[0].Connectivity.Protocol` dal contesto della macchina a stati e confrontarlo con il valore previsto in uno `Choice` stato. Se viene utilizzato un protocollo errato, stampa un messaggio utilizzando lo `LogMessage` stato e passa allo `Fail` stato.

In alternativa, è possibile utilizzare il codice di gestione degli errori per segnalare un errore di test per tipi di dispositivi errati.

(Facoltativo) Configura `userdata.json`

Il `userdata.json` file contiene tutte le informazioni aggiuntive richieste da una suite di test ma non specificate nel file `device.json`. Il formato di questo file dipende dal [userdata_scheme.jsonfile](#) definito nella suite di test. Se sei uno scrittore di test, assicurati di fornire queste informazioni agli utenti che eseguiranno le suite di test che scrivi.

(Facoltativo) Configura resource.json

Il `resource.json` file contiene informazioni su tutti i dispositivi che verranno utilizzati come dispositivi di risorse. I dispositivi di risorse sono dispositivi necessari per testare determinate funzionalità di un dispositivo sottoposto a test. Ad esempio, per testare la funzionalità Bluetooth di un dispositivo, è possibile utilizzare un dispositivo di risorse per verificare che il dispositivo sia in grado di connettersi correttamente ad esso. I dispositivi di risorse sono opzionali e puoi richiedere tutti i dispositivi di risorse di cui hai bisogno. In qualità di autore del test, utilizzi il [file test.json](#) per definire le funzionalità dei dispositivi di risorse necessarie per un test. I test runner utilizzano quindi il `resource.json` file per fornire un pool di dispositivi di risorse dotati delle funzionalità richieste. Assicurati di fornire queste informazioni agli utenti che eseguiranno le suite di test che scrivi.

I test runner possono fornire queste informazioni utilizzando il seguente `resource.json` file modello che si trova nella `<device-tester-extract-location>/configs/` cartella.

```
[
  {
    "id": "<pool-id>",
    "features": [
      {
        "name": "<feature-name>",
        "version": "<feature-value>",
        "jobSlots": <job-slots>
      }
    ],
    "devices": [
      {
        "id": "<device-id>",
        "connectivity": {
          "protocol": "ssh | uart | docker",
          // ssh
          "ip": "<ip-address>",
          "port": <port-number>,
          "auth": {
            "method": "pki | password",
            "credentials": {
              "user": "<user-name>",
              // pki
              "privKeyPath": "/path/to/private/key",

              // password
              "password": "<password>",
```

```

        }
    },

    // uart
    "serialPort": "<serial-port>",

    // docker
    "containerId": "<container-id>",
    "containerUser": "<container-user-name>",
  }
}
]
]

```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

id

Un ID alfanumerico definito dall'utente che identifica in modo univoco una raccolta di dispositivi denominata un pool di dispositivi. I dispositivi che appartengono a un pool devono avere lo stesso hardware. Durante l'esecuzione di una suite di test, i dispositivi del pool vengono utilizzati per parallelizzare il carico di lavoro. Più dispositivi vengono utilizzati per eseguire diversi test.

features

Facoltativo. Un array contenente le caratteristiche supportate del dispositivo. Le informazioni richieste in questo campo sono definite nei [file test.json](#) nella suite di test e determinano quali test eseguire e come eseguirli. Se la suite di test non richiede alcuna funzionalità, questo campo non è obbligatorio.

features.name

Il nome della funzionalità.

features.version

La versione della funzionalità.

features.jobSlots

Impostazione per indicare quanti test possono utilizzare il dispositivo contemporaneamente. Il valore predefinito è 1.

devices

Una serie di dispositivi nel pool da testare. È richiesto almeno un dispositivo.

`devices.id`

Un identificativo univoco definito dall'utente del dispositivo sottoposto a test.

`connectivity.protocol`

Il protocollo di comunicazione utilizzato per comunicare con questo dispositivo. Ogni dispositivo in un pool deve utilizzare lo stesso protocollo.

Attualmente, gli unici valori supportati sono `ssh` e `uart` per i dispositivi fisici e `docker` per i contenitori Docker.

`connectivity.ip`

L'indirizzo IP del dispositivo sottoposto a test.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `ssh`.

`connectivity.port`

Facoltativo. Il numero di porta da utilizzare per le connessioni SSH.

Il valore predefinito è 22.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `ssh`.

`connectivity.auth`

Informazioni di autenticazione per la connessione.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `ssh`.

`connectivity.auth.method`

Il metodo di autorizzazione utilizzato per accedere a un dispositivo con un determinato protocollo di connettività.

I valori supportati sono:

- `pki`
- `password`

`connectivity.auth.credentials`

Le credenziali utilizzate per l'autenticazione.

`connectivity.auth.credentials.password`

La password utilizzata per l'accesso al dispositivo da testare.

Questo valore si applica solo se `connectivity.auth.method` è impostato su `password`.

`connectivity.auth.credentials.privKeyPath`

Il percorso completo alla chiave privata utilizzata per accedere al dispositivo sottoposto a test.

Questo valore si applica solo se `connectivity.auth.method` è impostato su `pki`.

`connectivity.auth.credentials.user`

Il nome utente per l'accesso al dispositivo sottoposto a test.

`connectivity.serialPort`

Facoltativo. La porta seriale a cui è collegato il dispositivo.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `uart`.

`connectivity.containerId`

L'ID contenitore o il nome del contenitore Docker in fase di test.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `docker`.

`connectivity.containerUser`

Facoltativo. Il nome da utente a utente all'interno del contenitore. Il valore predefinito è l'utente fornito nel Dockerfile.

Il valore predefinito è 22.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `docker`.

(Facoltativo) Configura `config.json`

Il `config.json` file contiene informazioni di configurazione per IDT. In genere, i test runner non avranno bisogno di modificare questo file se non per fornire le proprie credenziali AWS utente per IDT e, facoltativamente, una regione. AWS Se vengono fornite AWS le credenziali con le autorizzazioni

richieste, AWS IoT Device Tester raccoglie e invia le metriche di utilizzo a. AWS Si tratta di una funzionalità opzionale e viene utilizzata per migliorare la funzionalità IDT. Per ulteriori informazioni, consulta [Metriche di utilizzo IDT](#).

I test runner possono configurare le proprie AWS credenziali in uno dei seguenti modi:

- File di credenziali

IDT usa lo stesso file delle credenziali di AWS CLI. Per ulteriori informazioni, consulta l'argomento relativo ai [file di configurazione e delle credenziali](#).

La posizione del file delle credenziali varia in base al sistema operativo in uso:

- macOS, Linux: `~/.aws/credentials`
- Windows: `C:\Users\UserName\.aws\credentials`
- Variabili di ambiente

Le variabili di ambiente sono variabili gestite dal sistema operativo e utilizzate dai comandi di sistema. Le variabili definite durante una sessione SSH non sono disponibili dopo la chiusura della sessione. IDT può utilizzare le variabili di `AWS_SECRET_ACCESS_KEY` ambiente `AWS_ACCESS_KEY_ID` e per memorizzare le credenziali AWS

Per impostare queste variabili su Linux, macOS o Unix, utilizza export:

```
export AWS_ACCESS_KEY_ID=<your_access_key_id>
export AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

Per impostare queste variabili su Windows, utilizza set:

```
set AWS_ACCESS_KEY_ID=<your_access_key_id>
set AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

Per configurare AWS le credenziali per IDT, i test runner modificano la auth sezione del `config.json` file che si trova nella cartella. `<device-tester-extract-location>/configs/`

```
{
  "log": {
    "location": "logs"
  },
  "configFiles": {
```

```
    "root": "configs",
    "device": "configs/device.json"
  },
  "testPath": "tests",
  "reportPath": "results",
  "awsRegion": "<region>",
  "auth": {
    "method": "file | environment",
    "credentials": {
      "profile": "<profile-name>"
    }
  }
}
```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

Note

Tutti i percorsi di questo file sono definiti in relazione a. *<device-tester-extract-location>*

log.location

Il percorso della cartella dei registri in. *<device-tester-extract-location>*

configFiles.root

Il percorso della cartella che contiene i file di configurazione.

configFiles.device

Il percorso del device.json file.

testPath

Il percorso della cartella che contiene le suite di test.

reportPath

Il percorso della cartella che conterrà i risultati dei test dopo che IDT avrà eseguito una suite di test.

awsRegion

Facoltativo. La AWS regione che verranno utilizzate dalle suite di test. Se non è impostata, le suite di test utilizzeranno la regione predefinita specificata in ciascuna suite di test.

auth.method

Il metodo utilizzato da IDT per recuperare AWS le credenziali. I valori supportati sono `file` il recupero delle credenziali da un file di credenziali e il recupero delle credenziali utilizzando le variabili `environment` di ambiente.

auth.credentials.profile

Il profilo delle credenziali da utilizzare dal file delle credenziali. Questa proprietà si applica solo se `auth.method` è impostata su `file`.

Esegui il debug ed esegui suite di test personalizzate

Dopo aver impostato la [configurazione richiesta](#), IDT può eseguire la suite di test. Il tempo di esecuzione della suite di test completa dipende dall'hardware e dalla composizione della suite di test. Per riferimento, sono necessari circa 30 minuti per completare l'intera suite di test di AWS IoT Greengrass qualificazione su un Raspberry Pi 3B.

Durante la scrittura della suite di test, è possibile utilizzare IDT per eseguire la suite di test in modalità debug per controllare il codice prima di eseguirlo o fornirlo ai test runner.

Esegui IDT in modalità di debug

Poiché le suite di test dipendono da IDT per interagire con i dispositivi, fornire il contesto e ricevere risultati, non è possibile semplicemente eseguire il debug delle suite di test in un IDE senza alcuna interazione IDT. A tale scopo, la CLI IDT fornisce `debug-test-suite` il comando che consente di eseguire IDT in modalità di debug. Eseguite il comando seguente per visualizzare le opzioni disponibili per: `debug-test-suite`

```
devicetester_[linux | mac | win_x86-64] debug-test-suite -h
```

Quando si esegue IDT in modalità debug, IDT non avvia effettivamente la suite di test né esegue la macchina a stati; interagisce invece con l'IDE per rispondere alle richieste effettuate dalla suite di test in esecuzione nell'IDE e stampa i log sulla console. IDT non scade e attende di uscire finché non viene interrotto manualmente. In modalità debug, IDT inoltre non esegue la macchina a stati e non genererà alcun file di report. Per eseguire il debug della suite di test, è necessario utilizzare l'IDE

per fornire alcune informazioni che IDT di solito ottiene dai file JSON di configurazione. Assicurati di fornire le seguenti informazioni:

- Variabili di ambiente e argomenti per ogni test. IDT non leggerà queste informazioni da `test.json` o `suite.json`.
- Argomenti per selezionare i dispositivi di risorse. IDT non leggerà queste informazioni da `test.json`.

Per eseguire il debug delle tue suite di test, completa i seguenti passaggi:

1. Crea i file di configurazione delle impostazioni necessari per eseguire la suite di test. Ad esempio, se la tua suite di test richiede `device.json`, `resource.json`, `user_data.json`, e, assicurati di configurarli tutti secondo necessità.
2. Esegui il comando seguente per mettere IDT in modalità debug e selezionare tutti i dispositivi necessari per eseguire il test.

```
devicetester_[linux | mac | win_x86-64] debug-test-suite [options]
```

Dopo aver eseguito questo comando, IDT attende le richieste dalla suite di test e quindi risponde ad esse. IDT genera anche le variabili di ambiente necessarie per il processo di elaborazione dei casi per IDT Client SDK.

3. Nel tuo IDE, usa la debug configurazione `run` o per effettuare le seguenti operazioni:
 - a. Imposta i valori delle variabili di ambiente generate da IDT.
 - b. Imposta il valore di tutte le variabili o gli argomenti di ambiente che hai specificato nel file `test.json` e `suite.json`.
 - c. Imposta i punti di interruzione in base alle esigenze.
4. Esegui la suite di test nel tuo IDE.

Puoi eseguire il debug e rieseguire la suite di test tutte le volte che è necessario. IDT non scade in modalità di debug.

5. Dopo aver completato il debug, interrompi IDT per uscire dalla modalità di debug.

Comandi IDT CLI per eseguire test

La sezione seguente descrive i comandi IDT CLI:

IDT v4.0.0

help

Elenca le informazioni sul comando specificato.

list-groups

Elenca i gruppi in una determinata suite di test.

list-suites

Elenca le suite di test disponibili.

list-supported-products

Elenca i prodotti supportati per la versione di IDT in uso, in questo caso AWS IoT Greengrass le versioni, e le versioni della suite di test di AWS IoT Greengrass qualificazione disponibili per la versione IDT corrente.

list-test-cases

Elenca i casi di test in un determinato gruppo di test. È supportata la seguente opzione:

- `group-id`. Il gruppo di test da cercare. Questa opzione è obbligatoria e deve specificare un singolo gruppo.

run-suite

Esegue una suite di test in un determinato pool di dispositivi. Di seguito sono riportate alcune opzioni di uso comune:

- `suite-id`. La versione della suite di test da eseguire. Se non specificato, IDT utilizza la versione più recente nella cartella `tests`.
- `group-id`. I gruppi di test da eseguire, sotto forma di elenco separato da virgole. Se non specificato, IDT esegue tutti i gruppi di test nella suite di test.
- `test-id`. I casi di test da eseguire, come elenco separato da virgole. Quando specificato, `group-id` deve specificare un singolo gruppo.
- `pool-id`. Il pool di dispositivi da testare. I test runner devono specificare un pool se hanno più pool di dispositivi definiti nel `device.json` file.
- `timeout-multiplier`. Configura IDT per modificare il timeout di esecuzione del test specificato nel `test.json` file per un test con un moltiplicatore definito dall'utente.

- `stop-on-first-failure`. Configura IDT per interrompere l'esecuzione al primo errore. Questa opzione deve essere utilizzata con `group-id` per eseguire il debug dei gruppi di test specificati.
- `userdata`. Imposta il file che contiene le informazioni sui dati utente necessarie per eseguire la suite di test. Questo è richiesto solo se `userdataRequired` è impostato su `true` nel `suite.json` file della suite di test.

Per ulteriori informazioni sulle opzioni `run-suite`, utilizzare l'opzione `help`:

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

debug-test-suite

Esegui la suite di test in modalità debug. Per ulteriori informazioni, consulta [Esegui IDT in modalità di debug](#).

Esamina i risultati e i registri dei test IDT

Questa sezione descrive il formato in cui IDT genera i log della console e i report dei test.

Formato dei messaggi della console

AWS IoT Device Tester utilizza un formato standard per stampare i messaggi sulla console quando avvia una suite di test. Il seguente estratto mostra un esempio di messaggio di console generato da IDT.

```
time="2000-01-02T03:04:05-07:00" level=info msg=Using suite: MyTestSuite_1.0.0  
executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

La maggior parte dei messaggi della console è composta dai seguenti campi:

time

Un timestamp ISO 8601 completo per l'evento registrato.

level

Il livello del messaggio per l'evento registrato. In genere, il livello del messaggio registrato è uno dei `info`, `warn`, o `error`. IDT emette un `panic` messaggio `fatal` or se rileva un evento previsto che ne causa la chiusura anticipata.

msg

Il messaggio registrato.

executionId

Una stringa ID univoca per il processo IDT corrente. Questo ID viene utilizzato per distinguere le singole esecuzioni IDT.

I messaggi della console generati da una suite di test forniscono informazioni aggiuntive sul dispositivo sottoposto a test e sulla suite di test, sul gruppo di test e sui casi di test eseguiti da IDT. Il seguente estratto mostra un esempio di messaggio di console generato da una suite di test.

```
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
```

La parte specifica del messaggio della console di test contiene i seguenti campi:

suiteId

Il nome della suite di test attualmente in esecuzione.

groupId

L'ID del gruppo di test attualmente in esecuzione.

testCaseId

L'ID del test case attualmente in esecuzione.

deviceId

Un ID del dispositivo sottoposto a test utilizzato dal test case corrente.

Per stampare un riepilogo del test sulla console quando un IDT termina l'esecuzione di un test, è necessario includere uno [Report stato](#) nella macchina a stati. Il riepilogo del test contiene informazioni sulla suite di test, i risultati dei test per ogni gruppo eseguito e le posizioni dei log e dei file di report generati. L'esempio seguente mostra un messaggio di riepilogo del test.

```
===== 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 Schema di report di Device Tester

`awsiotdevicetester_report.xml` è un rapporto firmato che contiene le seguenti informazioni:

- La versione di IDT.
- La versione della suite di test.
- La firma del rapporto e la chiave utilizzate per firmare il rapporto.
- Lo SKU del dispositivo e il nome del pool di dispositivi specificati nel `device.json` file.
- La versione del prodotto e le funzionalità del dispositivo testate.
- Il riepilogo aggregato dei risultati dei test. Queste informazioni sono le stesse contenute nel `suite-name_report.xml` file.

```

<apnreport>
  <awsiotdevicetesterversion>idt-version</awsiotdevicetesterversion>
  <testsuiteversion>test-suite-version</testsuiteversion>
  <signature>signature</signature>
  <keyname>keyname</keyname>
  <session>
    <testsession>execution-id</testsession>
    <starttime>start-time</starttime>

```

```

        <endtime>end-time</endtime>
    </session>
    <awsproduct>
        <name>product-name</name>
        <version>product-version</version>
        <features>
            <feature name="<feature-name>" value="supported | not-supported | <feature-
value>" type="optional | required"/>
        </features>
    </awsproduct>
    <device>
        <sku>device-sku</sku>
        <name>device-name</name>
        <features>
            <feature name="<feature-name>" value="<feature-value>"/>
        </features>
        <executionMethod>ssh | uart | docker</executionMethod>
    </device>
    <devenvironment>
        <os name="<os-name>"/>
    </devenvironment>
    <report>
        <suite-name-report-contents>
    </report>
</apnreport>

```

Il file `awsiotdevicetester_report.xml` contiene un tag `<awsproduct>` con le informazioni relative al prodotto sottoposto a test e le caratteristiche del prodotto che sono state convalidate dopo l'esecuzione di una suite di test.

Attributi utilizzati nel tag `<awsproduct>`

name

Il nome del prodotto sottoposto a test.

version

La versione del prodotto sottoposto a test.

features

Le caratteristiche convalidate. Le funzionalità contrassegnate come `required` sono necessarie per consentire alla suite di test di convalidare il dispositivo. Il seguente

frammento di codice mostra come questa informazione viene visualizzata nel file `awsiotdevicetester_report.xml`.

```
<feature name="ssh" value="supported" type="required"></feature>
```

Le funzionalità contrassegnate come non optional sono richieste per la convalida. I seguenti snippet mostrano caratteristiche facoltative.

```
<feature name="hsi" value="supported" type="optional"></feature>

<feature name="mqtt" value="not-supported" type="optional"></feature>
```

Schema di report della suite di test

Il `suite-name_Result.xml` rapporto è in [formato JUnit XML](#). Puoi eseguire l'integrazione in piattaforme di integrazione e distribuzione continue come [Jenkins](#), [Bambù](#) e così via. Il rapporto contiene un riepilogo aggregato dei risultati del test.

```
<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>
```

La sezione del rapporto in entrambe le sezioni `awsiotdevicetester_report.xml` o *suite-name_report.xml* elenca i test eseguiti e i risultati.

Il primo tag XML `<testsuites>` contiene il riepilogo dell'esecuzione dei test. Per esempio:

```
<testsuites name="MyTestSuite results" time="2299" tests="28" failures="0" errors="0"
disabled="0">
```

Attributi utilizzati nel tag **<testsuites>**

name

Il nome della suite di test.

time

Il tempo impiegato, in secondi, per eseguire la suite di test.

tests

Il numero di test eseguiti.

failures

Il numero di test eseguiti ma non superati.

errors

Il numero di test che IDT non è stato in grado di eseguire.

disabled

Questo attributo non è utilizzato e si può ignorare.

In caso di esiti negativi o errori nei test, puoi identificare il test non riuscito esaminando i tag XML `<testsuites>`. I tag XML `<testsuite>` all'interno del tag `<testsuites>` mostrano il riepilogo dei risultati dei test per un gruppo di test. Per esempio:

```
<testsuite name="combination" package="" tests="1" failures="0" time="161" disabled="0"
errors="0" skipped="0">
```

Il formato è simile al tag `<testsuites>`, ma con un attributo `skipped` che non viene utilizzato e che è possibile ignorare. All'interno di ogni tag XML `<testsuite>` ci sono tag `<testcase>` per ciascuno dei test eseguiti per un gruppo di test. Per esempio:

```
<testcase classname="Security Test" name="IP Change Tests" attempts="1"></testcase>>
```

Attributi utilizzati nel tag `<testcase>`

`name`

Il nome del test.

`attempts`

Il numero di volte che IDT ha eseguito il test.

Quando un test non riesce o si verifica un errore, i tag `<failure>` o `<error>` vengono aggiunti al tag `<testcase>` con informazioni per la risoluzione dei problemi. Per esempio:

```
<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>
```

Metriche di utilizzo IDT

Se fornisci AWS credenziali con le autorizzazioni richieste, AWS IoT Device Tester raccoglie e invia le metriche di utilizzo a. AWS Si tratta di una funzionalità opzionale e viene utilizzata per migliorare la funzionalità IDT. IDT raccoglie informazioni come le seguenti:

- L' Account AWS ID utilizzato per eseguire IDT
- I comandi IDT CLI utilizzati per eseguire i test
- La suite di test che viene eseguita
- Le suite di test nella *<device-tester-extract-location>* cartella

- Il numero di dispositivi configurati nel pool di dispositivi
- Nomi e tempi di esecuzione dei test case
- Informazioni sui risultati del test, ad esempio se i test sono stati superati, hanno avuto esito negativo, hanno riscontrato errori o sono stati ignorati
- Caratteristiche del prodotto testate
- Comportamento di uscita IDT, ad esempio uscite impreviste o anticipate

Tutte le informazioni inviate da IDT vengono inoltre registrate in un `metrics.log` file nella cartella. `<device-tester-extract-location>/results/<execution-id>/` È possibile visualizzare il file di registro per visualizzare le informazioni raccolte durante un'esecuzione di test. Questo file viene generato solo se si sceglie di raccogliere le metriche di utilizzo.

Per disabilitare la raccolta delle metriche, non è necessario intraprendere ulteriori azioni. Semplicemente non archiviate AWS le vostre credenziali e, se avete AWS credenziali memorizzate, non configurate il file `config.json` per accedervi.

Configura le tue credenziali AWS

Se non ne hai già uno Account AWS, devi [crearne uno](#). Se ne hai già uno Account AWS, devi semplicemente [configurare le autorizzazioni richieste per il](#) tuo account che consentano a IDT di inviare le metriche di utilizzo AWS a tuo nome.

Fase 1: Creare un Account AWS

In questo passaggio, crea e configura un Account AWS. Se ne hai già uno Account AWS, vai [a the section called "Fase 2: configurazione delle autorizzazioni per IDT"](#).

Iscriviti a un Account AWS

Se non ne hai uno Account AWS, completa i seguenti passaggi per crearne uno.

Per iscriverti a un Account AWS

1. Apri la <https://portal.aws.amazon.com/billing/registrazione>.
2. Segui le istruzioni online.

Parte della procedura di registrazione prevede la ricezione di una telefonata o di un messaggio di testo e l'immissione di un codice di verifica sulla tastiera del telefono.

Quando ti iscrivi a un Account AWS, Utente root dell'account AWS viene creato un. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWS nell'account. Come best practice di sicurezza, assegna l'accesso amministrativo a un utente e utilizza solo l'utente root per eseguire [attività che richiedono l'accesso di un utente root](#).

AWS ti invia un'email di conferma dopo il completamento della procedura di registrazione. In qualsiasi momento, puoi visualizzare l'attività corrente del tuo account e gestirlo accedendo a <https://aws.amazon.com/> e scegliendo Il mio account.

Crea un utente con accesso amministrativo

Dopo esserti registrato Account AWS, proteggi Utente root dell'account AWS AWS IAM Identity Center, abilita e crea un utente amministrativo in modo da non utilizzare l'utente root per le attività quotidiane.

Proteggi i tuoi Utente root dell'account AWS

1. Accedi [AWS Management Console](#) come proprietario dell'account scegliendo Utente root e inserendo il tuo indirizzo Account AWS email. Nella pagina successiva, inserisci la password.

Per informazioni sull'accesso utilizzando un utente root, consulta la pagina [Signing in as the root user](#) della Guida per l'utente di Accedi ad AWS .

2. Abilita l'autenticazione a più fattori (MFA) per l'utente root.

Per istruzioni, consulta [Abilitare un dispositivo MFA virtuale per l'utente Account AWS root \(console\)](#) nella Guida per l'utente IAM.

Crea un utente con accesso amministrativo

1. Abilita Centro identità IAM.

Per istruzioni, consulta [Abilitazione di AWS IAM Identity Center](#) nella Guida per l'utente di AWS IAM Identity Center .

2. In IAM Identity Center, assegna l'accesso amministrativo a un utente.

Per un tutorial sull'utilizzo di IAM Identity Center directory come fonte di identità, consulta [Configurare l'accesso utente con l'impostazione predefinita IAM Identity Center directory](#) nella Guida per l'AWS IAM Identity Center utente.

Accesso come utente amministratore

- Per accedere con l'utente IAM Identity Center, utilizza l'URL di accesso che è stato inviato al tuo indirizzo e-mail quando hai creato l'utente IAM Identity Center.

Per informazioni sull'accesso utilizzando un utente IAM Identity Center, consulta [AWS Accedere al portale di accesso](#) nella Guida per l'Accedi ad AWS utente.

Assegna l'accesso a ulteriori utenti

1. In IAM Identity Center, crea un set di autorizzazioni conforme alla best practice dell'applicazione di autorizzazioni con il privilegio minimo.

Segui le istruzioni riportate nella pagina [Creazione di un set di autorizzazioni](#) nella Guida per l'utente di AWS IAM Identity Center .

2. Assegna al gruppo prima gli utenti e poi l'accesso con autenticazione unica (Single Sign-On).

Per istruzioni, consulta [Aggiungere gruppi](#) nella Guida per l'utente di AWS IAM Identity Center .

Fase 2: configurazione delle autorizzazioni per IDT

In questo passaggio, configura le autorizzazioni utilizzate da IDT per eseguire test e raccogliere dati sull'utilizzo di IDT. Puoi utilizzare AWS Management Console or AWS Command Line Interface (AWS CLI) per creare una policy IAM e un utente per IDT, quindi allegare le policy all'utente.

- [Per configurare le autorizzazioni per IDT \(Console\)](#)
- [Per configurare le autorizzazioni per IDT \(AWS CLI\)](#)

Per configurare le autorizzazioni per IDT (Console)

Attenersi alla seguente procedura per utilizzare la console per configurare le autorizzazioni per IDT per AWS IoT Greengrass.

1. Accedere alla [console IAM](#).
2. Creare un criterio gestito dal cliente che concede le autorizzazioni per creare ruoli con autorizzazioni specifiche.
 - a. Nel riquadro di navigazione, seleziona Policy e quindi Crea policy.

- b. Nella scheda JSON, sostituire il contenuto del segnaposto con la seguente policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot-device-tester:SendMetrics"
      ],
      "Resource": "*"
    }
  ]
}
```

- c. Scegliere Next: Tags (Successivo: Tag).
- d. Scegliere Next:Review (Successivo: Rivedi).
- e. Per Nome, immetti **IDTUsageMetricsIAMPermissions**. In Riepilogo, esaminare le autorizzazioni concesse dai criteri.
- f. Scegliere Create Policy (Crea policy).
3. Crea un utente IAM e assegna le autorizzazioni all'utente.
- a. Crea un utente IAM. Segui i passaggi da 1 a 5 in [Creazione di utenti IAM \(console\)](#) nella Guida per l'utente IAM. Se hai già creato un utente IAM, vai al passaggio successivo.
- b. Allega le autorizzazioni al tuo utente IAM:
- Nella pagina Imposta le autorizzazioni, scegli Allega direttamente le politiche esistenti.
 - Cerca la IAMPermissions policy IDTUsageMetrics che hai creato nel passaggio precedente. Selezionare la casella di controllo.
- c. Scegli Successivo: Tag.
- d. Scegliere Next:Review per visualizzare un riepilogo delle tue scelte.
- e. Selezionare Create user (Crea utente).
- f. Per visualizzare le chiavi di accesso dell'utente (chiave di accesso IDs e chiavi di accesso segrete), scegli Mostra accanto alla password e alla chiave di accesso. Per salvare le chiavi di accesso, scegliere Scarica .csv e salvare il file in una posizione sicura. Utilizzerai queste informazioni in seguito per configurare il file AWS delle credenziali.

Per configurare le autorizzazioni per IDT (AWS CLI)

Segui questi passaggi per utilizzare per configurare AWS CLI le autorizzazioni per IDT. AWS IoT Greengrass Se sono già state configurate le autorizzazioni nella console, passare a [the section called “Configura il tuo dispositivo per eseguire test IDT”](#) o [the section called “Facoltativo: configurazione del container Docker”](#).

1. Sul tuo computer, installa e configura il file AWS CLI se non è già installato. Segui la procedura descritta in [Installazione di AWS CLI nella Guida AWS Command Line Interface](#) per l'utente.

 Note

AWS CLI È uno strumento open source che puoi utilizzare per interagire con AWS i servizi dalla tua shell a riga di comando.

2. Crea la seguente politica gestita dai clienti che concede le autorizzazioni per gestire IDT e ruoli. AWS IoT Greengrass

Linux, macOS, or Unix

```
aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-document '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot-device-tester:SendMetrics"
      ],
      "Resource": "*"
    }
  ]
}'
```

Windows command prompt

```
aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-document
```

```
'{"Version": "2012-10-17",  
  "Statement": [{"Effect": "Allow", "Action": ["iot-device-  
tester:SendMetrics"], "Resource": "*"}]}'
```

 Note

Questo passaggio include un esempio del prompt dei comandi di Windows perché utilizza una sintassi JSON diversa rispetto ai comandi del terminale Linux, macOS o Unix.

3. Crea un utente IAM e allega le autorizzazioni richieste da IDT for. AWS IoT Greengrass
 - a. Crea un utente IAM.

```
aws iam create-user --user-name user-name
```

- b. Allega la IDTUsageMetricsIAMPermissions policy che hai creato al tuo utente IAM. Sostituiscila *user-name* con il tuo nome utente IAM e *<account-id>* nel comando con l'ID del tuo Account AWS.

```
aws iam attach-user-policy --user-name user-name --policy-arn  
arn:aws:iam::<account-id>:policy/IDTGreengrassIAMPermissions
```

4. Creare una chiave di accesso segreta per l'utente.

```
aws iam create-access-key --user-name user-name
```

Memorizzare l'output in una posizione sicura. Utilizzerai queste informazioni in seguito per configurare il file AWS delle credenziali.

Fornisci le AWS credenziali a IDT

Per consentire a IDT di accedere alle tue AWS credenziali e inviare le metriche a AWS, procedi come segue:

1. Archivia le AWS credenziali per il tuo utente IAM come variabili di ambiente o in un file di credenziali:
 - a. Per utilizzare le variabili di ambiente, esegui il comando seguente:


```
AWS_ACCESS_KEY_ID=access-key  
AWS_SECRET_ACCESS_KEY=secret-access-key
```

- b. Per utilizzare il file delle credenziali, aggiungete le seguenti informazioni al `.aws/credentials` file:

```
[profile-name]  
aws_access_key_id=access-key  
aws_secret_access_key=secret-access-key
```

2. Configura la auth sezione del `config.json` file. Per ulteriori informazioni, consulta [\(Facoltativo\) Configura config.json](#).

IDT per la risoluzione dei problemi AWS IoT Greengrass

IDT for AWS IoT Greengrass scrive questi errori in varie posizioni in base al tipo di errore. Gli errori vengono scritti nella console, nei file di log e nei report di test.

Codici di errore

Nella tabella seguente vengono elencati i codici di errore generati da IDT per AWS IoT Greengrass.

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
101	InternalError	Si è verificato un errore interno.	Controlla i log nella directory <code><device-t ester-extract- location></code> / results. Se non riesci a eseguire il debug del problema, contatta AWS Developer Support .
102	TimeoutError	Il test non può essere completato in un	•

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
		intervallo di tempo limitato. Questo può accadere se: • La connessione di rete tra il computer di test e il dispositivo è lenta (ad esempio, se si utilizza una rete VPN). • Una rete lenta ritarda la comunicazione tra il dispositivo e il cloud. • Il campo <code>timeout</code> nei file di configurazione di test (<code>test.json</code>) è stato accidentalmente modificato.	Controllare la connessione e la velocità di rete. • Assicurarsi di non aver modificato alcun file nella directory <code>/test</code>. • Provare a eseguire manualmente il gruppo di test non riuscito con il flag <code>--group-id</code>. • Provare a eseguire la suite di test aumentando i timeout di test. Per ulteriori informazioni, consulta Errori di timeout.

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
103	PlatformNotSupport Error	Combinazione di sistema operativo/ architettura errata specificata in <code>device.json</code> .	Modificare la configurazione in una delle combinazioni supportate: • Linux, x86_64 • Linux, ARMv6l • Linux, ARMv7l • Linux, AArch64 • Ubuntu, x86_64 • OpenWRT, ARMv7l • OpenWRT, AArch64 Per ulteriori informazioni, consulta Configura dispositivo.json.

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
104	VersionNotSupportError	La versione del software AWS IoT Greengrass Core non è supportata dalla versione di IDT in uso.	Usa il <code>device_tester_bin</code> version comando per trovare la versione supportata del software AWS IoT Greengrass Core. Ad esempio, se si sta utilizzando macOS, usa <code>./device_tester_mac_x86_64</code> version. Per trovare la versione del software AWS IoT Greengrass Core che stai utilizzando: Se esegui test con il software AWS IoT Greengrass Core preinstallato, usa SSH per connetterti al tuo dispositivo AWS IoT Greengrass principale ed esegui <code><path-to-preinstalled-green-grass-location> /greengrass</code>

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
			<pre>ss/ggc/core/greengrassd --version</pre> • Se stai eseguendo test con una versione diversa del software AWS IoT Greengrass Core, vai alla directory <code>ter_green_grass_<os>/products/greengrass/gcc</code> directory. La versione del software AWS IoT Greengrass Core fa parte del nome del file.zip. È possibile testare una versione diversa del software AWS IoT Greengrass Core. Per ulteriori informazioni, consulta Guida introduttiva con AWS IoT Greengrass.

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
105	LanguageNotSupport Error	IDT supporta Python solo AWS IoT Greengrass per le librerie SDKs .	Assicurarsi che: • Il pacchetto SDK in devicetester_green_grass_ <i><os></i>/products/greengrass/ggsdk è l'SDK Python. • I contenuti della cartella bin in devicetester_green_grass_ <i><os></i>/tests/GGQ_1.0.0/suite/resources/runtime/arm/bin non sono stati modificati.

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
106	ValidationError	Alcuni campi in <code>device.json</code> o <code>config.json</code> non sono validi.	Controllare il messaggio di errore a destra del codice di errore nel report. • Tipo di autenticazione non valido per il dispositivo: specificare il metodo corretto per collegarsi al dispositivo. Per ulteriori informazioni, consulta the section called “Configura dispositivo.json”. • Percorso chiave privata non valido: specificare il percorso corretto alla chiave privata. Per ulteriori informazioni, consulta Configura dispositivo.json. • Non valido Regione AWS: specifica un valore valido Regione AWS nel file <code>config.js</code>

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
			on Per ulteriori informazioni, consulta Endpoint del servizio AWS. AWS credenziali: imposta AWS credenziali valide sulla macchina di test (utilizzando le variabili di ambiente o il <code>credentials</code> file). Verificare che il campo <code>auth</code> sia configurato correttamente. Per ulteriori informazioni, consulta the section called “Create and configure an AWS Account”. Input HSM non valido: controllare i campi <code>p11Provider</code>, <code>privateKeyLabel</code>, <code>slotLabel</code>, <code>slotUserPin</code> e <code>opensslEngine</code> in <code>device.json</code>.

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
107	SSHConnectionFallito	Il computer di test non si connette al dispositivo configurato.	Verificare che i seguenti campi nel file <code>device.json</code> siano corretti: • <code>ip</code> • <code>user</code> • <code>privKeyPath</code> • <code>password</code> Per ulteriori informazioni, consulta Configura dispositivi.json.

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
108	RunCommandError	Un test non è riuscito a eseguire un comando sul dispositivo sottoposto a test.	Verificare che l'accesso root sia consentito per l'utente configurato in <code>device.json</code>. Alcuni dispositivi vi richiedono una password durante l'esecuzione di comandi con accesso alla radice. Assicurarsi che l'accesso root sia consentito senza una password. Per ulteriori informazioni, consultare la documentazione relativa al dispositivo. Provare a eseguire manualmente sul dispositivo il comando che non viene eseguito per controllare se si verifica un errore.
109	PermissionDeniedError	Nessun accesso root.	Impostare l'accesso root per l'utente configurato sul dispositivo.

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
110	CreateFileError	Impossibile creare un file.	Controllare lo spazio su disco del dispositivo e le autorizzazioni di directory.
111	CreateDirError	Impossibile creare una directory.	Controllare lo spazio su disco del dispositivo e le autorizzazioni di directory.
112	InvalidPathError	Il percorso verso il software AWS IoT Greengrass Core non è corretto.	Verificare che il percorso nel messaggio di errore sia valido. Non modificare i file nella directory <code>devicetester_green_grass_<os></code> .
113	InvalidFileError	Un file non è valido.	Verificare che il file nel messaggio di errore sia valido.

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
114	ReadFileError	Impossibile leggere il file specificato.	Verificare quanto segue: • Le autorizzazioni per i file sono corrette. • <code>limits.config</code> consente di aprire un numero di file sufficiente. • Il file specificato nel messaggio di errore esiste ed è valido. Se il test viene eseguito su un macOS, aumentare il limite di file aperti. Il limite predefinito è di 256, sufficiente per eseguire il test.

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
115	FileNotFoundException	Un file obbligatorio non è stato trovato.	Verificare quanto segue: • Un file Greengrass compresso esiste sotto <code>devicetes ter_green grass_ <os>/ products/ greengrass/ ggc</code>. È possibile scaricare il file tar AWS IoT Greengrass Core dalla pagina dei download del software AWS IoT Greengrass Core. • Il pacchetto SDK è presente in <code>devicetes ter_green grass_ <os>/ products/ greengrass/ ggsdk</code>. • I file in <code>devicetes ter_green grass_ <os>/ tests</code> non sono stati modificati.

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
116	OpenFileFailed	Impossibile aprire il file specificato.	Verificare quanto segue: • Il file specificato nel messaggio di errore esiste ed è valido. • <code>limits.config</code> consente di aprire un numero di file sufficiente. Se il test viene eseguito su un macOS, aumentare il limite di file aperti. Il limite predefinito è di 256, sufficiente per eseguire il test.
117	WriteFileFailed	Impossibile scrivere nel file (può essere il computer in fase di test o il computer di test).	Verificate che la directory specificata nel messaggio di errore esista e che disponiate del permesso di scrittura.

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
118	FileCleanUpError	Un test non è stato in grado di rimuovere la directory o il file specificato oppure di eseguire l'umount del file specificato sul dispositivo remoto.	Se il file binario è ancora in esecuzione, potrebbe essere bloccato. Terminare il processo ed eliminare il file specificato.
119	InvalidInputError	Configurazione non valida.	Verificare che il file <code>suite.json</code> sia valido.
120	InvalidCredentialError	AWS Credenziali non valide.	• Verifica le tue credenziali. AWS Per ulteriori informazioni, consulta the section called “Configura le tue credenziali AWS”. • Verificare la connessione di rete ed eseguire nuovamente il gruppo di test. Anche i problemi di rete possono causare questo errore.

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
121	AWSSessionError	Impossibile creare una AWS sessione.	Questo errore può verificarsi se AWS le credenziali non sono valide o la connessione Internet è instabile. Prova a utilizzare AWS CLI per chiamare un'AWS operazione API.
122	AWSApiCallError	Si è verificato un errore AWS API.	Questo errore potrebbe essere causato da un problema di rete. Verificare la rete prima di riprovare il gruppo di test.

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
123	IpNotExistError	L'indirizzo IP non è incluso nelle informazioni di connettività.	Verificare la connessione Internet. È possibile utilizzare la AWS IoT Greengrass console per verificare le informazioni di connettività relative all'elemento AWS IoT Greengrass principale utilizzato dal test. Se le informazioni di connettività includono 10 endpoint è possibile rimuoverne alcuni o tutti e rieseguire il test. Per ulteriori informazioni, consulta Informazioni di connettività .
124	OTAJobNotCompleteError	Un processo OTA non è terminato.	Verificare la connessione Internet e riprovare a eseguire il gruppo di test OTA.

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
125	CreateGreengrassServiceRoleError	Si è verificato uno degli eventi seguenti: • Si è verificato un errore durante la creazione di un ruolo. • Si è verificato un errore durante l'associazione di una policy al ruolo AWS IoT Greengrass di servizio. • La policy associata al ruolo del servizio non è valida. • Si è verificato un errore durante l'associazione di un ruolo a un Account AWS	Configurare il ruolo AWS IoT Greengrass di servizio. Per ulteriori informazioni, consulta the section called “Ruolo del servizio Greengrass”.

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
126	DependenciesNotPresentError	Una o più dipendenze e necessarie per i test specifici non sono presenti sul dispositivo.	Controllare il log di test per vedere quali sono le dipendenze mancanti nel dispositivo: <i><device-tester-extension> /results/<execution-id>/logs/<test-case-name.log></i> .
127	Non valido HSMConfiguration	La configurazione HSM/PKCS fornita non è corretta.	Nel file <code>device.js</code> on , fornire la configurazione necessaria per interagire con HSM utilizzando PKCS#11.

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
128	OTAJobNotSucceededError	Il processo OTA non è andato a buon fine.	• Se il gruppo di test ota è stato eseguito singolarmente, eseguire il gruppo di test ggcdependencies per verificare che siano presenti tutte le dipendenze (ad esempio wget). Quindi ripetere il gruppo di test ota. • Esaminare i log dettagliati sotto <code><device-tester-extract-location> / results/ <execution-id>/logs/</code> per informazioni sulla risoluzione dei problemi e sugli errori. In particolare, controllare i seguenti log: • Log della console (test_manager.log)

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
			• Log dei casi di test OTA (ota_test.log) • Log del daemon GGC (ota_test_ggc_logs.tar.gz) • Log dell'agente OTA (ota_test_ota_logs.tar.gz) • Verificare la connessione Internet e riprovare a eseguire il gruppo di test ota. • Se il problema persiste, contatta AWS Developer Support.

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
129	NoConnectivityError	L'agente host non riesce a connettersi a Internet.	Controlla la connessione di rete e le impostazioni firewall. Riprovare il gruppo di test dopo che il problema di connettività è stato risolto.
130	NoPermissionError	L'utente IAM per cui stai utilizzando IDT AWS IoT Greengrass non è autorizzato a creare le AWS risorse necessarie per eseguire IDT.	Consulta l'argomento relativo al modello della policy di autorizzazione per il modello di policy che concede le autorizzazioni richieste per eseguire IDT per AWS IoT Greengrass.

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
131	LeftoverAgentExist Error	Il tuo dispositivo sta eseguendo AWS IoT Greengrass dei processi quando tenti di avviare IDT for. AWS IoT Greengrass	Assicurati che non sia presente alcun daemon Greengrass esistente in esecuzione e sul dispositivo. • Puoi utilizzare questo comando per arrestare il daemon: <code>sudo ./<absolute-path-to-greengrass-daemon> /greengrassd stop.</code> • Puoi anche terminare il daemon Greengrass tramite PID.  Note Se si utilizza un'installazione esistente di AWS IoT Greengrass configurata per l'avvio automatico dopo il riavvio,

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
			è necessari o arrestare il demone dopo il riavvio e prima di eseguire la suite di test.
132	DeviceTimeOffsetError	L'ora del dispositivo non è corretta.	Imposta il dispositivo sull'ora corretta.
133	Non valido MLConfiguration	La configurazione ML fornita non è corretta.	Nel file <code>device.json</code>, fornisci la configurazione corretta necessaria per eseguire i test di inferenza di ML. Per ulteriori informazioni, consulta the section called “Facoltativo: configurazione del dispositivo per la qualificazione di ML”.

Risoluzione di IDT per errori AWS IoT Greengrass

Quando si utilizza IDT, è necessario disporre dei file di configurazione corretti prima di eseguire IDT for. AWS IoT Greengrass Se ottieni errori di parsing e di configurazione, per prima cosa dovresti individuare e utilizzare un modello di configurazione appropriato per il tuo ambiente.

Se continui a riscontrare problemi, consulta la seguente procedura di debug.

Argomenti

- [Dove posso cercare gli errori?](#)

- [Errori di parsing](#)
- [Errore di parametro richiesto mancante](#)
- [Errore di avvio del test non riuscito](#)
- [Errore di accesso non autorizzato alla risorsa](#)
- [Errori di autorizzazione negata](#)
- [Errori di connessione SSH](#)
- [Errori di timeout](#)
- [Errori di comando non trovato durante l'esecuzione del test](#)
- [Eccezione di sicurezza su macOS](#)

Dove posso cercare gli errori?

Gli errori di alto livello vengono visualizzati sulla console durante l'esecuzione e un riepilogo dei test non riusciti con l'errore viene visualizzato quando tutti i test sono stati completati.

`awsiotdevicetester_report.xml` contiene un riepilogo di tutti gli errori che hanno causato un test non riuscito. I file di log per ogni sessione di test vengono archiviati in una directory denominata con un UUID per l'esecuzione del test che è stato visualizzato nella console durante la sessione di test.

La directory dei log di test si trova in `<device-tester-extract-location>/results/<execution-id>/logs/`. Questa directory contiene i file seguenti, che sono utili per il debug.

File	Descrizione
<code>test_manager.log</code>	Tutti i log che sono stati scritti nella console durante l'esecuzione del test. Un riepilogo dei risultati si trova al termine di questo file che include un elenco dei test non riusciti. I log di avviso e di errore in questo file possono fornire informazioni sull'errore.
<code><test-group-id> __<test-name> .log</code>	Log dettagliati per il test specifico.

File	Descrizione
<code><test-name> _ggc_logs.tar.gz</code>	Una raccolta compressa di tutti i log generati dal demone AWS IoT Greengrass principal e durante il test. Per ulteriori informazioni, consulta Risoluzione dei problemi AWS IoT Greengrass .
<code><test-name> _ota_logs.tar.gz</code>	Una raccolta compressa di log generati dall'agente OTA durante il AWS IoT Greengrass test. Solo per i test OTA.
<code><test-name> _basic_assertion_publisher_ggad_logs.tar.gz</code>	Una raccolta compressa di log generati dal dispositivo editore AWS IoT durante il test.
<code><test-name> _basic_assertion_subscriber_ggad_logs.tar.gz</code>	Una raccolta compressa di log generati dal dispositivo sottoscrittore AWS IoT durante il test.

Errori di parsing

Talvolta un refuso in una configurazione JSON può causare errori di parsing. Nella maggior parte dei casi, il problema è dovuto all'omissione di parentesi, virgole o virgolette nel file JSON. IDT esegue una convalida JSON e visualizza le informazioni di debug. Inoltre indica la riga in cui si è verificato l'errore, il numero di riga e il numero di colonna dell'errore di sintassi. Queste informazioni dovrebbero essere sufficienti per aiutarti a correggere l'errore, ma se ancora non riesci a individuare l'errore, puoi eseguire la convalida manualmente nel tuo IDE, in un editor di testo come Atom o Sublime o tramite uno strumento online come [JSONLint](#).

Errore di parametro richiesto mancante

Poiché si stanno aggiungendo nuove caratteristiche a IDT, potrebbero essere introdotte modifiche ai file di configurazione. L'utilizzo di un file di configurazione precedente potrebbe invalidare la tua configurazione. Se dovesse succedere, il file `<test_case_id>.log` in `/results/<execution-id>/logs` elenca in modo esplicito tutti i parametri mancanti. IDT convalida inoltre gli schemi dei tuoi file di configurazione JSON per assicurare che sia stata utilizzata la versione supportata più recente.

Errore di avvio del test non riuscito

È possibile che si verifichino errori relativi a problemi in fase di avvio del test. Le cause sono diverse, quindi esegui le operazioni descritte di seguito:

- Assicurati che il nome del pool incluso nel comando di esecuzione esista effettivamente. Al nome del pool si fa riferimento direttamente nel file `device.json`.
- Verifica che i parametri di configurazione dei dispositivi nel pool siano corretti.

Errore di accesso non autorizzato alla risorsa

È possibile che venga visualizzato il messaggio di errore `<user or role> is not authorized to access this resource` nell'output del terminale o nel file `test_manager.log` sotto `/results/<execution-id>/logs`. Per risolvere questo problema, associare la policy `AWSIoTDeviceTesterForGreengrassFullAccess` gestita all'utente del test. Per ulteriori informazioni, consulta [the section called “Crea e configura un Account AWS”](#).

Errori di autorizzazione negata

IDT esegue operazioni su varie directory e file in un dispositivo sottoposto a test. Alcune di queste operazioni richiedono l'accesso root. Per automatizzare queste operazioni, IDT deve essere in grado di eseguire comandi con il comando `sudo` senza la digitazione di una password.

Segui questi passaggi per consentire l'accesso al comando `sudo` senza la digitazione di una password.

Note

`user` e `username` si riferiscono all'utente SSH utilizzato da IDT per accedere al dispositivo sottoposto a test.

1. Utilizza `sudo usermod -aG sudo <ssh-username>` per aggiungere l'utente SSH al gruppo `sudo`.
2. Per rendere effettive le modifiche, esci ed esegui di nuovo l'accesso.
3. Apri il file `/etc/sudoers` e aggiungi la riga seguente alla fine del file: `<ssh-username> ALL=(ALL) NOPASSWD: ALL`

 Note

Come best practice, ti consigliamo di utilizzare `sudo visudo` quando modifichi `/etc/sudoers`.

Errori di connessione SSH

Quando UDT non è in grado di connettersi a un dispositivo sottoposto a test, gli errori di connessione vengono registrati in `/results/<execution-id>/logs/<test-case-id>.log`. I messaggi di errore SSH vengono visualizzati all'inizio di questo file di log perché la connessione a un dispositivo sottoposto a test è una delle prime operazioni eseguite da IDT.

La maggior parte delle configurazioni di Windows utilizza l'applicazione TTY terminale Pu per connettersi agli host Linux. Questa applicazione richiede che i file della chiave privata PEM standard siano convertiti in un formato Windows proprietario denominato PPK. Quando IDT è configurato nel `device.json`, utilizza solo i file PEM. Se si utilizza un file PPK, IDT non può creare una connessione SSH con il AWS IoT Greengrass dispositivo e non può eseguire test.

Errori di timeout

Puoi aumentare il timeout per ogni test specificando un moltiplicatore di timeout, che viene applicato al valore predefinito di ogni timeout del test. Qualsiasi valore configurato per questo flag deve essere maggiore o uguale a 1.0.

Per usare il moltiplicatore di timeout, utilizza il flag `--timeout-multiplier` durante l'esecuzione dei test. Per esempio:

```
./devicetester_linux run-suite --suite-id GGQ_1.0.0 --pool-id DevicePool1 --timeout-multiplier 2.5
```

Per ulteriori informazioni, esegui `run-suite --help`.

Errori di comando non trovato durante l'esecuzione del test

È necessaria una versione precedente della libreria OpenSSL (libssl1.0.0) per eseguire i test sui dispositivi. AWS IoT Greengrass La maggior parte delle distribuzioni Linux correnti utilizza libssl versione 1.0.2 o versioni successive (v1.1.0).

Ad esempio, su un Raspberry Pi, esegui i seguenti comandi per installare la versione richiesta di libssl:

1.

```
wget http://ftp.us.debian.org/debian/pool/main/o/openssl/  
libssl1.0.0_1.0.2l-1~bpo8+1_armhf.deb
```
2.

```
sudo dpkg -i libssl1.0.0_1.0.2l-1~bpo8+1_armhf.deb
```

Eccezione di sicurezza su macOS

Quando si esegue IDT su un computer host che utilizza macOS 10.15, il ticket di autenticazione notarile per IDT non viene rilevato correttamente e l'esecuzione di IDT è bloccata. Per eseguire IDT, è necessario concedere un'eccezione di sicurezza all'eseguibile. `devicetester_mac_x86-64`

Per concedere un'eccezione di sicurezza all'eseguibile IDT

1. Avvia Preferenze di Sistema dal menu Apple.
2. Scegli Sicurezza e privacy, quindi nella scheda Generale, fai clic sull'icona del lucchetto per apportare modifiche alle impostazioni di sicurezza.
3. Cerca il messaggio "`devicetester_mac_x86-64`" was blocked from use because it is not from an identified developer. e scegli Consenti comunque.
4. Accetta l'avviso di sicurezza.

In caso di domande sulla politica di supporto IDT, contatta l'[AWS assistenza clienti](#).

Politica di supporto per AWS IoT Device Tester per AWS IoT Greengrass V1

AWS IoT [Device Tester \(IDT\) for AWS IoT Greengrass](#) è un framework di test scaricabile che [consente di convalidare e qualificare i AWS IoT Greengrass dispositivi per l'inclusione nel Device Catalog.AWS Partner](#) Ti consigliamo di utilizzare la versione più recente di IDT per testare o qualificare AWS IoT Greengrass i tuoi dispositivi. Per ulteriori informazioni, consulta [Versioni supportate di IDT AWS IoT Greengrass V2 nella Guida per gli](#) AWS IoT Greengrass Version 2 sviluppatori.

Puoi anche utilizzare una qualsiasi delle versioni supportate di AWS IoT Greengrass IDT per testare o qualificare i tuoi dispositivi. Sebbene sia possibile continuare a utilizzare [versioni non supportate di IDT](#), tali versioni non ricevono correzioni di bug o aggiornamenti.

 Important

A partire dal 4 aprile 2022, AWS IoT Device Tester (IDT) for AWS IoT Greengrass V1 non genera più report di qualificazione firmati. [Non è più possibile qualificare nuovi AWS IoT Greengrass V1 dispositivi per inserirli nel catalogo dei dispositivi tramite il AWS Partner Device Qualification Program.AWS](#) Anche se non puoi qualificare i dispositivi Greengrass V1, puoi continuare a utilizzare IDT per AWS IoT Greengrass V1 testare i tuoi dispositivi Greengrass V1. [Ti consigliamo di utilizzare IDT per qualificare ed AWS IoT Greengrass V2 elencare i dispositivi Greengrass nel Device Catalog.AWS Partner](#)

In caso di domande sui criteri di supporto, contattare il [supporto clienti AWS](#).

Risoluzione dei problemi AWS IoT Greengrass

Questa sezione fornisce informazioni sulla risoluzione dei problemi e le possibili soluzioni per aiutare a risolvere i problemi relativi a. AWS IoT Greengrass

Per informazioni sulle AWS IoT Greengrass quote (limiti), vedere [Service](#) Quotas in. Riferimenti generali di Amazon Web Services

AWS IoT Greengrass Questioni principali

Se il software AWS IoT Greengrass Core non si avvia, prova i seguenti passaggi generali per la risoluzione dei problemi:

- Assicurati di installare i file binari appropriati per l'architettura. Per ulteriori informazioni, consulta [AWS IoT Greengrass Core Software](#).
- Assicurati che il tuo dispositivo core disponga di storage locale. Per ulteriori informazioni, consulta [the section called “Risoluzione dei problemi di storage”](#).
- Controlla `runtime.log` e `crash.log` per i messaggi di errore. Per ulteriori informazioni, consulta [the section called “Risoluzione dei problemi con i log”](#).

Cerca tra i seguenti sintomi ed errori per trovare informazioni utili a risolvere i problemi relativi a un AWS IoT Greengrass core.

Problemi

- [Errore: nel file di configurazione manca il CaPath, CertPath o. KeyPath Il processo daemon Greengrass con \[pid = <pid>\] è terminato.](#)
- [Errore: impossibile analizzare/<greengrass-root> /config/config.json.](#)
- [Errore: si è verificato un errore durante la generazione della configurazione TLS: ErrUnknown URIScheme](#)
- [Errore: l'avvio di Runtime non è riuscito: impossibile avviare i worker: test del container scaduto.](#)
- [<address>Errore: Impossibile richiamare PutLogEvents su Cloudwatch locale, LogGroup:/GreengrassSystem/connection_manager, errore:: invio della richiesta non riuscito causato da: Post http RequestError://<path>/cloudwatch/logs/: dial tcp: getsockopt: connessione rifiutata, risposta: {}.](#)

- [<region><account-id><function-name><version><file-name>Errore: impossibile creare il server a causa di: failed to load group: chmod/<greengrass-root>/:aws:lambda: :function::ggc/deployment/lambda/arn: nessun file o directory di questo tipo.](#)
- [Il software AWS IoT Greengrass Core non si avvia dopo il passaggio dall'esecuzione senza containerizzazione all'esecuzione in un contenitore Greengrass.](#)
- [Errore: la dimensione di spool deve essere almeno 262144 byte.](#)
- [Errore: \[ERROR\]-Cloud messaging error: si è verificato un errore durante il tentativo di pubblicare un messaggio. {"errorString": "operation timed out"}](#)
- [<version>Errore: container_linux.go:344: l'avvio del processo del contenitore ha causato «process_linux.go:424: l'init del contenitore ha causato\" rootfs_linux.go:64: mount\\\"\"\"_ipc.sock: autorizzazione negata\\\" \"\"\". greengrass/ggc/socket/greengrass_ipc.sock\\\"\" to rootfs \\\"\"/greengrass/ggc/packages rootfs/merged\\\"\" at \\\"\"/greengrass_ipc.sock\\\"\" caused \\\"\"stat /greengrass/ggc/socket/greengrass](#)
- [Errore: daemon Greengrass in esecuzione con PID: <process-id>. Alcuni componenti di sistema non sono stati avviati. Controlla se ci sono errori in 'runtime.log'.](#)
- [La shadow del dispositivo non si sincronizza con il cloud.](#)
- [ERRORE: non in grado di accettare la connessione TCP. accept tcp \[::\]:8000: accept4: troppi file aperti.](#)
- [Errore: errore di esecuzione runtime: impossibile avviare il container lambda. container_linux.go:259: l'avvio del processo del container ha causato "process_linux.go:345: container init caused \"rootfs_linux.go:50: preparing rootfs caused \\\"permission denied\\\"\"\".](#)
- [Avviso: \[WARN\] - \[5\] GK Remote: errore durante il recupero dei dati della chiave pubblica: la chiave privata per non è ErrPrincipalNotConfigured impostata. MqttCertificate](#)
- [<account-id><role-name><region>Errore: autorizzazione negata durante il tentativo di utilizzare il ruolo arn:aws:iam: :role/ per accedere a s3 url https://-greengrass-updates.s3.<region><architecture><distribution-version>.amazonaws.com/core/ /greengrass-core- .tar.gz.](#)
- [Il AWS IoT Greengrass core è configurato per utilizzare un proxy di rete e la funzione Lambda non può effettuare connessioni in uscita.](#)
- [Il core si trova in un ciclo infinito di connessione-disconnessione. Il file runtime.log contiene una serie di voci di connessione e disconnessione.](#)
- [Errore: impossibile avviare il container lambda. container_linux.go: 259: l'avvio del processo del container ha causato "process_linux.go: 345: container init caused\"rootfs_linux.go: 62: mounting \\\"proc\\\"\" to rootfs \\\"](#)

- [\[ERRORE\] -errore di esecuzione in fase di esecuzione: impossibile avviare il contenitore lambda. <ggc-path>{"errorString»: «impossibile inizializzare i supporti dei contenitori: impossibile mascherare la radice di greengrass nella directory superiore di overlay: impossibile creare il dispositivo maschera nella directory: il file esiste"}](#)
- [\[ERRORE\] -Distribuzione non riuscita. {"DeploymentId»: <deployment-id>"«, «errorString»: «Processo di test del contenitore con <pid>pid non riuscito: stato del processo del contenitore: stato di uscita 1"}](#)
- [Errore: \[ERROR\] -Errore di esecuzione di runtime: impossibile avviare il contenitore lambda. <ggc-version><ggc-version>{"errorString»: «impossibile inizializzare i supporti del contenitore: impossibile creare l'overlay fs per il contenitore: montaggio dell'overlay su/////<ggc-version><ggc-version><ggc-version>/dns:/,greengrass/ggc/packages/rootfs/merged failed: failed to mount with args source="\no_source\" dest="\greengrass/ggc/packagesrootfs/merged\" fstype="\overlay\" flags="\0\" data="\lowerdir=/greengrass/ggc/packages//rootfs/work\»: troppi livelli di collegamenti upperdir=/greengrass/ggc/packages simbolici " rootfs/upper,workdir=/greengrass/ggc/packages](#)
- [Errore: \[DEBUG\] - Impossibile ottenere route. Eliminare il messaggio.](#)
- [Errore: \[Errno 24\] Troppi file aperti <lambda-function>, \[Errno 24\] Troppi file aperti](#)
- [Errore: il server ds non è riuscito ad avviare l'ascolto di socket: listen unix<ggc-path>/ggc/socket/greengrass_ipc.sock: bind: argomento non valido](#)
- [\[INFORMAZIONI\] \(Copier\) aws.greengrass. StreamManager: stdout. Causato da: com.fasterxml.jackson.databind. JsonMappingException: Instant supera l'istante minimo o massimo](#)
- [GPG error: https://dnw9lb6lzp2d8.cloudfront.net stable InRelease: The following signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass Master Key](#)

Errore: nel file di configurazione manca il CaPath, CertPath o. KeyPath Il processo daemon Greengrass con [pid = <pid>] è terminato.

Soluzione: questo errore potrebbe comparire `crash.log` quando il software AWS IoT Greengrass Core non si avvia. Questa situazione può verificarsi se esegui v1.6 o una versione precedente. Esegui una di queste operazioni:

- Esegui l'aggiornamento alla versione 1.7 o successiva. Ti consigliamo di utilizzare sempre la versione più recente del software AWS IoT Greengrass Core. Per scaricare le informazioni, consulta [AWS IoT Greengrass Core Software](#).
- Utilizzate il `config.json` formato corretto per la versione AWS IoT Greengrass del software Core in uso. Per ulteriori informazioni, consulta [the section called “AWS IoT Greengrass file di configurazione principale”](#).

Note

Per scoprire quale versione del software AWS IoT Greengrass Core è installata sul dispositivo principale, esegui i seguenti comandi nel terminale del dispositivo.

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

Errore: impossibile analizzare/<greengrass-root> /config/config.json.

Soluzione: è possibile visualizzare questo errore quando il software AWS IoT Greengrass Core non si avvia. Verifica che il [file di configurazione Greengrass](#) utilizzi un formato JSON valido.

Apri `config.json` (situato in `/greengrass-root/config`) e convalida il formato JSON. Ad esempio, verifica che le virgole siano utilizzate correttamente.

Errore: si è verificato un errore durante la generazione della configurazione TLS: ErrUnknown URIScheme

Soluzione: è possibile visualizzare questo errore quando il software AWS IoT Greengrass Core non si avvia. Assicurati che le proprietà nella sezione [crypto](#) del file di configurazione Greengrass siano valide. Il messaggio di errore fornisce ulteriori informazioni.

Aprire `config.json` (disponibile in `/greengrass-root/config`) e controllare la sezione `crypto`. Ad esempio, i percorsi dei certificati e delle chiavi devono utilizzare il formato URI corretto e puntare alla posizione corretta.

Errore: l'avvio di Runtime non è riuscito: impossibile avviare i worker: test del container scaduto.

Soluzione: è possibile visualizzare questo errore quando il software AWS IoT Greengrass Core non si avvia. Imposta la proprietà `postStartHealthCheckTimeout` nel [file di configurazione Greengrass](#). Questa proprietà facoltativa configura la quantità di tempo (in millisecondi) che il daemon Greengrass attende per completare il controllo dello stato post-avvio. Il valore predefinito è 30 secondi (30000 ms).

Apri `config.json` (situato in `/greengrass-root/config`). Nell'oggetto `runtime`, aggiungi la proprietà `postStartHealthCheckTimeout` e imposta il valore su un numero maggiore di 30000. Aggiungi una virgola dove necessario per creare un documento JSON valido. Per esempio:

```
...
"runtime" : {
  "cgroup" : {
    "useSystemd" : "yes"
  },
  "postStartHealthCheckTimeout" : 40000
},
...
```

<address>Errore: Impossibile richiamare PutLogEvents su Cloudwatch locale, LogGroup:/GreengrassSystem/connection_manager, errore:: invio della richiesta non riuscito causato da: Post http RequestError://<path>/cloudwatch/logs/: dial tcp: getsockopt: connessione rifiutata, risposta: {}.

Soluzione: è possibile visualizzare questo errore quando il software AWS IoT Greengrass Core non si avvia. Ciò può verificarsi se si utilizza un Raspberry Pi e la configurazione della memoria richiesta non è stata completata. AWS IoT Greengrass Per ulteriori informazioni, consultare [questa fase](#).

<region><account-id><function-name><version><file-name>Errore: impossibile creare il server a causa di: failed to load group: chmod/<greengrass-root>/:aws:lambda: :function::/ggc/deployment/lambda/arn: nessun file o directory di questo tipo.

Soluzione: è possibile visualizzare questo errore quando il software AWS IoT Greengrass Core non si avvia. Se hai distribuito un eseguibile [Lambda](#) nel core, controlla la proprietà `Handler` della funzione nel file (che si trova in `group.json` *greengrass-root*//ggc/deployment/group). Se il nome dell'handler non corrisponde esattamente al nome del file eseguibile compilato, sostituisci i contenuti del file `group.json` con un oggetto JSON vuoto (`{}`) ed esegui i comandi seguenti per avviare AWS IoT Greengrass:

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

Quindi, utilizza l'[API AWS Lambda](#) per aggiornare il parametro `handler` della configurazione della funzione, pubblicare una nuova versione della funzione e aggiornare l'alias. Per ulteriori informazioni, consulta [Funzione Versioni multiple e alias di funzioni AWS Lambda](#).

Se hai aggiunto la funzione al gruppo Greengrass per alias (consigliato), puoi ora ridistribuire il gruppo. In caso contrario, prima di distribuire il gruppo, devi puntare alla nuova versione o al nuovo alias della funzione nella definizione del gruppo e nelle sottoscrizioni.

Il software AWS IoT Greengrass Core non si avvia dopo il passaggio dall'esecuzione senza containerizzazione all'esecuzione in un contenitore Greengrass.

Soluzione: controlla che tutte le dipendenze container siano presenti.

Errore: la dimensione di spool deve essere almeno 262144 byte.

Soluzione: è possibile visualizzare questo errore quando il software AWS IoT Greengrass Core non si avvia. Apri il file `group.json` (situato in */greengrass-root*/ggc/deployment/group),

sostituisci i contenuti del file con un oggetto JSON vuoto ({}), ed esegui i comandi seguenti per avviare AWS IoT Greengrass:

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

Quindi, segui i passaggi nella procedura [the section called “Come aggiungere i messaggi alla cache nello storage locale”](#). Per la funzione `GGCloudSpooler`, assicurati di specificare un valore `GG_CONFIG_MAX_SIZE_BYTES` maggiore o uguale a 262144.

Errore: [ERROR]-Cloud messaging error: si è verificato un errore durante il tentativo di pubblicare un messaggio. {"errorString": "operation timed out"}

Soluzione: potresti visualizzare questo errore in `GGCloudSpooler.log` quando il core Greengrass non è in grado di inviare messaggi MQTT a AWS IoT Core. Ciò può verificarsi se l'ambiente core dispone di larghezza di banda limitata e latenza elevata. [Se utilizzi la versione AWS IoT Greengrass 1.10.2 o successiva, prova ad aumentare il valore nel file `config.json.mqttOperationTimeout`](#). Se la proprietà non è presente, aggiungerla all'oggetto `coreThing`. Per esempio:

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

Il valore predefinito è 5 e il valore minimo è 5.

<version>Errore: container_linux.go:344: l'avvio del processo del contenitore ha causato «process_linux.go:424: l'init del contenitore ha causato\" rootfs_linux.go:64: mount\\\"»///_ipc.sock: autorizzazione negata\\\"\". greengrass/ggc/socket/greengrass_ipc.sock\\\" to rootfs \\\"/greengrass/ggc/packages rootfs/merged\\\" at \\\"/greengrass_ipc.sock\\\" caused \\\"stat /greengrass/ggc/socket/greengrass

Soluzione: questo **runtime.log** errore potrebbe verificarsi AWS IoT Greengrass quando il software Core non si avvia. Questo si verifica se umask è maggiore di 0022. Per risolvere questo problema, devi impostare umask su 0022 o un valore inferiore. Il valore 0022 concede a tutti le autorizzazioni in lettura dei nuovi file per impostazione predefinita.

Errore: daemon Greengrass in esecuzione con PID: <process-id>. Alcuni componenti di sistema non sono stati avviati. Controlla se ci sono errori in 'runtime.log'.

Soluzione: è possibile visualizzare questo errore quando il software AWS IoT Greengrass Core non si avvia. Controlla `runtime.log` e `crash.log` per le informazioni di errore specifiche. Per ulteriori informazioni, consulta [the section called “Risoluzione dei problemi con i log”](#).

La shadow del dispositivo non si sincronizza con il cloud.

Soluzione: assicurati di disporre AWS IoT Greengrass delle autorizzazioni `iot:UpdateThingShadow` e delle `iot:GetThingShadow` azioni per il ruolo del [servizio Greengrass](#). Se il ruolo del servizio usa la policy gestita `AWSGreengrassResourceAccessRolePolicy`, queste autorizzazioni sono incluse per impostazione predefinita.

Per informazioni, consulta [Risoluzione dei problemi di timeout della sincronizzazione shadow](#).

ERRORE: non in grado di accettare la connessione TCP. accept tcp [::]:8000: accept4: troppi file aperti.

Soluzione: è possibile visualizzare questo errore nell'output di script greengrassd. Ciò può verificarsi se il limite dei descrittori di file per il software AWS IoT Greengrass Core ha raggiunto la soglia e deve essere aumentato.

Utilizzate il seguente comando e riavviate il software AWS IoT Greengrass Core.

```
ulimit -n 2048
```

 Note

In questo esempio, il limite è aumentato a 2048. Scegli un valore adatto per il tuo caso d'uso.

Errore: errore di esecuzione runtime: impossibile avviare il container lambda. container_linux.go:259: l'avvio del processo del container ha causato "process_linux.go:345: container init caused \"rootfs_linux.go:50: preparing rootfs caused \\\"permission denied\\\"\"".

Soluzione: AWS IoT Greengrass installatelo direttamente nella directory principale oppure assicuratevi che la directory in cui è installato il software AWS IoT Greengrass Core e le relative directory principali dispongano delle execute autorizzazioni per tutti.

Avviso: [WARN] - [5] GK Remote: errore durante il recupero dei dati della chiave pubblica: la chiave privata per non è ErrPrincipalNotConfigured impostata. MqttCertificate

Soluzione: AWS IoT Greengrass utilizza un gestore comune per convalidare le proprietà di tutti i principali di sicurezza. Questo avviso in runtime.log è previsto se non viene specificata una chiave privata personalizzata per il server MQTT locale. Per ulteriori informazioni, consulta [the section called "Principal di sicurezza"](#).

<account-id><role-name><region>Errore: autorizzazione negata durante il tentativo di utilizzare il ruolo arn:aws:iam: :role/ per accedere a s3 url https://-greengrass-updates.s3. <region><architecture><distribution-version>.amazonaws.com/core/ /greengrass-core- .tar.gz.

Soluzione: potresti visualizzare questo errore quando un aggiornamento (OTA) fallisce. over-the-air Nella politica del ruolo del firmatario, aggiungi l'obiettivo Regione AWS comeResource. Questo ruolo di firmatario viene utilizzato per preassegnare l'URL S3 per l'aggiornamento del software. AWS IoT Greengrass Per ulteriori informazioni, consulta [Ruolo firmatario URL S3](#).

Il AWS IoT Greengrass core è configurato per utilizzare un [proxy di rete](#) e la funzione Lambda non può effettuare connessioni in uscita.

Soluzione: a seconda del runtime e degli eseguibili utilizzati dalla funzione Lambda per creare connessioni, potresti ricevere anche errori di timeout della connessione. Assicurati che le funzioni Lambda utilizzino la configurazione proxy appropriata per connettersi tramite il proxy di rete. AWS IoT Greengrass passa la configurazione del proxy alle funzioni Lambda definite dall'utente tramite http_proxy le variabili di https_proxy ambiente, no_proxy e. È possibile accedervi come mostrato nel seguente frammento di codice Python.

```
import os
print(os.environ['http_proxy'])
```

Utilizzare la stessa capitalizzazione della variabile definita nell'ambiente, ad esempio utilizzare tutte lettere minuscole http_proxy o maiuscole HTTP_PROXY. Per queste variabili, AWS IoT Greengrass supporta entrambe.

Note

La maggior parte delle librerie comuni utilizzate per effettuare le connessioni (ad esempio i pacchetti boto3 o cURL e requests python) utilizzano queste variabili di ambiente per impostazione predefinita.

Il core si trova in un ciclo infinito di connessione-disconnessione. Il file `runtime.log` contiene una serie di voci di connessione e disconnessione.

Soluzione: ciò può accadere quando un altro dispositivo è hardcoded per utilizzare il nome dell'oggetto core come ID client per le connessioni MQTT a AWS IoT. Connessioni simultanee nella stessa unità Regione AWS e Account AWS devono utilizzare un client IDs unico. Per impostazione predefinita, il core utilizza il nome dell'oggetto core come ID client per queste connessioni.

Per risolvere questo problema, è possibile modificare l'ID client utilizzato dall'altro dispositivo per la connessione (scelta consigliata) oppure sovrascrivere il valore predefinito per il core.

Per sovrascrivere l'ID client predefinito per il dispositivo core

1. Eseguite il seguente comando per arrestare il demone Greengrass:

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

2. Apri `greengrass-root/config/config.json` per la modifica come utente su.
3. Nell'oggetto `coreThing`, aggiungere la proprietà `coreClientId` e impostare il valore sull'ID client personalizzato. Il valore deve essere costituito da un numero di caratteri compreso tra 1 e 128. Deve essere unico nella versione corrente Regione AWS per. Account AWS

```
"coreClientId": "MyCustomClientId"
```

4. Avvia il daemon.

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

Errore: impossibile avviare il container lambda. container_linux.go: 259: l'avvio del processo del container ha causato "process_linux.go: 345: container init caused\"rootfs_linux.go: 62: mounting \" proc\" to rootfs \"

Soluzione: su alcune piattaforme, potresti visualizzare questo errore `runtime.log` quando AWS IoT Greengrass tenta di montare il `/proc` file system per creare un contenitore Lambda. In alternativa, potresti visualizzare errori simili, ad esempio `operation not permitted` o `EPERM`. Questi errori possono verificarsi anche se i test eseguiti sulla piattaforma dallo script dello strumento di controllo delle dipendenze vengono superati.

Prova una delle seguenti soluzioni possibili:

- Abilitare l'opzione `CONFIG_DEVPTS_MULTIPLE_INSTANCES` nel kernel Linux.
- Impostare le opzioni di montaggio `/proc` sull'host solo su `rw,relatim`.
- Aggiornare il kernel Linux alla versione 4.9 o successiva.

 Note

Questo problema non è correlato al montaggio di `/proc` per l'accesso alle risorse locali.

[ERRORE] -errore di esecuzione in fase di esecuzione: impossibile avviare il contenitore lambda. <ggc-path>{"errorString": «impossibile inizializzare i supporti dei contenitori: impossibile mascherare la radice di greengrass nella directory superiore di overlay: impossibile creare il dispositivo maschera nella directory: il file esiste"}

Soluzione: potresti visualizzare questo errore in `runtime.log` quando la distribuzione fallisce. Questo errore si verifica se una funzione Lambda del AWS IoT Greengrass gruppo non è in grado di accedere alla `/usr` directory nel file system del core.

Per risolvere il problema, aggiungere una risorsa volume locale al gruppo e quindi distribuire il gruppo. Questa risorsa deve:

- Specificare `/usr` come percorso di origine e percorso di destinazione.
- Aggiungere automaticamente le autorizzazioni del gruppo OS del gruppo Linux che possiede la risorsa
- Diventa affiliato alla funzione Lambda e consenti l'accesso in sola lettura.

[ERRORE] -Distribuzione non riuscita. {"DeploymentId": <deployment-id>"«, «errorString»: «Processo di test del contenitore con <pid>pid non riuscito: stato del processo del contenitore: stato di uscita 1"}

Soluzione: potresti visualizzare questo errore in `runtime.log` quando la distribuzione fallisce. Questo errore si verifica se una funzione Lambda del AWS IoT Greengrass gruppo non è in grado di accedere alla `/usr` directory nel file system del core.

Puoi confermare che questo è il caso verificando la presenza `GGCanary.log` di errori aggiuntivi. Se la funzione Lambda non può accedere alla `/usr` directory, `GGCanary.log` conterrà il seguente errore:

```
[ERROR]-standard_init_linux.go:207: exec user process caused "no such file or directory"
```

Per risolvere il problema, aggiungere una risorsa volume locale al gruppo e quindi distribuire il gruppo. Questa risorsa deve:

- Specificare `/usr` come percorso di origine e percorso di destinazione.
- Aggiungere automaticamente le autorizzazioni del gruppo OS del gruppo Linux che possiede la risorsa
- Diventa affiliato alla funzione Lambda e consenti l'accesso in sola lettura.

Errore: [ERROR] -Errore di esecuzione di runtime: impossibile avviare il contenitore lambda. <ggc-version><ggc-version>{"errorString": «impossibile inizializzare i supporti del contenitore: impossibile creare l'overlay fs per il contenitore: montaggio dell'overlay su/////<ggc-version><ggc-version><ggc-version>/dns:/,greengrass/ggc/packages/rootfs/merged failed: failed to mount with args source=\"no_source\" dest=\"/greengrass/ggc/packages/rootfs/merged\" fstype=\"overlay\" flags=\"0\" data=\"lowerdir=/greengrass/ggc/packages//rootfs/work\"»: troppi livelli di collegamenti upperdir=/greengrass/ggc/packages simbolici \"} rootfs/upper,workdir=/greengrass/ggc/packages

Soluzione: è possibile che venga visualizzato questo errore nel file quando il software Core non si avvia. `runtime.log` AWS IoT Greengrass Questo problema potrebbe essere più comune sui sistemi operativi Debian.

Per risolvere il problema, procedere come segue:

1. Aggiorna il software AWS IoT Greengrass Core alla versione 1.9.3 o successiva. Questo dovrebbe risolvere automaticamente il problema.
2. Se l'errore persiste dopo l'aggiornamento del software AWS IoT Greengrass Core, imposta la `system.useOverlayWithTmpfs` proprietà su `true` nel [file config.json](#).

Example Esempio

```
{
  "system": {
    "useOverlayWithTmpfs": true
  },
  "coreThing": {
    "caPath": "root-ca.pem",
    "certPath": "cloud.pem.crt",
    "keyPath": "cloud.pem.key",
    ...
  },
  ...
}
```

 Note

La versione AWS IoT Greengrass del software Core è mostrata nel messaggio di errore. Per trovare la versione del kernel Linux, eseguire `uname -r`.

Errore: [DEBUG] - Impossibile ottenere route. Eliminare il messaggio.

Soluzione: controllare le sottoscrizioni nel gruppo e verificare l'esistenza della sottoscrizione elencata nel messaggio [DEBUG].

Errore: [Errno 24] Troppi file aperti <lambda-function>, [Errno 24] Troppi file aperti

Soluzione: potresti visualizzare questo errore nel file di registro della funzione Lambda se la funzione crea un'istanza `StreamManagerClient` nel gestore delle funzioni. Si consiglia di creare il client all'esterno del gestore. Per ulteriori informazioni, consulta [the section called "Utilizzalo StreamManagerClient per lavorare con gli stream"](#).

Errore: il server ds non è riuscito ad avviare l'ascolto di socket: listen unix<ggc-path>/ggc/socket/greengrass_ipc.sock: bind: argomento non valido

Soluzione: potresti visualizzare questo errore quando il software Core non si avvia. AWS IoT Greengrass Questo errore si verifica quando il software AWS IoT Greengrass Core viene installato in una cartella con un percorso di file lungo. Reinstalla il software AWS IoT Greengrass Core in una cartella con un percorso di file con meno di 79 byte, se non usi una [directory di scrittura](#), o 83 byte, se utilizzi una directory di scrittura.

[INFORMAZIONI] (Copier) aws.greengrass.StreamManager: stdout.
Causato da: com.fasterxml.jackson.databind.JsonMappingException:
Instant supera l'istante minimo o massimo

Quando aggiorni il software di AWS IoT Greengrass base alla versione 1.11.3, potresti visualizzare il seguente errore nei log dello stream manager se lo stream manager non si avvia.

```
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["lastExportTime"]
{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}
```

Se utilizzi una versione del software di AWS IoT Greengrass base precedente alla v1.11.3 e desideri eseguire l'aggiornamento a una versione successiva, utilizza un aggiornamento OTA per eseguire l'aggiornamento alla v1.11.4.

GPG error: <https://dnw9lb6lzp2d8.cloudfront.net> stable InRelease: The following signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass Master Key

Quando esegui `apt update` su un dispositivo in cui hai [installato il software di AWS IoT Greengrass base da un repository APT](#), potresti visualizzare il seguente errore.

```
Err:4 https://dnw9lb6lzp2d8.cloudfront.net stable InRelease
The following signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass
Master Key
Reading package lists... Done
W: GPG error: https://dnw9lb6lzp2d8.cloudfront.net stable InRelease: The following
signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass Master Key
```

Questo errore si verifica perché AWS IoT Greengrass non offre più la possibilità di installare o aggiornare il software di AWS IoT Greengrass base dal repository APT. Per eseguirlo

correttamente apt update, rimuovi il AWS IoT Greengrass repository dall'elenco delle fonti del dispositivo.

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

Problemi relativi alla distribuzione

Utilizza le informazioni seguenti per risolvere i problemi di distribuzione.

Problemi

- [La distribuzione corrente non funziona e si desidera ripristinare una distribuzione funzionante precedente.](#)
- [Visualizzi un errore 403 Forbidden sulla distribuzione nei log.](#)
- [Si verifica un ConcurrentDeployment errore quando si esegue il comando create-deployment per la prima volta.](#)
- [Errore: Greengrass non è autorizzato ad assumere il ruolo di servizio associato a questo account, oppure l'errore: Non riuscito: il ruolo di servizio TES non è associato a questo account.](#)
- [Errore: impossibile eseguire il passaggio di download nella distribuzione. errore durante il download: errore durante il download del file di definizione del gruppo:... x509: il certificato è scaduto o non è ancora valido](#)
- [La distribuzione non viene completata.](#)
- [Errore: impossibile trovare gli eseguibili java o java8 oppure l'errore: Implementazione <deployment-id> di tipo NewDeployment per gruppo <group-id> non riuscita errore: worker with <worker-id> failed to initialize with reason La versione Java installata deve essere maggiore o uguale a 8](#)
- [La distribuzione non viene completata e il file runtime.log contiene più voci "wait 1s for container to stop".](#)
- [La distribuzione non viene completata e runtime.log mostra il seguente messaggio di errore: "\[ERROR\]-Greengrass deployment error: failed to report deployment status back to cloud {"deploymentId": "<deployment-id>", "errorString": "Failed to initiate PUT, endpoint: https://<deployment-status>, error: Put https://<deployment-status>: proxyconnect tcp: x509: certificate signed by unknown authority"}"](#)

- [<path>Errore: la distribuzione <deployment-id>del tipo NewDeployment per il gruppo <group-id>non è riuscita. Errore durante l'elaborazione. group config non è valido: 112 o \[119 0\] non dispongono dell'autorizzazione rw sul file:.](#)
- [Errore: < list-of-function-arns > sono configurati per l'esecuzione come root ma Greengrass non è configurato per eseguire funzioni Lambda con autorizzazioni root.](#)
- [Errore: distribuzione <deployment-id>di tipo NewDeployment per gruppo <group-id>non riuscita Errore di distribuzione di Greengrass: impossibile eseguire la fase di download nella distribuzione. errore durante l'elaborazione: impossibile caricare il file di gruppo scaricato: impossibile trovare l'UID in base al nome utente, UserName: ggc_user: user: unknown user ggc_user.](#)
- [Errore: errore di esecuzione \[ERROR\]-runtime: impossibile avviare il container lambda. {"errorString": "failed to initialize container mounts: failed to mask greengrass root in overlay upper dir: failed to create mask device at directory <ggc-path>: file exists"}.](#)
- [Errore: distribuzione <deployment-id>di tipo NewDeployment per gruppo <group-id>non riuscita errore: avvio del processo non riuscito: container_linux.go:259: l'avvio del processo contenitore ha causato «process_linux.go:250: l'esecuzione del processo exec setns per init ha causato\ " wait: nessun processo figlio\ "».](#)
- [Errore: \[<host-prefix>WARN\] -MQTT \[client\] dial tcp: lookup -ats.iot. <region>.amazonaws.com: nessun host di questo tipo... \[ERROR\] -Errore di implementazione di Greengrass: impossibile riportare lo stato della distribuzione al cloud... net/http: richiesta annullata in attesa della connessione \(Client.Timeout superato in attesa delle intestazioni\)](#)

La distribuzione corrente non funziona e si desidera ripristinare una distribuzione funzionante precedente.

Soluzione: utilizza la AWS IoT console o l' AWS IoT Greengrass API per ridistribuire una distribuzione funzionante precedente. In questo modo viene distribuita la versione del gruppo corrispondente sul dispositivo core.

Per ridistribuire una distribuzione (console)

1. Nella pagina di configurazione del gruppo, scegli la scheda Distribuzioni. Questa pagina mostra la cronologia di distribuzione per il gruppo, incluse la data e l'ora, la versione del gruppo e lo stato di ogni tentativo di distribuzione.

2. Trova la riga contenente la distribuzione che desideri ridistribuire. Seleziona la distribuzione che desideri ridistribuire e scegli Ridistribuisce.

Deployments	Group history overview			By deployment
Subscriptions	Deployed	Version	Status	
Cores				
Devices	Jul 1, 2019 1:56:49 PM -0700	8dd1d899-4ac9-4f5d-afe4-22de086efc62	Successfully complet...	...
Lambdas	Jul 1, 2019 1:41:47 PM -0700	4ad66e5d-3808-446b-940a-b1a788898382	Successfully complet...	...
Resources				
Connectors	Jun 18, 2019 8:16:02 AM -0700	1f3870b6-850e-4c97-8018-c872e17b235b	Failed	...

Per ridistribuire una distribuzione (CLI)

1. Utilizzalo [ListDeployments](#) per trovare l'ID della distribuzione che desideri ridistribuire. Per esempio:

```
aws greengrass list-deployments --group-id 74d0b623-c2f2-4cad-9acc-ef92f61fcdf7
```

Il comando restituisce l'elenco delle distribuzioni per il gruppo.

```
{
  "Deployments": [
    {
      "DeploymentId": "8d179428-f617-4a77-8a0c-3d61fb8446a6",
      "DeploymentType": "NewDeployment",
      "GroupArn": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/groups/74d0b623-c2f2-4cad-9acc-ef92f61fcdf7/versions/8dd1d899-4ac9-4f5d-afe4-22de086efc62",
      "CreatedAt": "2019-07-01T20:56:49.641Z"
    },
    {
      "DeploymentId": "f8e4c455-8ac4-453a-8252-512dc3e9c596",
      "DeploymentType": "NewDeployment",
      "GroupArn": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/groups/74d0b623-c2f2-4cad-9acc-ef92f61fcdf7/versions/4ad66e5d-3808-446b-940a-b1a788898382",
      "CreatedAt": "2019-07-01T20:41:47.048Z"
    },
    {
      "DeploymentId": "e4aca044-bbd8-41b4-b697-930ca7c40f3e",

```

```

        "DeploymentType": "NewDeployment",
        "GroupArn": "arn:aws:greengrass:us-west-2::123456789012:/greengrass/
groups/74d0b623-c2f2-4cad-9acc-ef92f61fcaf7/versions/1f3870b6-850e-4c97-8018-
c872e17b235b",
        "CreatedAt": "2019-06-18T15:16:02.965Z"
    }
]
}

```

Note

Questi AWS CLI comandi utilizzano valori di esempio per il gruppo e l'ID di distribuzione. Quando esegui i comandi, assicurati di sostituire i valori di esempio.

2. Utilizzato [CreateDeployment](#) per ridistribuire la distribuzione di destinazione. Imposta il tipo di distribuzione su Redeployment. Per esempio:

```

aws greengrass create-deployment --deployment-type Redeployment \
  --group-id 74d0b623-c2f2-4cad-9acc-ef92f61fcaf7 \
  --deployment-id f8e4c455-8ac4-453a-8252-512dc3e9c596

```

Il comando restituisce l'ARN e l'ID della nuova distribuzione.

```

{
  "DeploymentId": "f9ed02b7-c28e-4df6-83b1-e9553ddd0fc2",
  "DeploymentArn": "arn:aws:greengrass:us-west-2::123456789012:/greengrass/
groups/74d0b623-c2f2-4cad-9acc-ef92f61fcaf7/deployments/f9ed02b7-c28e-4df6-83b1-
e9553ddd0fc2"
}

```

3. Utilizzare [GetDeploymentStatus](#) per ottenere lo stato della distribuzione.

Visualizzi un errore 403 Forbidden sulla distribuzione nei log.

Soluzione: assicurati che la policy del AWS IoT Greengrass core nel cloud includa tra "greengrass:*" le azioni consentite.

Si verifica un `ConcurrentDeployment` errore quando si esegue il comando `create-deployment` per la prima volta.

Soluzione: potrebbe essere in corso una distribuzione. Puoi eseguire [get-deployment-status](#) per vedere se una distribuzione è stata creata. In caso contrario, prova a creare nuovamente la distribuzione.

Errore: Greengrass non è autorizzato ad assumere il ruolo di servizio associato a questo account, oppure l'errore: Non riuscito: il ruolo di servizio TES non è associato a questo account.

Soluzione: è possibile visualizzare questo errore quando la distribuzione ha esito negativo. Verifica che un ruolo di servizio Greengrass sia associato al tuo Account AWS ruolo attuale. Regione AWS Per ulteriori informazioni, consulta [the section called “Gestione del ruolo del servizio \(CLI\)”](#) o [the section called “Gestione del ruolo del servizio \(console\)”](#).

Errore: impossibile eseguire il passaggio di download nella distribuzione.
errore durante il download: errore durante il download del file di definizione del gruppo:... x509: il certificato è scaduto o non è ancora valido

Soluzione: è possibile visualizzare questo errore in `runtime.log` quando la distribuzione ha esito negativo. Se viene visualizzato un errore `Deployment failed` che contiene il messaggio `x509: certificate has expired or is not yet valid`, controllare l'orologio del dispositivo. I certificati TLS e X.509 forniscono una base sicura per la creazione di sistemi IoT, ma richiedono tempi precisi su server e client. I dispositivi IoT devono avere l'ora corretta (entro 15 minuti) prima di tentare di connettersi a AWS IoT Greengrass o ad altri servizi TLS che utilizzano certificati server. Per ulteriori informazioni, consulta [Using Device Time per convalidare i certificati del AWS IoT server](#) sull'Internet of Things sul blog AWS ufficiale.

La distribuzione non viene completata.

Soluzione: eseguire quanto segue:

- Assicurati che il AWS IoT Greengrass demone sia in esecuzione sul tuo dispositivo principale. Nel terminale del dispositivo principale, esegui i seguenti comandi per verificare se il demone è in esecuzione e avvialo, se necessario.

1. Per controllare se il daemon è in esecuzione:

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

Se l'output contiene una voce `root` per `/greengrass/ggc/packages/1.11.6/bin/daemon`, allora il daemon è in esecuzione.

La versione indicata nel percorso dipende dalla versione del software AWS IoT Greengrass Core installata sul dispositivo principale.

2. Per avviare il demone:

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

- Assicurati che il dispositivo core sia collegato e gli endpoint di connessione core siano configurati correttamente.

Errore: impossibile trovare gli eseguibili java o java8 oppure l'errore: Implementazione <deployment-id>di tipo NewDeployment per gruppo <group-id>non riuscita errore: worker with <worker-id>failed to initialize with reason La versione Java installata deve essere maggiore o uguale a 8

Soluzione: se stream manager è abilitato per il AWS IoT Greengrass core, è necessario installare il runtime Java 8 sul dispositivo principale prima di distribuire il gruppo. Per ulteriori informazioni, consulta i [requisiti](#) per Gestore di flussi. Lo stream manager è abilitato per impostazione predefinita quando si utilizza il flusso di lavoro di creazione del gruppo predefinito nella AWS IoT console per creare un gruppo.

In alternativa, disabilita stream manager e quindi distribuisce il gruppo. Per ulteriori informazioni, consulta [the section called “Configurazione delle impostazioni \(console\)”](#).

La distribuzione non viene completata e il file `runtime.log` contiene più voci `"wait 1s for container to stop"`.

Soluzione: esegui i seguenti comandi nel terminale del dispositivo principale per riavviare il AWS IoT Greengrass demone.

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

La distribuzione non viene completata e **`runtime.log`** mostra il seguente messaggio di errore: `"[ERROR]-Greengrass deployment error: failed to report deployment status back to cloud {"deploymentId": "<deployment-id>", "errorString": "Failed to initiate PUT, endpoint: https://<deployment-status>, error: Put https://<deployment-status>: proxyconnect tcp: x509: certificate signed by unknown authority"}"`

Soluzione: questo errore potrebbe essere visualizzato in `runtime.log` quando Greengrass Core è configurato per utilizzare una connessione proxy HTTPS e la catena di certificati del server proxy non è attendibile nel sistema. Per provare a risolvere questo problema, aggiungere la catena di certificati al certificato CA radice. Greengrass Core aggiunge i certificati di questo file al pool di certificati utilizzato per l'autenticazione TLS nelle connessioni HTTPS e MQTT con AWS IoT Greengrass.

L'esempio seguente mostra un certificato CA del server proxy aggiunto al file del certificato della CA radice:

```
# My proxy CA  
-----BEGIN CERTIFICATE-----  
MIIIEFTCCAv2gAwIQWgIVAMHSAzWG/5YVRYtRQ0xXUTEpHuEmApzGCSqGSIB3DQEK  
\nCwUAhUL9MQswCQwJVUzEPMAVUzEYMBYGA1UECgwP1hem9uLmNvbSBjbMuMRww  
... content of proxy CA certificate ...  
+vHIR1t0e5JAm5\noTIZGoFbK82A0/n07f/t5PSIDAIm9V3Gc3pSXxCCAQoFYnui  
GaPUlGk1gCE84a0X\n7Rp/1ND/PuMZ/s8Yj1kY2NmYmNjMCAXDTE5MTEyN2cM216  
gJMIADggEPADf2/m45hzEXAMPLE=  
-----END CERTIFICATE-----
```

```
# Amazon Root CA 1
-----BEGIN CERTIFICATE-----
MIIDQTCCAimgF6AwIBAgITBmyfz/5mjAo54vB4ikPmljZKyjANJmApzyMZFo6qBg
ADA5MQswCQYDVQQGEwJVUzEPMA0tMVT8QtPHRh8jrdkGA1UEChMGMGdV3QQDExBBKW
... content of root CA certificate ...
o/ufQJQWUCyziar1hem9uMRkwFwYVPSHCb2XV4cdFyQzR1KldZwgJcIQ6XUDgHaa
5MsI+yMRQ+hDaXJioblDxgjUka642M4UwtBV8oK2xJNDd2ZhwLnoQdeXeGADKkpy
rqXRfKoQnoZsG4q5WTP46EXAMPLE
-----END CERTIFICATE-----
```

Per impostazione predefinita, il file del certificato della CA radice si trova in `/greengrass-root/certs/root.ca.pem`. Per trovare la posizione sul dispositivo principale, controllare la proprietà `crypto.caPath` in [config.json](#).

Note

greengrass-root rappresenta il percorso in cui il software AWS IoT Greengrass Core è installato sul dispositivo. In genere, questa è la directory `/greengrass`.

<path>Errore: la distribuzione <deployment-id> del tipo NewDeployment per il gruppo <group-id> non è riuscita. Errore durante l'elaborazione. group config non è valido: 112 o [119 0] non dispongono dell'autorizzazione rw sul file:.

Soluzione: assicurati che il gruppo proprietario della directory disponga delle autorizzazioni di lettura e scrittura per la *<path>* directory.

Errore: < list-of-function-arns > sono configurati per l'esecuzione come root ma Greengrass non è configurato per eseguire funzioni Lambda con autorizzazioni root.

Soluzione: è possibile visualizzare questo errore in `runtime.log` quando la distribuzione ha esito negativo. Assicurati di aver configurato AWS IoT Greengrass per consentire l'esecuzione delle

funzioni Lambda con i permessi di root. Cambia il valore di `allowFunctionsToRunAsRoot` in `greengrass_root/config/config.json` to `yes` o modifica la funzione Lambda per eseguirla come un altro utente/gruppo. Per ulteriori informazioni, consulta [the section called “Esecuzione di una funzione Lambda come utente root”](#).

Errore: distribuzione <deployment-id>di tipo NewDeployment per gruppo <group-id>non riuscita Errore di distribuzione di Greengrass: impossibile eseguire la fase di download nella distribuzione. errore durante l'elaborazione: impossibile caricare il file di gruppo scaricato: impossibile trovare l'UID in base al nome utente, Username: ggc_user: user: unknown user ggc_user.

Soluzione: se l'[identità di accesso predefinita del AWS IoT Greengrass gruppo utilizza gli account](#) di sistema standard, l'utente e il gruppo devono essere presenti sul dispositivo. `ggc_user` `ggc_group` Per istruzioni che mostrano come aggiungere l'utente e il gruppo, consulta questa [fase](#). Assicurati di inserire i nomi esattamente come mostrato.

Errore: errore di esecuzione [ERROR]-runtime: impossibile avviare il container lambda. {"errorString": "failed to initialize container mounts: failed to mask greengrass root in overlay upper dir: failed to create mask device at directory <ggc-path>: file exists"}

Soluzione: è possibile visualizzare questo errore in `runtime.log` quando la distribuzione ha esito negativo. Questo errore si verifica se una funzione Lambda del gruppo Greengrass non può accedere alla `/usr` directory nel file system del core. Per risolvere il problema, aggiungere una [risorsa volume locale](#) al gruppo e quindi distribuire il gruppo. La risorsa deve:

- Specificare `/usr` come percorso di origine e percorso di destinazione.
- Aggiungere automaticamente le autorizzazioni del gruppo OS del gruppo Linux che possiede la risorsa
- Diventa affiliato alla funzione Lambda e consenti l'accesso in sola lettura.

Errore: distribuzione <deployment-id>di tipo NewDeployment per gruppo <group-id>non riuscita errore: avvio del processo non riuscito: container_linux.go:259: l'avvio del processo contenitore ha causato «process_linux.go:250: l'esecuzione del processo exec setns per init ha causato\" wait: nessun processo figlio\" ».

Soluzione: è possibile visualizzare questo errore quando la distribuzione ha esito negativo. Riprova a eseguire la distribuzione.

Errore: [<host-prefix>WARN] -MQTT [client] dial tcp: lookup -ats.iot.<region>.amazonaws.com: nessun host di questo tipo... [ERROR] - Errore di implementazione di Greengrass: impossibile riportare lo stato della distribuzione al cloud... net/http: richiesta annullata in attesa della connessione (Client.Timeout superato in attesa delle intestazioni)

Soluzione: questo potrebbe essere visualizzato se utilizzi `systemd-resolved`, che abilita l'impostazione DNSSEC per impostazione predefinita. Di conseguenza, molti domini pubblici non vengono riconosciuti. I tentativi di raggiungere l' AWS IoT Greengrass endpoint non riescono a trovare l'host, quindi le distribuzioni rimangono invariate. In Progress

Puoi utilizzare i comandi e l'output seguenti per verificare questo problema. Sostituisci il *region* segnato negli endpoint con il tuo. Regione AWS

```
$ ping greengrass-ats.iot.region.amazonaws.com
ping: greengrass-ats.iot.region.amazonaws.com: Name or service not known
```

```
$ systemd-resolve greengrass-ats.iot.region.amazonaws.com
greengrass-ats.iot.region.amazonaws.com: resolve call failed: DNSSEC validation failed:
failed-auxiliary
```


Una possibile soluzione è disabilitare DNSSEC. Quando DNSSEC è `false`, le ricerche DNS non vengono convalidate DNSSEC. Per ulteriori informazioni, consulta questo [problema noto](#) per `systemd`.

1. Aggiungere `DNSSEC=false` a `/etc/systemd/resolved.conf`.
2. Riavviare `systemd-resolved`.

Per informazioni su `resolved.conf` e DNSSEC, esegui `man resolved.conf` nel terminale.

Problemi relativi alla creazione del gruppo e della funzione

Utilizzate le seguenti informazioni per risolvere i problemi relativi alla creazione di un AWS IoT Greengrass gruppo o di una funzione Greengrass Lambda.

Problemi

- [Errore: la configurazione 'IsolationMode' per il gruppo non è valida.](#)
- [Errore: la configurazione 'IsolationMode' per la funzione con arn non <function-arn>è valida.](#)
- [Errore: la MemorySize configurazione per la funzione con arn <function-arn>non è consentita in =. IsolationMode NoContainer](#)
- [Errore: la configurazione di Access Sysfs per la funzione con arn <function-arn>non è consentita in =. IsolationMode NoContainer](#)
- [Errore: la MemorySize configurazione per la funzione con arn <function-arn>è richiesta in IsolationMode =. GreengrassContainer](#)
- [Errore: la funzione <function-arn>si riferisce a una risorsa <resource-type>di tipo non consentito in IsolationMode =NoContainer.](#)
- [Errore: la configurazione dell'esecuzione per la funzione con arn <function-arn> non è consentita.](#)

Errore: la configurazione 'IsolationMode' per il gruppo non è valida.

Soluzione: questo errore si verifica quando il valore `IsolationMode` in `DefaultConfig` di `function-definition-version` non è supportato. I valori supportati sono `GreengrassContainer` e `NoContainer`.

Errore: la configurazione 'IsolationMode' per la funzione con arn non <function-arn> è valida.

Soluzione: questo errore si verifica quando il valore `IsolationMode` in <function-arn> di `function-definition-version` non è supportato. I valori supportati sono `GreengrassContainer` e `NoContainer`.

Errore: la `MemorySize` configurazione per la funzione con arn <function-arn> non è consentita in `=`. `IsolationMode NoContainer`

Soluzione: questo errore si verifica quando si specifica un `MemorySize` valore e si sceglie di eseguirlo senza containerizzazione. Le funzioni Lambda eseguite senza containerizzazione non possono avere limiti di memoria. Puoi rimuovere il limite o modificare la funzione Lambda per eseguirla in un AWS IoT Greengrass contenitore.

Errore: la configurazione di `Access Sysfs` per la funzione con arn <function-arn> non è consentita in `=`. `IsolationMode NoContainer`

Soluzione: questo errore si verifica quando si specifica `true` for `AccessSysfs` e si sceglie di eseguirlo senza containerizzazione. Le funzioni Lambda eseguite senza containerizzazione devono avere il codice aggiornato per accedere direttamente al file system e non possono essere utilizzate. `AccessSysfs` È possibile specificare un valore di `false` for `AccessSysfs` oppure modificare la funzione Lambda per eseguirla in un AWS IoT Greengrass contenitore.

Errore: la `MemorySize` configurazione per la funzione con arn <function-arn> è richiesta in `IsolationMode =`. `GreengrassContainer`

Soluzione: questo errore si verifica perché non è stato specificato un `MemorySize` limite per una funzione Lambda in esecuzione in un AWS IoT Greengrass contenitore. Specifica un valore `MemorySize` per risolvere l'errore.

Errore: la funzione <function-arn> si riferisce a una risorsa <resource-type> di tipo non consentito in IsolationMode =NoContainer.

Soluzione: non è possibile accedere a `Local.DeviceLocal.Volume`, `ML_Model.SageMaker.JobML_Model.S3_Object`, o tipi di `S3_Object.Generic_Archive` risorse quando si esegue una funzione Lambda senza containerizzazione. Se hai bisogno di questi tipi di risorse, devi eseguirli in un contenitore. AWS IoT Greengrass Puoi anche accedere direttamente ai dispositivi locali durante l'esecuzione senza containerizzazione modificando il codice nella funzione Lambda.

Errore: la configurazione dell'esecuzione per la funzione con arn <function-arn> non è consentita.

Soluzione: questo errore si verifica quando si crea una funzione Lambda di sistema con `GGIPDetector` o `GGCloudSpooler` e la configurazione o è stata specificata `IsolationMode`. `RunAs` È necessario omettere i `Execution` parametri per questa funzione Lambda di sistema.

Problemi di individuazione

Utilizza le seguenti informazioni per risolvere i problemi relativi al servizio Discovery. AWS IoT Greengrass

Problemi

- [Errore: il dispositivo è membro di troppi gruppi, è possibile che i dispositivi non siano presenti in più di 10 gruppi](#)

Errore: il dispositivo è membro di troppi gruppi, è possibile che i dispositivi non siano presenti in più di 10 gruppi

Soluzione: si tratta di una limitazione nota. Un [dispositivo client](#) può far parte di un massimo di 10 gruppi.

Problemi relativi alle risorse di Machine Learning

Utilizza le seguenti informazioni per risolvere i problemi relativi alle risorse di machine learning.

Problemi

- [MLModelProprietario non valido: GroupOwnerSetting viene fornito nella risorsa del modello ML, ma non è presente GroupOwner o non GroupPermission è presente](#)
- [NoContainer la funzione non può configurare l'autorizzazione quando si collegano risorse di Machine Learning. <function-arn>si riferisce alla risorsa Machine Learning <resource-id>con autorizzazione <ro/rw> nella politica di accesso alle risorse.](#)
- [La funzione <function-arn>si riferisce alla risorsa di Machine Learning <resource-id>con autorizzazione mancante in entrambe ResourceAccessPolicy le risorse OwnerSetting.](#)
- [La funzione <function-arn>si riferisce alla risorsa Machine Learning <resource-id>con autorizzazione\ "rw\», mentre l'impostazione del proprietario della risorsa consente GroupPermission solo\ "ro\».](#)
- [NoContainer La funzione <function-arn>si riferisce alle risorse del percorso di destinazione annidato.](#)
- [Lambda <function-arn> ottiene l'accesso alla risorsa <resource-id> condividendo lo stesso ID del proprietario del gruppo](#)

MLModelProprietario non valido: GroupOwnerSetting viene fornito nella risorsa del modello ML, ma non è presente GroupOwner o non GroupPermission è presente

Soluzione: viene visualizzato questo errore se una risorsa di machine learning contiene l'[ResourceDownloadOwnerSetting](#) oggetto ma il requisito GroupOwner o la GroupPermission proprietà non sono definiti. Per risolvere questo problema, definisci la proprietà mancante.

NoContainer la funzione non può configurare l'autorizzazione quando si collegano risorse di Machine Learning. <function-arn> si riferisce alla risorsa Machine Learning <resource-id> con autorizzazione <ro/rw> nella politica di accesso alle risorse.

Soluzione: viene visualizzato questo errore se una funzione Lambda non containerizzata specifica autorizzazioni a livello di funzione per una risorsa di machine learning. Le funzioni non containerizzate devono ereditare le autorizzazioni dalle autorizzazioni del proprietario della risorsa definite nella risorsa di machine learning. Per risolvere questo problema, scegli di [ereditare le autorizzazioni del proprietario della risorsa](#) (console) o [rimuovere le autorizzazioni dalla politica di accesso alle risorse \(API\) della funzione Lambda](#).

La funzione <function-arn> si riferisce alla risorsa di Machine Learning <resource-id> con autorizzazione mancante in entrambe ResourceAccessPolicy le risorse OwnerSetting.

Soluzione: viene visualizzato questo errore se le autorizzazioni per la risorsa di machine learning non sono configurate per la funzione Lambda o la risorsa allegata. Per risolvere questo problema, configura le autorizzazioni nella [ResourceAccessPolicy](#) proprietà per la funzione Lambda o nella proprietà per [OwnerSetting](#) la risorsa.

La funzione <function-arn> si riferisce alla risorsa Machine Learning <resource-id> con autorizzazione \ "rw\», mentre l'impostazione del proprietario della risorsa consente GroupPermission solo \ "ro\».

Soluzione: viene visualizzato questo errore se le autorizzazioni di accesso definite per la funzione Lambda allegata superano le autorizzazioni del proprietario della risorsa definite per la risorsa di machine learning. Per risolvere questo problema, imposta autorizzazioni più restrittive per la funzione Lambda o autorizzazioni meno restrittive per il proprietario della risorsa.

NoContainer La funzione <function-arn> si riferisce alle risorse del percorso di destinazione annidato.

Soluzione: viene visualizzato questo errore se più risorse di machine learning collegate a una funzione Lambda non containerizzata utilizzano lo stesso percorso di destinazione o un percorso di destinazione annidato. Per risolvere questo problema, specifica percorsi di destinazione separati per le risorse.

Lambda <function-arn> ottiene l'accesso alla risorsa <resource-id> condividendo lo stesso ID del proprietario del gruppo

Soluzione: viene visualizzato questo errore `runtime.log` se viene specificato lo stesso gruppo di sistema operativo come identità [Esegui come](#) identità della funzione Lambda e [proprietario della risorsa](#) per una risorsa di machine learning, ma la risorsa non è associata alla funzione Lambda. Questa configurazione fornisce alla funzione Lambda autorizzazioni implicite che può utilizzare per accedere alla risorsa senza autorizzazione. AWS IoT Greengrass

Per risolvere questo problema, usa un gruppo di sistemi operativi diverso per una delle proprietà o collega la risorsa di machine learning alla funzione Lambda.

AWS IoT Greengrass problemi principali di Docker

Utilizza le seguenti informazioni per risolvere i problemi relativi all'esecuzione di un AWS IoT Greengrass core in un contenitore Docker.

Problemi

- [Errore: opzioni sconosciute: -. no-include-email](#)
- [Attenzione: IPv4 è disattivata. Networking non funzionerà.](#)
- [Errore: un firewall sta bloccando la condivisione di file tra finestre e contenitori.](#)
- [Errore: si è verificato un errore \(AccessDeniedException\) durante la chiamata dell' GetAuthorizationToken operazione: User: arn:aws:iam: :user/ <account-id><user-name>non è autorizzato a eseguire: ecr: on resource: * GetAuthorizationToken](#)
- [Errore: impossibile creare container per il servizio greengrass: Conflict. Il nome del contenitore «/aws-iot-greengrass" è già in uso.](#)
- [Errore: \[FATAL\]-Non è riuscito a reimpostare lo spazio dei nomi del montaggio del thread a causa di un errore inaspettato: "operation not permitted". Per mantenere la coerenza, GGC si arresterà e dovrà essere riavviato manualmente.](#)

Errore: opzioni sconosciute: -. no-include-email

Soluzione: questo errore può verificarsi quando si esegue il comando `aws ecr get-login`. Assicurati di avere installata la AWS CLI versione più recente (ad esempio, esegui: `pip install awscli --upgrade --user`). Se utilizzi Windows e hai installato l'interfaccia a riga di comando utilizzando il programma di installazione di MSI, è necessario ripetere il processo di installazione. Per ulteriori informazioni, vedere [Installazione di AWS Command Line Interface su Microsoft Windows](#) nella Guida per l'AWS Command Line Interface utente.

Attenzione: IPv4 è disattivata. Networking non funzionerà.

Soluzione: è possibile che venga visualizzato questo avviso o un messaggio simile quando si esegue AWS IoT Greengrass su un computer Linux. [Abilita l'inoltro di IPv4 rete come descritto in questo passaggio](#). AWS IoT Greengrass l'implementazione nel cloud e le comunicazioni MQTT non funzionano quando l'IPv4 inoltro non è abilitato. Per ulteriori informazioni, consulta [Configurazione di parametri kernel associati a uno spazio dei nomi \(sysctls\) durante il runtime](#) nella documentazione Docker.

Errore: un firewall sta bloccando la condivisione di file tra finestre e contenitori.

Soluzione: è possibile ricevere questo errore o un messaggio `Firewall Detected` quando viene eseguito Docker su un computer Windows. Questo errore può inoltre verificarsi se sei connesso a una rete privata virtuale (VPN, Virtual Private Network) e le impostazioni di rete impediscono il montaggio dell'unità condivisa. In tal caso, disattivare la rete VPN e riavviare il container Docker.

Errore: si è verificato un errore (`AccessDeniedException`) durante la chiamata dell' `GetAuthorizationToken` operazione: User: `arn:aws:iam: :user/<account-id><user-name>` non è autorizzato a eseguire: `ecr: on resource: *GetAuthorizationToken`

Potresti ricevere questo errore durante l'esecuzione del `aws ecr get-login-password` comando se non disponi di autorizzazioni sufficienti per accedere a un repository Amazon ECR. Per ulteriori informazioni, consulta [Esempi di policy per i repository di Amazon ECR](#) e [Accesso a un repository Amazon ECR nella](#) Amazon ECR User Guide.

Errore: impossibile creare container per il servizio greengrass: Conflict. Il nome del contenitore «/aws-iot-greengrass» è già in uso.

Soluzione: questo può verificarsi quando il nome del container viene utilizzato da un container precedente. Per risolvere questo problema, eseguire il comando seguente per rimuovere il vecchio container Docker:

```
docker rm -f $(docker ps -a -q -f "name=aws-iot-greengrass")
```


Errore: [FATAL]-Non è riuscito a reimpostare lo spazio dei nomi del montaggio del thread a causa di un errore inaspettato: "operation not permitted". Per mantenere la coerenza, GGC si arresterà e dovrà essere riavviato manualmente.

Soluzione: questo errore `runtime.log` può verificarsi quando si tenta di distribuire una funzione `GreengrassContainer Lambda` su un core in esecuzione in AWS IoT Greengrass un contenitore Docker. Attualmente, solo le funzioni `NoContainer Lambda` possono essere implementate in un contenitore Greengrass Docker.

Per risolvere questo problema, [assicurati che tutte le funzioni Lambda siano in NoContainer modalità](#) e avvia una nuova distribuzione. Quindi, quando avvii il contenitore, non monta in associazione la deployment directory esistente sul contenitore Docker principale AWS IoT Greengrass . Al contrario, crea una directory deployment vuota al suo posto ed esegui il montaggio vincolato nel container Docker. Ciò consente al nuovo contenitore Docker di ricevere la distribuzione più recente con le funzioni Lambda in esecuzione in NoContainer modalità.

Per ulteriori informazioni, consulta [the section called "Esegui AWS IoT Greengrass in un contenitore Docker"](#).

Risoluzione dei problemi con i log

È possibile configurare le impostazioni di registrazione per un gruppo Greengrass, ad esempio se inviare i log a Logs CloudWatch , archiviare i log nel file system locale o entrambi. Per ottenere informazioni dettagliate durante la risoluzione dei problemi, è possibile modificare temporaneamente il livello di registrazione in DEBUG. Le modifiche alle impostazioni di registrazione diventano effettive quando si distribuisce il gruppo. Per ulteriori informazioni, consulta [the section called "Configurare la registrazione per AWS IoT Greengrass"](#).

Nel file system locale, AWS IoT Greengrass archivia i log nelle seguenti posizioni. La lettura dei log nel file system richiede autorizzazioni root.

greengrass-root/ggc/var/log/crash.log

Mostra i messaggi generati quando un AWS IoT Greengrass core si blocca.

greengrass-root/ggc/var/log/system/runtime.log

Mostra i messaggi che indicano quale componente restituisce un errore.

greengrass-root/ggc/var/log/system/

Contiene tutti i log dei componenti del AWS IoT Greengrass sistema, come il gestore dei certificati e il gestore delle connessioni. Utilizzando i messaggi in *ggc/var/log/system/* and *ggc/var/log/system/runtime.log*, dovresti essere in grado di scoprire quale errore si è verificato nei componenti AWS IoT Greengrass del sistema.

greengrass-root/ggc/var/log/system/localwatch/

Contiene i log del AWS IoT Greengrass componente che gestisce il caricamento dei log di Greengrass su Logs. CloudWatch Se non riesci a visualizzare i log in di Greengrass CloudWatch, puoi utilizzare questi registri per la risoluzione dei problemi.

greengrass-root/ggc/var/log/user/

Contiene tutti i log delle funzioni Lambda definite dall'utente. Controlla questa cartella per trovare i messaggi di errore delle funzioni Lambda locali.

Note

Come impostazione predefinita, *greengrass-root* è la directory /greengrass. Se è configurata una [directory di scrittura](#), i log si trovano in tale directory.

Se i log sono configurati per essere archiviati nel cloud, usa CloudWatch Logs per visualizzare i messaggi di log. *crash.log* si trova solo nei log del file system sul dispositivo principale AWS IoT Greengrass.

Se AWS IoT è configurato per la scrittura di registri CloudWatch, controllali se si verificano errori di connessione quando i componenti del sistema tentano di connettersi a AWS IoT.

Per ulteriori informazioni sulla AWS IoT Greengrass registrazione, vedere. [the section called "Monitoraggio con i log AWS IoT Greengrass"](#)

Note

I log del software AWS IoT Greengrass Core v1.0 sono archiviati nella directory.
greengrass-root/var/log

Risoluzione dei problemi di storage

Quando lo storage di file locale è pieno, alcuni componenti potrebbero iniziare a restituire errori:

- Gli aggiornamenti shadow locali non si verificano.
- I nuovi certificati AWS IoT Greengrass principali del server MQTT non possono essere scaricati localmente.
- Le distribuzioni hanno esito negativo.

È sempre consigliabile conoscere la quantità di spazio libero disponibile in locale. È possibile calcolare lo spazio libero in base alle dimensioni delle funzioni Lambda distribuite, alla configurazione di registrazione ([the section called “Risoluzione dei problemi con i log”](#) vedi) e al numero di shadow archiviate localmente.

Risoluzione dei problemi relativi ai messaggi

Tutti i messaggi inviati localmente AWS IoT Greengrass vengono inviati con QoS 0. Per impostazione predefinita, AWS IoT Greengrass archivia i messaggi in una coda in memoria. Pertanto, i messaggi non elaborati andranno persi quando il core Greengrass viene riavviato, ad esempio dopo la distribuzione di un gruppo o il riavvio di un dispositivo. Tuttavia, è possibile configurare AWS IoT Greengrass (versione 1.6 o successiva) la memorizzazione nella cache dei messaggi nel file system in modo che persistano anche dopo i riavvii principali. Puoi inoltre configurare le dimensioni della coda. Se configuri le dimensioni della coda, assicurati che siano maggiori o uguali a 262144 byte (256 KB). In caso contrario, AWS IoT Greengrass potrebbe non avviarsi correttamente. Per ulteriori informazioni, consulta [the section called “Coda di messaggi MQTT”](#).

Note

Se utilizzi la coda in memoria predefinita, è consigliabile distribuire gruppi o riavviare il dispositivo nel momento in cui l'interruzione di servizio non crea particolari problemi.

Puoi inoltre configurare il core per stabilire sessioni persistenti con AWS IoT. Ciò consente al core di ricevere messaggi inviati Cloud AWS mentre il core è offline. Per ulteriori informazioni, consulta [the section called “Sessioni persistenti MQTT con AWS IoT Core”](#).

Risoluzione dei problemi di timeout della sincronizzazione shadow

Ritardi significativi nelle comunicazioni tra un dispositivo core Greengrass e il cloud potrebbero causare un errore nella sincronizzazione shadow a causa di un timeout. In questo caso, ti consigliamo di controllare le voci di log simili alle seguenti:

```
[2017-07-20T10:01:58.006Z][ERROR]-cloud_shadow_client.go:57,Cloud shadow
client error: unable to get cloud shadow what_the_thing_is_named for
synchronization. Get https://1234567890abcd.iot.us-west-2.amazonaws.com:8443/things/
what_the_thing_is_named/shadow: net/http: request canceled (Client.Timeout exceeded
while awaiting headers)
[2017-07-20T10:01:58.006Z][WARN]-sync_manager.go:263,Failed to get cloud
copy: Get https://1234567890abcd.iot.us-west-2.amazonaws.com:8443/things/
what_the_thing_is_named/shadow: net/http: request canceled (Client.Timeout exceeded
while awaiting headers)
[2017-07-20T10:01:58.006Z][ERROR]-sync_manager.go:375,Failed to execute sync operation
{what_the_thing_is_named VersionDiscontinued []}"
```

Una possibile soluzione consiste nel configurare la quantità di tempo che il dispositivo core deve attendere per ricevere una risposta dell'host. Apri il file [config.json](#) in *greengrass-root*/config e aggiungi un campo `system.shadowSyncTimeout` con un valore di timeout in secondi. Per esempio:

```
{
  "system": {
    "shadowSyncTimeout": 10
  },
  "coreThing": {
    "caPath": "root-ca.pem",
    "certPath": "cloud.pem.crt",
    "keyPath": "cloud.pem.key",
    ...
  },
  ...
}
```

Se non è specificato alcun valore `shadowSyncTimeout` in `config.json`, il valore predefinito è 5 secondi.

 Note

Per il software AWS IoT Greengrass Core v1.6 e versioni precedenti, l'impostazione predefinita `shadowSyncTimeout` è 1 secondo.

Seleziona Re:post AWS

Se non riesci a risolvere il problema utilizzando le informazioni sulla risoluzione dei problemi contenute in questo argomento, puoi cercare [Risoluzione dei problemi](#) o controllare il [AWS IoT Greengrass tag su AWS re:post](#) per eventuali problemi correlati o pubblicare una nuova domanda. I membri del AWS IoT Greengrass team AWS monitorano attivamente Re:POST.

Cronologia dei documenti per AWS IoT Greengrass

La tabella seguente descrive importanti modifiche alla Developer Guide dopo giugno 2018. AWS IoT Greengrass Per ricevere notifiche sugli aggiornamenti di questa documentazione, è possibile sottoscrivere un feed RSS.

Modifica	Descrizione	Data
Aggiornamento alla fine del supporto per Snap v1.11.x	Sono state aggiornate le informazioni sulla fine del supporto per AWS IoT Greengrass core v 1.11.x Snap su snapcraft.io.	22 settembre 2023
Fine del supporto per Snap v1.11.x	Sono state aggiunte informazioni sulla fine del supporto per AWS IoT Greengrass core v 1.11.x Snap su snapcraft.io.	19 settembre 2023
Immagini AWS IoT Greengrass Docker per v1.11.6	Le immagini Docker per il software AWS IoT Greengrass Core v1.11.6 sono disponibili su Amazon Elastic Container Registry (Amazon ECR) e Docker Hub. Ti consigliamo di utilizzare sempre la versione più recente.	12 aprile 2022
AWS IoT Device Tester (IDT) per la deprecazione AWS IoT Greengrass V1	IDT per AWS IoT Greengrass V1 non genererà più report di qualificazione firmati.	4 aprile 2022
Aggiornamento di supporto per AWS IoT Device Tester per AWS IoT Greengrass	IDT per la AWS IoT Greengrass versione 4.4.1 ora supporta l'utilizzo della versione software di AWS IoT	24 marzo 2022

Greengrass base v1.11.6 per la qualificazione dei dispositivi.

[AWS IoT Greengrass è stata rilasciata la versione 1.11.6](#)

È disponibile la versione 1.11.6 del software AWS IoT Greengrass Core. Questa versione contiene i miglioramenti delle prestazioni e le correzioni di bug. Ti consigliamo di utilizzare sempre la versione più recente.

24 marzo 2022

[Rilasciata la versione 12 del SiteWise connettore IoT](#)

È disponibile la versione 12 del SiteWise connettore IoT. Questa versione contiene correzioni di bug.

23 dicembre 2021

[Immagini Docker per AWS IoT Greengrass v1.11.5 e v1.10.5](#)

Le immagini Docker per il software AWS IoT Greengrass Core v1.11.5 e v1.10.5 sono disponibili su Amazon Elastic Container Registry (Amazon ECR) e Docker Hub. Ti consigliamo di eseguire sempre la versione più recente.

22 dicembre 2021

[AWS IoT Greengrass V1 politica di manutenzione](#)

La politica di AWS IoT Greengrass V1 manutenzione definisce i diversi livelli di manutenzione e aggiornamento per il AWS IoT Greengrass V1 servizio e il software di AWS IoT Greengrass base v1.x.

20 dicembre 2021

[AWS IoT Rilasciata la versione 4.4.1 di Device Tester](#)

IDT per la AWS IoT Greengrass versione 4.4.1 è ora disponibile. Questa versione include la suite di AWS IoT Greengrass qualificazione (GGQ) v1.3.1 e supporta l'utilizzo delle versioni software AWS IoT Greengrass principali v1.11.5 e v1.10.5 per la qualificazione dei dispositivi.

[AWS IoT Greengrass rilasciate le versioni 1.11.5 e 1.10.5](#)

Sono disponibili le versioni 1.11.5 e 1.10.5 del software Core. AWS IoT Greengrass. Queste versioni contengono miglioramenti delle prestazioni e correzioni di bug. Ti consigliamo di utilizzare sempre la versione più recente.

[Immagini Docker AWS IoT Greengrass v1.11.4 e v1.10.4 ripubblicate](#)

Le immagini Docker per le versioni del software AWS IoT Greengrass Core 1.11.4 e 1.10.4 sono state ripubblicate su Amazon Elastic Container Registry (Amazon ECR) e Docker Hub per risolvere le correzioni di bug. BusyBox. Per utilizzare le immagini Docker più recenti, usa i tag o. 1.11.4-1 1.10.4-1 Per ulteriori informazioni sui tag disponibili, consulta [amazon/aws-iot-greengrass](#) in Docker Hub.

CloudWatch Il connettore Metrics supporta timestamp duplicati nei dati di input	Ora puoi inviare dati di input con timestamp duplicati a questo connettore.	19 novembre 2021
Aggiornamento della prevenzione del "confused deputy" tra servizi	AWS IoT Greengrass supporta l'utilizzo delle aws:SourceArn chiavi di contesto delle condizioni aws:SourceAccount globali nelle politiche delle risorse IAM per prevenire il confuso problema del vice.	1° novembre 2021
Immagini Docker per AWS IoT Greengrass v1.11.4	Le immagini Docker per il software AWS IoT Greengrass Core v1.11.4 sono disponibili su Amazon Elastic Container Registry (Amazon ECR) e Docker Hub. Ti consigliamo di utilizzare sempre la versione più recente.	24 agosto 2021
Snap pubblicato nella AWS IoT Greengrass versione 1.11.4	È disponibile la versione 1.11.4 dello snap. AWS IoT Greengrass Ti consigliamo di eseguire sempre la versione più recente.	20 agosto 2021
Aggiornamento di supporto per AWS IoT Device Tester per AWS IoT Greengrass	IDT per la AWS IoT Greengrass versione 4.1.0 ora supporta l'utilizzo della versione software di AWS IoT Greengrass base v1.11.4 per la qualificazione dei dispositivi.	18 agosto 2021

[AWS IoT Greengrass è stata rilasciata la versione 1.11.4](#)

È disponibile la versione 1.11.4 del software AWS IoT Greengrass Core. Questa versione corregge un problema con lo stream manager che impediva gli aggiornamenti alla v1.11.3 da una versione precedente del software Core. AWS IoT Greengrass Ti consigliamo di utilizzare sempre la versione più recente.

17 agosto 2021

[Endpoint VPC \(AWS PrivateLink\)](#)

AWS IoT Greengrass ora supporta l'interfaccia VPC endpoints (AWS PrivateLink) per il AWS IoT Greengrass piano di controllo. Puoi stabilire una connessione privata tra il tuo VPC e il piano di AWS IoT Greengrass controllo.

16 agosto 2021

[AWS IoT Rilasciata la versione 4.1.0 di Device Tester](#)

È disponibile la versione 4.1.0 di AWS IoT Device Tester for. AWS IoT Greengrass Questa versione supporta l'utilizzo delle versioni software AWS IoT Greengrass principali 1.11.3 e 1.10.4 per la qualificazione dei dispositivi.

23 giugno 2021

[Snap pubblicato nella AWS IoT Greengrass versione 1.11.3](#)

La versione 1.11.3 dello AWS IoT Greengrass snap contiene miglioramenti delle prestazioni e correzioni di errori. Come best practice, consigliamo di utilizzare sempre la versione più recente.

15 giugno 2021

[Rilasciate le immagini Docker per la versione AWS IoT Greengrass 1.11.3 e la versione 1.10.4](#)

Le immagini Docker per il software AWS IoT Greengrass Core v1.11.3 e v1.10.4 sono disponibili su Amazon Elastic Container Registry (Amazon ECR) e Docker Hub. Queste versioni di Core contengono miglioramenti delle prestazioni e correzioni di bug. AWS IoT Greengrass Ti consigliamo di utilizzare sempre la versione più recente.

15 giugno 2021

[AWS IoT Greengrass è stata rilasciata la versione 1.11.3](#)

È disponibile la versione 1.11.3 del software AWS IoT Greengrass Core. Questa versione contiene i miglioramenti delle prestazioni e le correzioni di bug. Ti consigliamo di utilizzare sempre la versione più recente.

14 giugno 2021

[AWS IoT Greengrass è stata rilasciata la versione 1.10.4](#)

È disponibile la versione 1.10.4 del software AWS IoT Greengrass Core. Questa versione contiene i miglioramenti delle prestazioni e le correzioni di bug. Ti consigliamo di utilizzare sempre la versione più recente.

14 giugno 2021

[È stata rilasciata la versione 2 dell'adattatore di protocollo Modbus-TCP](#)

È disponibile la versione 2 del connettore Modbus-TCP Protocol Adapter. Questa versione ha aggiunto il supporto per ASCII e le stringhe sorgente UTF8 codificate. ISO8859

24 maggio 2021

[È stata rilasciata la versione 7 del Docker Application Deployment Connector](#)

È disponibile la versione 7 del connettore di distribuzione delle applicazioni Greengrass Docker.

5 aprile 2021

[AWS IoT Greengrass è stata rilasciata la versione 1.11.1](#)

È disponibile la versione 1.11.1 del software AWS IoT Greengrass Core. Questa versione contiene i miglioramenti delle prestazioni e le correzioni di bug. Ti consigliamo di utilizzare sempre la versione più recente.

29 marzo 2021

[AWS IoT Rilasciata la versione 4.0.2 di Device Tester](#)

È disponibile la versione 4.0.2 di AWS IoT Device Tester for. AWS IoT Greengrass Questa versione sostituisce IDT v4.0.0 e aggiunge il supporto per la versione 1.11.1 del software Core. AWS IoT Greengrass Questo risolve anche un problema che faceva sì che IDT mascherasse gli errori di Hardware Security Integration (HSI).

29 marzo 2021

[Rilasciata la versione 11 del SiteWise connettore IoT](#)

È disponibile la versione 11 del SiteWise connettore IoT. Questo avvia il supporto per le stringhe che contengono caratteri nascosti o non stampabili. Questa versione include anche miglioramenti generali delle prestazioni e correzioni di bug.

24 marzo 2021

[Snap v1.11.0 ripubblicato AWS IoT Greengrass](#)

AWS IoT Greengrass La versione 1.11.0 di snap è stata ripubblicata su Snapcraft per risolvere correzioni di bug e un possibile arresto anomalo dell'applicazione quando si utilizza l'interprete Python. AWS IoT Greengrass non fornisce snap per le versioni software 1.10 e 1.9.

19 marzo 2021

<u>Aggiornamento di supporto per AWS IoT Device Tester per AWS IoT Greengrass</u>	IDT per la AWS IoT Greengrass versione 4.0.0 ora supporta l'utilizzo della versione software di AWS IoT Greengrass base v1.10.3 per la qualificazione dei dispositivi.	18 marzo 2021
<u>AWS IoT Greengrass Snap v1.8.4 ripubblicato</u>	AWS IoT Greengrass La versione 1.8.4 di snap è stata ripubblicata su Snapcraft per risolvere correzioni di bug e un possibile arresto anomalo dell'applicazione quando si utilizza l'interprete Python.	15 marzo 2021
<u>AWS IoT Greengrass Immagine Docker v1.11.0 ripubblicata per ARMv7l</u>	L'immagine Docker per la versione 1.11.0 del software AWS IoT Greengrass Core per la piattaforma ARMv7l è stata ripubblicata su Amazon Elastic Container Registry (Amazon ECR) e Docker Hub per risolvere correzioni di bug e un possibile arresto anomalo dell'applicazione quando si utilizza l'interprete Python.	8 marzo 2021
<u>AWS IoT Greengrass Rilasciati e le immagini Docker v1.10.3</u>	Le immagini Docker per la versione 1.10.3 del software AWS IoT Greengrass Core sono ora disponibili su Amazon Elastic Container Registry (Amazon ECR) e Docker Hub.	8 marzo 2021

[Immagini AWS IoT Greengrass Docker ripubblicate nelle versioni 1.11.0 e 1.9.4](#)

Le immagini Docker per le versioni del software AWS IoT Greengrass Core 1.11.0 e 1.9.4 sono state ripubblicate su Amazon Elastic Container Registry (Amazon ECR) e Docker Hub per risolvere correzioni di bug e un possibile arresto anomalo dell'applicazione quando si utilizza l'interprete Python. Le immagini Docker per ARMv7l non sono state ripubblicate al momento.

26 febbraio 2021

[AWS IoT Greengrass è stata rilasciata la versione 1.10.3](#)

È disponibile la versione 1.10.3 del software AWS IoT Greengrass Core. Questa versione aggiunge la proprietà di configurazione `systemComponentAutoTimeout` principale e contiene miglioramenti delle prestazioni e correzioni di bug. Ti consigliamo di utilizzare sempre la versione più recente.

24 febbraio 2021

[Rilasciata la versione 10 del SiteWise connettore IoT](#)

È disponibile la versione 10 del SiteWise connettore IoT. Questa versione risolve i problemi di stabilità del StreamManager client in caso di interruzione della connessione e migliora la gestione dei valori OPC-UA quando è assente. SourceTimestamp Utilizza il SiteWise connettore IoT per inviare i dati di dispositivi e apparecchiature locali alle proprietà degli asset in IoT SiteWise.

22 gennaio 2021

[Rilasciata la versione 9 del SiteWise connettore IoT](#)

È disponibile la versione 9 del SiteWise connettore IoT. Ciò avvia il supporto per destinazioni di streaming StreamManager Greengrass personalizzate, deadbanding OPC-UA, modalità di scansione personalizzata e velocità di scansione personalizzata. Ciò include anche prestazioni migliorate durante gli aggiornamenti della configurazione effettuati dal SiteWise gateway IoT. Utilizza il SiteWise connettore IoT per inviare i dati di dispositivi e apparecchiature locali alle proprietà degli asset in IoT SiteWise.

15 dicembre 2020

[AWS IoT Rilasciata la versione 4.0.0 di Device Tester](#)

È disponibile la versione 4.0.0 di AWS IoT Device Tester per AWS IoT Greengrass. Questa versione consente di utilizzare IDT per sviluppare ed eseguire suite di test personalizzate per la convalida dei dispositivi. Sono incluse anche le applicazioni IDT con firma di codice per macOS e Windows.

15 dicembre 2020

[AWS IoT Greengrass snap v1.11](#)

La versione 1.11.0 dello AWS IoT Greengrass snap supporta funzioni Lambda non containerizzate. Come best practice, consigliamo di utilizzare sempre la versione più recente.

6 dicembre 2020

[Rilasciata la versione 8 del SiteWise connettore IoT](#)

È disponibile la versione 8 del SiteWise connettore IoT. Questa versione migliora la stabilità quando il connettore presenta una connettività di rete intermittente. Utilizza il SiteWise connettore IoT per inviare i dati di dispositivi e apparecchiature locali alle proprietà degli asset in IoT SiteWise.

19 novembre 2020

[Il connettore Kinesis Firehose supporta la modalità No container](#)

È possibile utilizzare il `IsolationMode` parametro per configurare la modalità di containerizzazione per il connettore.

19 ottobre 2020

[Rilasciata la versione 6 del Docker Application Deployment Connector](#)

È disponibile la versione 6 del connettore di distribuzione delle applicazioni Greengrass Docker.

18 settembre 2020

[AWS IoT Greengrass è stata rilasciata la versione 1.11.0](#)

È disponibile la versione 1.11.0 del software AWS IoT Greengrass Core. Questa versione aggiunge la funzionalità di telemetria sanitaria del sistema e un'API di controllo sanitario locale. Stream Manager ora può esportare dati su Amazon Simple Storage Service (Amazon S3) e IoT. SiteWise Questa versione contiene anche miglioramenti delle prestazioni e correzioni di bug. Ti consigliamo di utilizzare sempre la versione più recente.

16 settembre 2020

[Rilasciata la versione 7 del SiteWise connettore IoT](#)

È disponibile la versione 7 del SiteWise connettore IoT. Questa versione corregge un problema relativo alle metriche del gateway. Utilizza il SiteWise connettore IoT per inviare i dati di dispositivi e apparecchiature locali alle proprietà degli asset in IoT SiteWise.

14 agosto 2020

[ServiceNow MetricBase I connettori Integration, Splunk Integration e Twilio Notifications supportano la modalità No container](#)

È possibile utilizzare il `IsolationMode` parametro per configurare la modalità di containerizzazione per il connettore.

30 luglio 2020

[Il connettore SNS supporta la modalità No container](#)

È possibile utilizzare il `IsolationMode` parametro per configurare la modalità di containerizzazione per il connettore.

6 luglio 2020

[CloudWatch Il connettore Metrics supporta la modalità No container](#)

È possibile utilizzare il `IsolationMode` parametro per configurare la modalità di containerizzazione per il connettore.

17 giugno 2020

[AWS IoT Greengrass è stata rilasciata la versione 1.10.2](#)

È disponibile la versione 1.10.2 del software AWS IoT Greengrass Core. Questa versione aggiunge la proprietà di configurazione `mqttOperationTimeout` principale e contiene miglioramenti delle prestazioni e correzioni di bug. Ti consigliamo di utilizzare sempre la versione più recente.

8 giugno 2020

[Gli installatori di machine learning di Tensorflow sono obsoleti](#)

AWS IoT Greengrass Gli installatori preconfezionati di machine learning di TensorFlow sono obsoleti. Esempi di Machine Learning sono stati aggiornati a Python 3.7.

29 maggio 2020

[Il supporto del framework Chainer e gli installatori di machine learning Greengrass sono obsoleti](#)

AWS IoT Greengrass i programmi di installazione e download preconfezionati di machine learning per e DLR sono obsoleti. MXNet Il supporto del framework Chainer e i download associati sono obsoleti.

4 maggio 2020

[Rilasciata la versione 6 del SiteWise connettore IoT](#)

È disponibile la versione 6 del SiteWise connettore IoT. Questa versione aggiunge il supporto per le CloudWatch metriche e l'individuazione automatica di nuovi tag OPC-UA. Ciò significa che non è necessario riavviare il gateway quando i tag cambiano per le sorgenti OPC-UA. Questa versione del connettore richiede lo stream manager e il software AWS IoT Greengrass Core v1.10.0 o superiore. Utilizza il SiteWise connettore IoT per inviare i dati di dispositivi e apparecchiature locali alle proprietà degli asset in IoT SiteWise.

29 aprile 2020

[Connettori aggiornati a Python 3.7](#)

I connettori che supportano il runtime Python sono stati aggiornati a Python 3.7. Si consiglia di aggiornare le versioni dei connettori da Python 2.7 a Python 3.7.

29 aprile 2020

<u>La configurazione del dispositivo Greengrass può essere eseguita in modalità silenziosa</u>	Puoi eseguire l'impostazione del dispositivo Greengrass in modalità silenziosa in modo che lo script non richieda alcun valore.	27 aprile 2020
<u>Nuove immagini di base Docker</u>	È possibile scaricare immagini AWS IoT Greengrass Docker basate su immagini di base di Alpine Linux (x86_64, ARMv7L o). AArch64	23 aprile 2020
<u>AWS IoT Greengrass è stata rilasciata la versione 1.10.1</u>	È disponibile la versione 1.10.1 del software AWS IoT Greengrass Core. Questa versione contiene i miglioramenti delle prestazioni e le correzioni di bug. Ti consigliamo di utilizzare sempre la versione più recente.	16 aprile 2020
<u>Nuovo capitolo sulla sicurezza</u>	AWS IoT Greengrass i contenuti di sicurezza sono stati riorganizzati, con l'aggiunta di nuove informazioni.	30 marzo 2020
<u>Usa il gestore di pacchetti APT per l'installazione AWS IoT Greengrass</u>	Sulle distribuzioni Linux basate su Debian supportate, puoi usarlo apt per installare il software AWS IoT Greengrass Core sui tuoi dispositivi.	26 febbraio 2020

[Rilasciata la versione 5 del SiteWise connettore IoT](#)

È disponibile la versione 5 del SiteWise connettore IoT. Questa versione corregge un problema di compatibilità con il software AWS IoT Greengrass Core v1.9.4. Utilizza il SiteWise connettore IoT per inviare i dati di dispositivi e apparecchiature locali alle proprietà degli asset in IoT SiteWise.

12 febbraio 2020

[Nuovo script per configurare rapidamente un dispositivo principale](#)

È possibile utilizzare la configurazione del dispositivo Greengrass per configurare il dispositivo core in pochi minuti. Inoltre, AWS IoT Greengrass ora supporta le funzioni Lambda di Node.js 12.x.

20 dicembre 2019

[AWS IoT Greengrass è stata rilasciata la versione 1.10.0](#)

È disponibile la versione 1.10.0 del software AWS IoT Greengrass Core. Le nuove funzionalità di questa versione includono Stream manager, supporto per container con il connettore di distribuzione delle applicazioni Docker, funzioni Lambda non containerizzate che possono accedere a risorse di machine learning, supporto per sessioni persistenti MQTT AWS IoT con e supporto per il traffico MQTT locale su una porta specifica.

25 novembre 2019

[Supporto da console per le notifiche di distribuzione](#)

Usa la EventBridge console Amazon per creare regole di evento che si attivano quando le distribuzioni del gruppo Greengrass cambiano stato.

14 novembre 2019

[AWS IoT Greengrass è stata rilasciata la versione 1.9.4](#)

È disponibile la versione 1.9.4 del software AWS IoT Greengrass Core. Questa versione contiene i miglioramenti delle prestazioni e le correzioni di bug. Come best practice, ti consigliamo di eseguire sempre la versione più recente.

17 ottobre 2019

[Supporto da console per la gestione del ruolo di servizio Greengrass](#)

Usa le funzionalità nuove e migliorate della AWS IoT console per gestire il tuo ruolo di servizio Greengrass.

4 ottobre 2019

[Supporto da console per la gestione dei tag a livello di gruppo](#)

Puoi creare, visualizzare e gestire i tag per i tuoi gruppi Greengrass nella AWS IoT console.

23 settembre 2019

[Nuovi connettori per l'apprendimento automatico](#)

Utilizza il connettore ML Feedback per pubblicare input e previsioni del modello e il connettore ML Object Detection per eseguire un servizio di inferenza di rilevamento dell'oggetto locale.

19 settembre 2019

[AWS IoT Greengrass rilasciata la versione 1.9.3](#)

È disponibile la versione 1.9.3 del software AWS IoT Greengrass Core. Questa versione consente di installare il software AWS IoT Greengrass Core sulle distribuzioni Raspbian su architetture ARMv6L, supporta gli aggiornamenti OTA sulla porta 443 con ALPN e contiene una correzione di bug per i payload binari inviati dalle funzioni Lambda di Python 2.7 ad altre funzioni Lambda.

12 settembre 2019

[AWS IoT Greengrass è stata rilasciata la versione 1.8.4](#)

È disponibile la versione 1.8.4 del software AWS IoT Greengrass Core. Questa versione contiene i miglioramenti delle prestazioni e le correzioni di bug. Se esegui v1.8.x, ti consigliamo di effettuare l'upgrade a v1.8.4 o v1.9.3. Per le versioni precedenti, ti consigliamo di effettuare l'upgrade a v1.9.3.

30 agosto 2019

[AWS IoT Greengrass è stata rilasciata la versione 1.9.2 con supporto per OpenWrt](#)

È disponibile la versione 1.9.2 del software AWS IoT Greengrass Core. Questa versione consente di installare il software AWS IoT Greengrass Core su OpenWrt distribuzioni con architetture Armv8 () AArch64 e ARMv7l.

20 giugno 2019

[AWS IoT Greengrass è stata rilasciata la versione 1.8.3](#)

È disponibile la versione 1.8.3 del software AWS IoT Greengrass Core. Questa versione contiene i miglioramenti generali delle prestazioni e le correzioni di bug. Se esegui v1.8.x, ti consigliamo di effettuare l'upgrade a v1.8.3 o v1.9.2. Per le versioni precedenti, consigliamo l'upgrade a v1.9.2.

20 giugno 2019

[AWS IoT Greengrass è stata
rilasciata la versione 1.9.1](#)

È disponibile la versione 1.9.1 del software AWS IoT Greengrass Core. Questa versione contiene una correzione di bug per i messaggi AWS IoT che contengono un carattere jolly nell'argomento.

10 maggio 2019

[AWS IoT Greengrass è stata
rilasciata la versione 1.8.2](#)

È disponibile la versione 1.8.2 del software AWS IoT Greengrass Core. Questa versione contiene i miglioramenti generali delle prestazioni e le correzioni di bug. Se esegui v1.8.x, ti consigliamo di effettuare l'upgrade a v1.8.2 o v1.9.0. Per le versioni precedenti, consigliamo l'upgrade a v1.9.0.

2 maggio 2019

[AWS IoT Greengrass è stata
rilasciata la versione 1.9.0](#)

Nuove funzionalità: supporto per i runtime Lambda di Python 3.7 e Node.js 8.10, connessioni MQTT ottimizzate e supporto chiave Elliptic Curve (EC) per il server MQTT locale.

1 maggio 2019

[AWS IoT Greengrass è stata rilasciata la versione 1.8.1](#)

È disponibile la versione 1.8.1 del software AWS IoT Greengrass Core. Questa versione contiene i miglioramenti generali delle prestazioni e le correzioni di bug. Come best practice, ti consigliamo di eseguire sempre la versione più recente.

18 Aprile 2019

[AWS IoT Greengrass snap disponibile su snapcraft](#)

Usa l'app AWS IoT Greengrass Snap Store per progettare, testare e distribuire rapidamente software su dispositivi Linux con AWS IoT Greengrass.

1 Aprile 2019

[Support per un maggiore controllo degli accessi tramite autorizzazioni basate su tag](#)

Puoi utilizzare i tag nelle policy AWS Identity and Access Management (IAM) per controllare l'accesso alle tue AWS IoT Greengrass risorse.

29 marzo 2019

[Rilasciato il connettore IoT Analytics](#)

Utilizza il connettore IoT Analytics per inviare i dati dei dispositivi locali ai AWS IoT Analytics canali.

15 marzo 2019

[Supporto in batch nel connettore Kinesis Firehose](#)

Il connettore Kinesis Firehose supporta l'invio di record di dati in batch ad Amazon Data Firehose a intervalli specificati.

15 marzo 2019

[AWS CloudFormationAWS IoT Greengrass supporto per le risorse](#)

Utilizza AWS CloudFormation modelli per creare e gestire AWS IoT Greengrass risorse.

15 marzo 2019

[AWS IoT Greengrass rilasciata la versione 1.8.0](#)

Nuove funzionalità: identità di accesso predefinita configurabile per le funzioni Lambda, supporto per il traffico HTTPS sulla porta 443 e client con nome IDs prevedibile per le connessioni MQTT con. AWS IoT

7 marzo 2019

[AWS IoT Greengrass rilasciate le versioni 1.7.1 e 1.6.1](#)

Sono disponibili le versioni 1.7.1 e 1.6.1 del software Core. AWS IoT Greengrass Queste versioni richiedono il kernel Linux versione 3.17 o successive. È consigliabile che i clienti che eseguono qualsiasi versione del software core Greengrass eseguano immediatamente l'aggiornamento alla versione 1.7.1.

11 febbraio 2019

[SageMaker Runtime di deep learning AI Neo](#)

Il runtime di deep learning SageMaker AI Neo supporta modelli di machine learning ottimizzati dal compilatore di deep learning SageMaker AI Neo.

28 novembre 2018

[Esegui AWS IoT Greengrass in un contenitore Docker](#)

Puoi eseguirlo AWS IoT Greengrass in un contenitore Docker configurando il tuo gruppo Greengrass per l'esecuzione senza containerizzazione.

26 novembre 2018

[AWS IoT Greengrass è stata rilasciata la versione 1.7.0](#)

Nuove funzionalità: connettori Greengrass, gestore di segreti locali, impostazioni di isolamento e autorizzazione per le funzioni Lambda, sicurezza hardware root of trust, connessione tramite ALPN o proxy di rete e supporto Raspbian Stretch.

26 novembre 2018

[AWS IoT Greengrass download di software](#)

I pacchetti AWS IoT Greengrass Core software, AWS IoT Greengrass Core SDK e AWS IoT Greengrass Machine Learning SDK sono disponibili per il download tramite Amazon. CloudFront

26 novembre 2018

[AWS IoT Device Tester per AWS IoT Greengrass](#)

Usa AWS IoT Device Tester per AWS IoT Greengrass verificare che l'architettura della CPU, la configurazione del kernel e i driver funzionino con. AWS IoT Greengrass

26 novembre 2018

[AWS CloudTrail registrazione per le chiamate API AWS IoT Greengrass](#)

AWS IoT Greengrass è integrato con AWS CloudTrail, un servizio che fornisce una registrazione delle azioni intraprese da un utente, un ruolo o un AWS servizio in AWS IoT Greengrass.

29 ottobre 2018

[Support per la versione
TensorFlow 1.10.1 su NVIDIA
Jetson TX2](#)

La libreria TensorFlow precompilata per NVIDIA Jetson che fornisce ora utilizza la versione 1.10.1. TX2 AWS IoT Greengrass TensorFlow Questo supporta Jetpack 3.3 e CUDA Toolkit 9.0.

18 ottobre 2018

[Support per le MXNet risorse
di machine learning della
versione 1.2.1](#)

AWS IoT Greengrass supporta modelli di apprendimento automatico addestrati utilizzando MXNet la versione 1.2.1.

29 agosto 2018

[AWS IoT Greengrass è stata
rilasciata la versione 1.6.0](#)

Nuove funzionalità: eseguibile Lambda, coda di messaggi configurabile, intervallo di tentativi di riconnessione configurabile, risorse di volume in /proc e directory di scrittura configurabile.

26 luglio 2018

Aggiornamenti precedenti

La tabella seguente descrive importanti modifiche alla Developer Guide prima di luglio 2018. AWS IoT Greengrass

Modifica	Descrizione	Data
AWS IoT Greengrass Rilasciata la versione 1.5.0	Nuove caratteristiche: - Inferenza Machine Learning locale che usa modelli qualificati per il cloud. Per ulteriori informazioni, consulta Esecuzione dell'inferenza di Machine Learning. - Le funzioni Greengrass Lambda supportano dati di input binari, oltre a JSON.	29 marzo 2018

Modifica	Descrizione	Data
	Per ulteriori informazioni, consulta Versioni di AWS IoT Greengrass Core .	
AWS IoT Greengrass Rilasciata la versione 1.3.0	Nuove caratteristiche: • Over-the-air Agente di aggiornamento (OTA) in grado di gestire i processi di aggiornamento Greengrass distribuiti sul cloud. Per ulteriori informazioni, consulta Aggiornamenti OTA del software AWS IoT Greengrass Core. • Accedi alle periferiche e alle risorse locali dalle funzioni Greengrass Lambda. Per ulteriori informazioni, consulta Accedi alle risorse locali con funzioni e connettori Lambda.	27 Novembre 2017
AWS IoT Greengrass Rilasciata la versione 1.1.0	Nuove caratteristiche: • Reimposta i gruppi distribuiti AWS IoT Greengrass . Per ulteriori informazioni, consulta Reimpostazione delle distribuzioni. • Support per i runtime Node.js 6.10 e Java 8 Lambda, oltre a Python 2.7.	20 settembre 2017
AWS IoT Greengrass Rilasciata la versione 1.0.0	AWS IoT Greengrass è disponibile a livello generale.	7 giugno 2017